Aalto University
School of Science
Master's Programme in Life Science Technologies

Toni Tamminen

Software Implementation for Temperature-Controlled Retinal Pigment Epithelium Heating Device

Master's Thesis

Espoo, August 21, 2018

Supervisor: Professor Ari Koskelainen

Thesis advisor(s): Marja Pitkänen, M.Sc. (Tech)

Aalto University
School of Science
Master's Programme in Life Science Technologies

ABSTRACT OF THE
MASTER'S THESIS

Author Toni Tamminen

Title of thesis Software Implementation for Temperature-Controlled Retinal Pigment Epithelium Heating Device

Master's programme Life Science Technologies

Thesis supervisor Prof. Ari Koskelainen

Major / Code Biomedical Engineering / SCI3059

Department Neuroscience and Biomedical Engineering

Thesis advisor(s) Marja Pitkänen M.Sc. (Tech)

Date August 21, 2018 **Number of pages** 51+40 **Language** English

Age-related macular degeneration (AMD) was the leading cause for unavoidable blindness in 2010 and continues to affect an estimated 150 million people worldwide. It has been suggested that heating the retinal pigment epithelium (RPE) could slow down the progress of the disease or even cure it entirely. The treatment consists of heating the retina of an eye to therapeutic temperatures to inflict the generation of heat-shock proteins (HSPs).

A device that relies on electroretinogram (ERG) recordings while inflicting local hyperthermia on the RPE has been developed in our research team. The measured ERG responses can be characterised and shown a direct dependency to the experienced temperature at the retina. These in turn are utilised in controlling the heating of the retina to therapeutic temperatures. The aim of this thesis was to implement a new software for the use of such a device, while considering the needs this software must account for to facilitate a reliable, safe and useful software interaction for the RPE heating device. Eight distinct requirements for the new software were identified: maintainability; dynamicity; accuracy and precision; pulse sequences; automation; safety, error handling and user friendliness; testing and validation; as well as documentation. The software was implemented with National Instruments LabVIEW™ and MathWorks MATLAB®. The results were validated and verified with unit testing, bench testing and in a full experiment on a mouse subject.

The bench testing and mouse experiment testing provided satisfying results. The software functioned without errors during both types of testing or only had very minute types of errors. The software could still be developed to contain more automation, such as factoring in safety features through eye movement detection and more importantly facilitating feedback-controlled heating through a PID controller, which would be of importance when planning clinical trials and use of the device in treatment of AMD.

Keywords Electroretinogram, ERG, retinal heating, retinal pigment epithelium heating

Aalto-yliopisto
Perustieteiden korkeakoulu
Master's Programme in Life Science Technologies

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä Toni Tamminen

Työn nimi Ohjelmistokehitys lämpötilakontrolloidun silmänpohjan epiteelisolujen lämmityslaitetta varten

Koulutusohjelma Life Science Technologies

Valvoja Prof. Ari Koskelainen

Pääaine / koodi Biomedical Engineering / SCI3059

Työn ohjaaja(t) DI Marja Pitkänen

Päivämäärä 21.08.2018 **Sivumäärä** 51+40 **Kieli** englanti

Silmänpohjan ikärappeuma (AMD) oli yleisin vailla parannuskeinoa oleva, sokeutta aiheuttava sairaus vuonna 2010, ja se vaikuttaa noin 150 miljoonan ihmisen elämään maailmanlaajuisesti. Kirjallisuudessa on esitetty, että silmänpohjan pigmenttiepiteelikerroksen (RPE) lämmittäminen voisi hidastaa taudin kulkua tai parantaa sen kokonaan. Tällainen hoito saavutettaisiin lämmittämällä silmänpohjaa terapeuttisiin lämpötiloihin, jolloin saadaan aikaan lämpösokkiproteiinien (HSP) muodostumista.

Tutkimusryhmässämme on kehitetty laite, joka perustuu elektroretinogrammin (ERG) rekisteröintiin, samalla kun lämmityslaserilla aiheutetaan RPE:lle paikallinen hypertermia. Talteenotetulla ERG:llä voidaan estimoida silmänpohjan lämpötilaa. Estimoitua lämpötilaa hyödynnetään lämmityslaserin säätämisessä terapeuttiselle lämpötila-alueelle. Tämän diplomityön tavoitteena oli implementoida uusi ohjelmisto kyseistä laitetta varten, samalla ottaen huomioon luotettavuus- ja turvallisuusnäkökulmia, sekä muita hyödyllisiä ominaisuuksia laitteen ja ohjelmiston yhteistoiminnassa. Työssä määriteltiin kahdeksan eri vaatimusta uudelle ohjelmistolle: ylläpidettävyys; dynaamisuus; tarkkuus ja täsmällisyys; pulssisekvenssit; automaatio; turvallisuus, virheiden käsittely ja käyttäjäystävällisyys; verifiointi ja validointi; sekä dokumentointi. Ohjelmisto toteutettiin käyttäen ohjelmistoja: National Instruments LabVIEW™ sekä MathWorks MATLAB®. Ohjelmisto validoitiin testaamalla kaikki osiot erikseen (yksikkötestaus) sekä koko ohjelmisto mittausta simuloivassa tilanteessa. Lopuksi ohjelmisto testattiin myös oikeassa hiiren silmänpohjan lämmityskokeessa.

Testauksissa ohjelmisto toimi halutulla tavalla ja esiintyneet virheet pystyttiin nopeasti korjaamaan viimeistä versiota varten. Ohjelmistoa voidaan jatkossa kehittää sisältämään enemmän automaatiota, kuten turvallisuusominaisuuksia silmän liikkeiden tunnistamiseen sekä lämmityksen säätämiseen takaisinkytkentämenetelmällä. Molemmat olisivat tärkeitä ominaisuuksia siirryttäessä kliiniseen tutkimukseen ja laitteen kliiniseen käyttöön AMD:n hoitamiseksi.

Avainsanat Elektroretinogrammi, ERG, silmänpohjan lämmitys, silmänpohjan pigmenttiepiteelin lämmitys

Acknowledgements

The work of this thesis took place at the Department of Neuroscience and Biomedical Engineering, Aalto University.

I would like to thank my thesis advisor Marja Pitkänen for her great support by giving constructive feedback and valuable advice throughout the writing process. In addition, I would also like to express my gratitude to Professor Ari Koskelainen for giving me the opportunity to work on this particular topic and for his valuable recommendations. My thanks also go to my co-workers for their support and for making my day after sometimes having spent the entire day testing the software in a windowless ERG recording room. Finally, I appreciate all the support of my friends and family throughout the process. Studying a degree in medicine simultaneously with the task of writing this thesis would not have been possible if it was not for the support of all these people.

Espoo, August 21, 2018

Toni Tamminen

Table of Contents

Abbreviations and Units

Abbreviations

ADC	Analogue-to-digital converter
AMD	Age-related macular degeneration
DAQ	Data acquisition
ERG	Electroretinogram
fERG	Flicker electroretinogram
FFT	Fast-Fourier transform
FIFO	First-in, first-out
GPU	Graphics processing unit
HSP	Heat shock protein
INL	Inner nuclear layer
I/O	Input/output
IR	Infrared
LED	Light emitting diode
LSB	Least significant bit
Mux	Multiplexer
ONL	Outer nuclear layer
PWM	Pulse-width modulation
RAM	Random-access memory
RPE	Retinal pigment epithelium
SNR	Signal-to-noise ratio
t_p	Time-to-peak
USB	Universal serial bus
VI	Virtual instrument

Units

B	Bytes
Hz	Hertz
Ω	Ohms
S/s	Samples per second
V	Volts

1. Introduction

Age-related macular degeneration (AMD) was the leading cause for unavoidable vision loss worldwide in 2010, according to the World Health Organisation [1]. There are currently an estimated of 150 million AMD patients worldwide [2] but this is projected to grow. One study estimated the number of AMD patients to grow to 196 million in 2020 and 288 million in 2040 [2]. While the exact costs for society worldwide do vary, another study in 2005 estimated the AMD to cause a total economic burden of $30 billion in the United States alone [3]. Therefore, the health and economic consequences of the disease are of immense significance.

AMD usually presents itself at an old age. It involves the degeneration of the macular region of the eye, which is rich with cone cells and forms the region of accurate vision [4]. As a result, AMD patients can eventually lose their accurate vision. The disease is further categorised into dry and wet form. The wet form is treatable, although it requires intravitreal injections, performed by an ophthalmologist every six weeks for a duration of two years [4]. Hence, it is apparent that this treatment option is neither convenient nor economical for the patient. The dry form, on the other hand, is incurable and less manageable, constituting approximately 90 % of all AMD cases [4,5]. In sum, a study on potential treatment options for both forms of AMD is of broad and current interest.

Numerous studies exist on pathogenesis of AMD along with studies on prevention by dietary choices and the use of eye protection [4]. In principle, many of these prevention mechanisms only lower the chances of the disease onset. Some studies have also identified potential treatment options for AMD. However, none has been effectively shown to stop or reverse the progress of the disease for any patient, although the wet form is more manageable. It has been suggested that thermotherapy applied to the retinal layer would have the potential to treat both forms of AMD, either curing or slowing down the disease progress [6]. This thesis is based on the hypothesis that AMD could be treated with heating the retina to therapeutic temperatures.

Heating the retina safely requires precision and knowledge of the temperature experienced by the retina. It has been shown that at different temperatures the retina exhibits varying bioelectric response recorded as electroretinogram (ERG) which is thus a potential temperature estimator. The response arises from the visual cells on the retina whose ERG response kinetics is proportional to the temperature experienced by the cells [7–9]. Using this information, it would be in theory, possible to heat the retina to a desired temperature and safely maintain it there when the parameters affecting the response

kinetics are known in function of temperature. The required therapeutic temperature should be reached and maintained at a stable level within a small margin of error. Reaching too high a retinal temperature could result in permanent retinal damage, but too low temperatures in heat treatment would not result in significant therapeutic effects. Further animal experiments and model building for the retinal temperature is required for constructing a reliable estimator of the retinal temperatures, and to investigate the therapeutic effect of thermotherapy.

The focus in this thesis is the development of a robust software for managing the ERG recording and retinal pigment epithelium (RPE) heating in animal experiments. Additionally, prospective applicability of the treatment set-up for clinical trials is investigated and reported. The goal for the future is to automate the entire process of ERG recordings, thermotherapy and safety aspects through feedback mechanisms, and introduce a medical device for treating AMD on human patients.

1.1. Goal, Objectives and Scope

The goal of this thesis is to produce a new software for actualising the experiments that aim at developing the heating treatment described before. Along the process, additional identified needs will be considered. Some of the most essential objectives of this goal include enforcement of pulse sequences, partial automation, real-time data analysis, and modularity. The accuracy and precision of this system are considered one of the most important features.

An older version of software exists for mouse experiments at the department of Neuroscience and Biomedical Engineering of Aalto University. However, a new one is needed because the previous version lacks expandability of features and does not meet even the near-future needs. This is because it was originally built for only registering ERG responses one at a time and to be analysed separately after concluding each experiment. Building a software from the ground up allows to better consider the future needs and account for expandability later. The software needs to collect and analyse data, as well as perform RPE heating on mice reliably and safely to reach therapeutic temperature levels.

The scope of the described goal and objectives is to produce a novel experiment program, which connects the ERG recording and analysis with the retinal heating device. When the retinal temperature estimation model and safety are mastered with the new software, it would be possible to take the set-up to clinical trials to investigate the applicability of the

RPE heating as a treatment option for humans with a potentially large health impact in treatment of AMD or other identified eye diseases benefitting from thermotherapy. Hence, the applicability of the current retinal temperature model for humans and safety aspects form a small part of this thesis.

1.2. Structure of the Thesis

In the following chapter, I will discuss the theoretical background, providing chief overview of the anatomy and physiology of the retina, examine ERG and its thermal dependency as well as cover the heating of the retina. In the third chapter, I will go through specific requirements for the software implementation and related hardware. In the fourth chapter, an overview and documentation of the software implementation is covered. Furthermore, I will gather points of improvement and considerations for the future of the controlled heating and its software. The final two chapters, five and six, I have dedicated for discussion and conclusions.

The appendix of the thesis includes a developer-oriented documentation of the recording software. The documentation covers overall description of the code behind the recording program, as well as more specific description of the utilised functions.

2. Theoretical Background

The theoretical background for this thesis can be divided into biological and technical components. Chapter 2.1 concerns the anatomical structure of the retina and differences between mouse and human eye. In the following segment (Chapter 2.2) I will discuss about the idea of RPE heating and why it is significant. Chapter 2.3 will be touching upon different methods for recording ERG and its thermal dependencies. The software will collect electrical signals, including ERG, through various sensors for processing and analysis. Chapter 2.4 introduces the topic of data acquisition, which is pivotal for collecting the electrical signals for analysis. Finally, in the prospective of developing a software, Chapter 2.5 introduces the theory on software engineering.

2.1. Anatomy of the Eye and Retina

Retina is the tissue covering the inside of an eyeball at the back of the eye, and its structure is similar with all vertebrates [10]. In a histological haematoxylin and eosin stain, the different layers of the retina are clearly visible (Figure 1). The dark nuclei bodies can be identified in three different layers which are connected by two synaptic layers. The nuclear layers are composed of the ganglion cell layer, the inner nuclear layer (INL) and the outer nuclear layer (ONL). The light detecting cells, photoreceptors, are located on the ONL, farthest away from the light source in comparison to the other layers. Medial to the photoreceptors is the RPE layer and Bruch's membrane which acts as a barrier separating the retina and choroid layer lying beyond the membrane [10,11].

There are two types of photoreceptors found in all vertebrate retinas: rod and cone cells. The former is a cell type specialised for dim light conditions, while the latter type is adapted for brighter conditions and able to distinguish various wavelengths of visible light. The rods saturate in bright light but recover their sensitivity over time as the eye is in dark conditions. Both photoreceptors convert visible light into electrical signals propagating towards the INL, where they connect to bipolar cells through a chemical synapse. Bipolar cells are part of the INL, which either hyperpolarise (OFF bipolar cells) or depolarise (ON bipolar cells) in response to light striking photoreceptors. Effectively bipolar cells transfer signal from photoreceptors to the ganglion cells. The ganglion cells carry the signal further as optical nerve to the brain. The photoreceptor cell signalling is modulated by the horizontal cells of the INL. Amacrine cells, which are also located in the INL, modulate the electrical activity of ganglion and bipolar cells. [10]

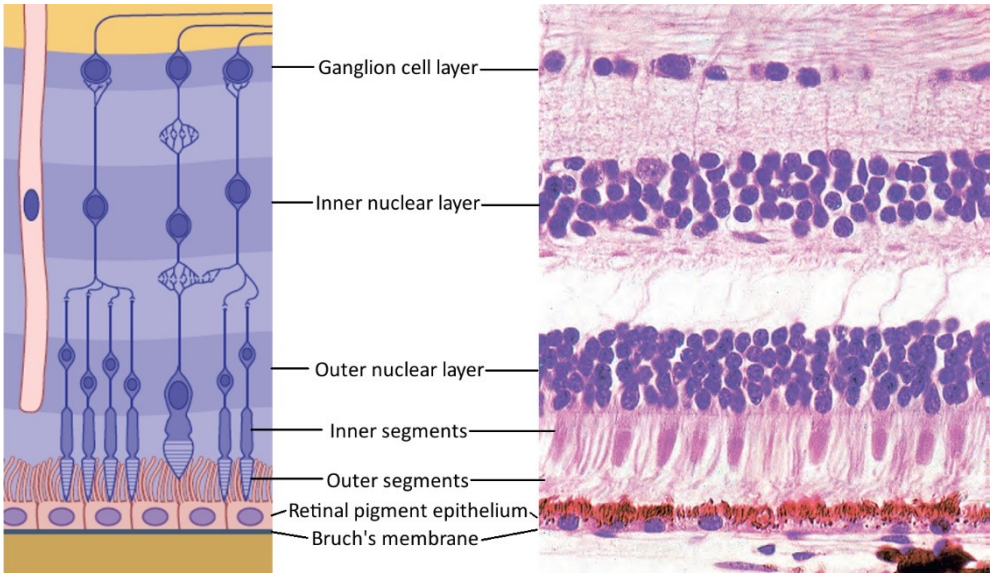

Figure 1. Schematic picture of the retina (left) and haematoxylin and eosin stain of the retina (right). Adapted from: [12].

The RPE is pigmented simple cuboidal epithelium layer responsible for nourishing the photoreceptor cells [11], as well as providing protection against photo-oxidative stress by absorbing scattered light [13]. As the RPE is situated right next to the choroid (separated by Bruch's membrane), it can experience large heat gradients when the two tissues are at different temperatures. This is due to the large number of blood vessels and their pronounced blood flow on the choroid layer, causing a large heat convection [12].

The eye fundus is the part of the retina observed directly across the lens of an eye. In fundoscopy or photographs of the human fundus, it is possible to identify blood vessels, optic disc, macula, and fovea. The macula is a slightly yellow-coloured region of the fundus containing the fovea which is high in concentration of cones, and therefore a region of accurate vision [12]. At the centre of the fovea is an avascular region, fovea centralis [14]. In AMD, mainly the photoreceptor cells of the macular region are degenerated gradually, resulting at first in loss of the accurate vision and blurring [4].

The pathogenesis of AMD is not yet fully known. Potential treatment options for humans are being investigated through animal models, which can be undertaken with any vertebrate. In our approach AMD is being simulated with mice models for better comprehension of the underlying features before moving to clinical trials. The shape of ERG responses in both mice and humans are alike due to the similarities in the structure of the retina, phototransduction mechanisms, and neuronal functionality. However, mice and humans have notably different anatomical properties of the eye (for an overview, see

Figure 2), and require different protocols for ERG recordings. The differences call for caution in planning when moving from test trials with mice to clinical trials with humans. For one, human patients are normally not needed to be anaesthetised during electroretinography. However, experiments with mice requires recording under general anaesthesia to minimise their movement. The disadvantage of general anaesthesia is its possible influence on the ERG signals [15]. Significant differences in ERG amplitudes between anaesthetised and awake adults have been shown in a small study [16], and for mice significant differences in amplitudes and time-to-peak values were successfully proven between various anaesthetic agents [17].

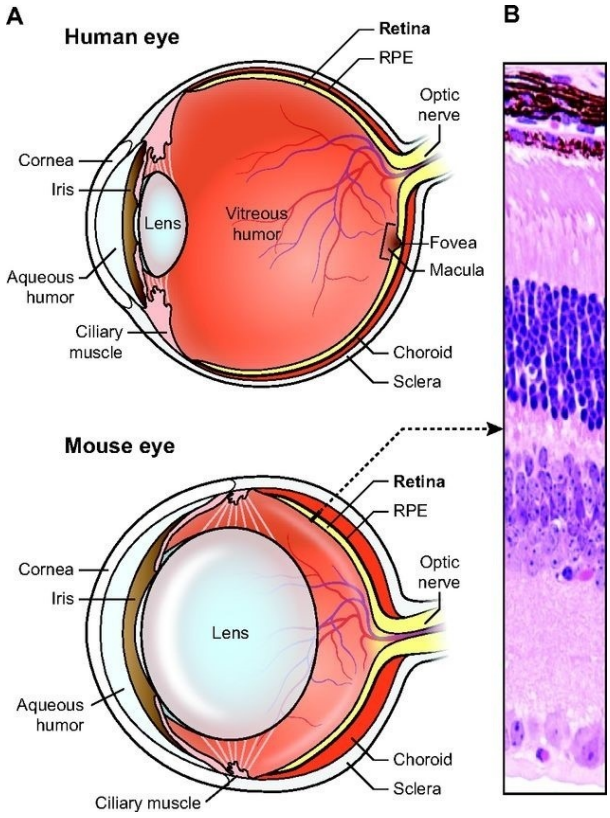

Figure 2. The structural differences between the human and mouse eye. A: schematic sagittal cross-section of the human and mouse eye (not to scale). As can be observed, the relative size of the lens of the mouse eye, in relation to the size of the eyeball, is much larger than that of human. B: mouse retinal haematoxylin and eosin stain. The histological structure is similar to that of a human illustrated in Figure 1. Adapted from: [18].

The anatomical differences between human and mouse eyes are mainly confined to macroscopic structures. They include different sized eyeball and its proportion to the lens, as well as the different vitreous and aqueous volumes. For mice, the lens, combined with

cornea, can fill up as much as 60 % of the total optical length of the eye (see Figure 2). For humans the lens is much smaller in comparison to the size of the eyeball. On cellular level, mice have a bigger rod-to-cone ratio than humans, making study of rods more ideal in mice [19,20]. Unlike humans, mice do not have a macula. Therefore, AMD and its symptoms can only be mimicked with mice; for instance, by using genetically modified mice [21]. Some quantifiable features between murine and human eye considered relevant for this thesis are listed in Table 1.

Table 1. Comparison of differences between the mouse and human eye properties

	Mouse	**Human**
Axial length of the eye	3.37 mm [19]	24 mm [22]
Volume of aqueous humor	4.4 µl [19]	260 µl [22]
Volume of vitreous humor	5.3 µl [19]	5.2 ml [22]
Lens axial length	1.9 mm [23]	4 mm [24]
Retinal area	15.6 mm^2 [19]	1204 ± 184 mm^2 [20]
Cone-to-rod ratio	0.028 [19]	17.7 [20]

2.2. Retinal Thermotherapy

The scope of RPE heating in this thesis is to reach high enough temperatures in the RPE to produce so called heat shock proteins (HSPs). HSPs are molecular chaperones for correcting protein folding [25], hence preventing the formation of detrimental protein aggregates. For this reason, it has been suggested that HSPs would be an important factor in AMD and possibly in its prevention [6,26]. Heating can also be used for photocoagulation of the retinal cells, but this is not considered relevant in this thesis.

One solution for heating of the RPE layer is the use of an infrared (IR) light source, such as a laser or a light emitting diode (LED). Carefully selected long enough wavelength IR does not cause a photoresponse, as the photopigments of neither cones nor rods absorb significantly wavelengths past the visible light range [10]. Therefore, IR light source does not interfere with the responses elicited by light stimuli in electroretinography. It is good to note, though, that the transmittance of light through cornea, lens, as well as aqueous and vitreous humor, peaks in the range of 650 to 850 nm [27]. Transmittance quickly drops after 1000 to 1200 nm, suggesting to avoid the use higher wavelength IR than this for RPE heating. Additionally, transmittance in all the tissues diminish slightly in aging [27].

2.3. Electroretinogram

Electroretinogram (ERG) is an electrophysiological recording technique that registers electrical potentials arising from the retina. The electrical response is quantifiable by placing an indicator electrode in electrical contact with the cornea or at different levels inside the retina [28,29]. The indicator electrode is compared against a reference electrode placed anywhere but the same eye. Figure 3 illustrates the electrical current flows through the retina, vitreous, cornea and sclera of the eye. In it, A represents the local current pathway confined entirely to the retina. B, on the contrary, represents the remote current pathway which traverses from retina and through the vitreous and cornea. It completes the circuit by traversing through sclera back to retina. [28] The pathway recorded in non-invasive ERG is the pathway B, which is a sum potential. One approach for this is to obtain a corneal contact with contact lens [29]. This is also the technique used in the recording set-up in this thesis.

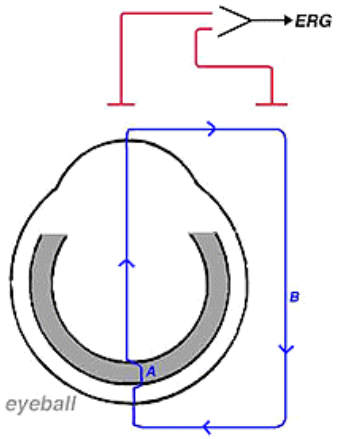

Figure 3. A schematic drawing of the extracellular currents in ERG. The two current pathways are A and B, which can be described as local and remote, respectively. Non-invasive ERG techniques involve the recording of the remote (B) current flows. [28]

Upon stimulating the eye with a light source, a characteristic ERG response can be observed as a set of electrical potential changes produced by changes in the currents reflecting the retinal cell activity. Different conditions can dictate which cells compose the final ERG signal. In dark-adapted conditions and in low background illumination, a very dim flash produces an ERG signal that is generated mainly by the rod signal pathway [10]. On the other hand, a very strong flash or flash produced under background illumination produces signals more generated by the cone pathway [10].

Depending on the stimulus light intensity, length and modulation, four characteristic waveforms can be observed: a-, b-, c- and d-waves which arise from the superposition of the signals generated in different cell types of the retina. The negative a-wave is derived from rods and cones, while the positive b-wave following the a-wave is derived mostly from bipolar cells combined with a negative Müller cell (glial) component due to the Müller cell interaction with the photoreceptor cells. The RPE cells mediate the c-wave. The d-wave is bipolar cell mediated off-signalling when the light source is turned off in a long stimulus. [28,29] If the light stimulus is dim and brief, a bit different response is elicited. For a dark-adapted eye a dim flash produces only a b-wave component [28]. As the stimulus strength is increased, the amplitude of the b-wave increases, the time-to-peak (t_p) value of it shortens, and an a-wave becomes visible and increased in amplitude [28]. The time-to-peak, or implicit time, is the amount of time taken from the given stimulus until an a-wave or b-wave trough. An example of a two second on and off stimulus is illustrated in Figure 4, where all four types of waves are readily observed.

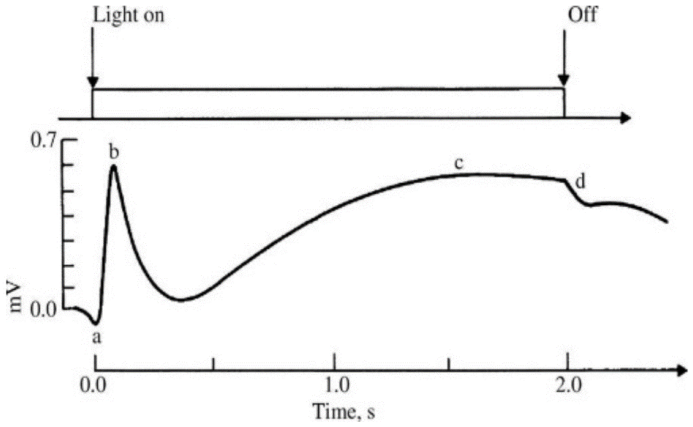

Figure 4. Vertebrate ERG response for 2 second light stimulus. There are four identifiable wave components: a, b, c and d. For a shorter flash stimulus, the response would consist mostly of b-wave and possibly negligible a-wave. [29]

The rod and cone responses are separable by careful selection of the light stimulus. When providing a dark-adapted retina with a dim flash, mostly rod-mediated mechanisms are active and conveyed through ERG. On the other hand, providing high intensity enough constant background lighting will saturate the rods, secluding the rod responses from the recorded ERG. [15]

Typically, a flash response is generated with an on and off type of light stimulus. Although a flash stimulus yields more isolated response, there are also other ways to stimulate the eye to generate different types of ERG responses. A relatively common alternative to flash stimulus is flicker ERG (fERG) where the light stimulus is produced

with a certain flicker rate. The stimulus waveform can be of any shape, for instance square wave (periodical on and off), sine wave or sawtooth wave. [15,28] Rod-mediated responses are present at lower frequency ranges of flicker, typically below 15 Hz, while cone-derived response can be isolated with a flicker greater than 30 Hz. [28]

Each photoreceptor type has a unique sensitivity to various wavelengths that differs by cones and rods as seen in Figure 5 for human photoreceptors [30]. The various photoreceptor types have their spectral sensitivity curves at different regions of the visible spectrum. Each photoreceptor cell type has its absorption peak at different wavelength. Cones consist of blue-sensitive S-cones, green-sensitive M-cones and red-sensitive L-cones with absorption peaks at 420 nm, 534 nm and 564 nm, respectively. Rods have their absorption peak at 498 nm.

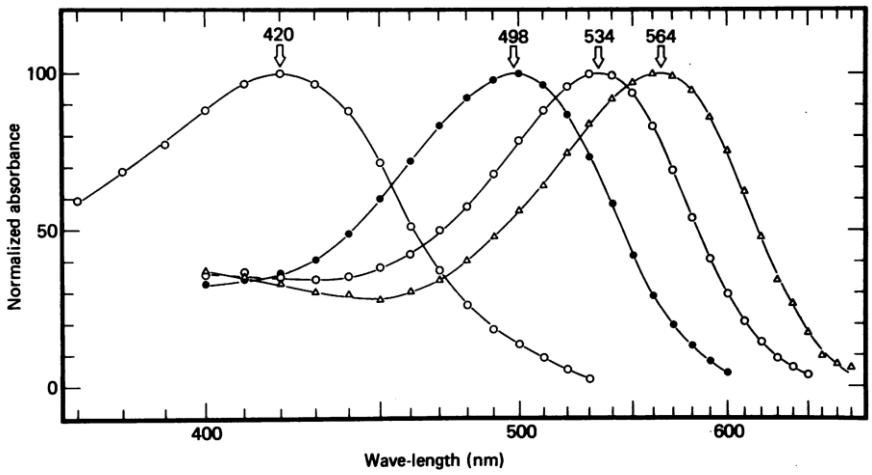

Figure 5. Spectral sensitivity of the human photoreceptor cells. The plots and their absorption peaks from left to right are: blue-sensitive S-cones (420 nm), rods (498 nm), green-sensitive M-cones (534 nm) and red-sensitive L-cones (564 nm). [30]

Often it could be useful to be able to isolate responses generated only by a single photoreceptor cell type. In the case of rods, this is easily attained by stimulating the eye with only dim light in dark-adapted conditions [15]. For cones, it becomes more difficult as the spectra of three different types of cones overlap. On the one hand, a short wavelength stimulus aimed at extracting blue-sensitive S-cone response would minimise the effect of other photoreceptor cell types. On the other hand, the number of S-cones is 2-4 times lower than the other cone types and are not found in fovea, making it more difficult generate quantifiable responses [10].

2.3.1. Characterising Electroretinogram Responses

Evaluating and characterising ERG signals quantitatively in an automated manner can yield more information than just qualitative inspection of the signals. Different methods are applicable for different types of stimuli and can be based on the temporal information, recorded signal magnitudes, shapes, and spectral properties. Our aim is to extract quantifiable information from the ERG signal, which can be further used in characterising the desired phenomena, or, for example, the retinal temperature, as in this thesis.

The most indubitable feature in an ERG response to a flash stimulus is the amplitude information [28]. The amplitude value of a-wave or b-wave impart information on the strength of the photoresponse. When the stimulus source is intensified, also the amplitudes are expected to grow. The problem with the use of absolute amplitude values is that they are often not constant across a span of recordings, as they can vary among individuals and throughout the recording set-up. Normalising or using relative amplitudes would be an option to tackle this and have more comparable data. The amplitude information of other wave types (c- and d-waves) would require using a longer-lasting light stimulus, which is not desired when collecting rod-mediated ERG or aiming at higher temporal accuracy.

Another useful feature is the time-to-peak (t_p) value of the signal troughs, i.e. the amount of time it takes to reach the peak of either the a-wave or the b-wave. Finding a t_p value for the signal can easily be attained accurately by implementing an algorithm that iterates a polynomial fit around the maximal value of the wave, as has been shown in mice experiments [9]. Onset latency is the measure of how long it takes from the stimulus onset until the signal begins to increase (b-wave) or decrease (a-wave) toward the trough [15]. A faster onset latency corresponds to shorter t_p value as well.

Oscillatory potentials (OPs) are small oscillatory waves that can be observed partially overlapping with a- and b-wave, and are thought to be originated in amacrine cells [31]. For analysis, the amplitude of these oscillations and their frequency can be quantified.

2.3.2. Electroretinogram's Thermal Dependency

In regard to this thesis, the ERG is a tool that has been used in our research team to estimate temperature of the retina in function of time. This is based on features that can be quantified from the ERG signals stemming from a response to a stimulus. Some of these were introduced in the previous chapter.

When the temperature of the retina increases the observed kinetic features change. The photoresponse kinetics arise from molecular mechanisms of the photoreceptors and of the other cell layers of the retina, which generally activate and deactivate faster at higher temperatures, as is the case in recorded ERG signals under light stimulus of invariable wavelength and intensity. [7–9] Using this information, it is possible to reverse-engineer the temperature experienced on the retina by utilisation of the ERG features.

The retinal a-wave and b-wave amplitudes and their t_p values have been shown to be dependent on the body temperature, and thus affected by retinal temperature, in rodents. As the temperatures are decreased below normal body temperature, the amplitudes of both waves decrease, while the t_p interval becomes longer [32,33]. On the other hand, induced regional hyperthermia on the retina generally decreases the a- and b-wave amplitudes, but shortens the t_p interval [9]. In the range of 30 to 37 degrees, the change in amplitudes follows approximately linearly changes in temperature [32]. Various ERG features and their dependencies to higher retinal temperatures are gathered in Table 2.

Table 2. Various ERG features, their contributor cells and observed effects in response to increased retinal temperature (regional hyperthermia)

Feature	**Contributing cells** [31]	**Effect of higher retinal temperature**
a-wave amplitude	Photoreceptors and bipolar cells	In mice: increased amplitude* [32,33]; sometimes decreased [9]
b-wave amplitude	Photoreceptors and bipolar cells	In mice: increased amplitude* [32,33]; sometimes decreased [9,33]
a-wave t_p	Photoreceptors and bipolar cells	Shorter a-wave t_p in mice [9,32,33]
b-wave t_p	Photoreceptors and bipolar cells	Shorter b-wave t_p in mice [9,32,33]
Onset latency	Photoreceptors and bipolar cells	Shorter onset latency in mice [9]
Oscillatory potentials	Amacrine cells	Increased amplitudes in rabbits* [34]; decreased amplitudes and lower frequency in carps* [35]
Spectral sensitivity	Photoreceptors	Higher b-wave amplitude for 780 nm stimulus than for 532 nm in mice [9]
b-wave integration time	Photoreceptors and bipolar cells	Integration time decreases in mice. Inferred from [9]

*The effect is inferred from experiments where the retina or entire subject was brought to temperatures *below* their regular body temperature.

2.4. Data Acquisition

In order to record signals arising from ERG potentials and save these in digital format, various components are required. Firstly, a sensor placed in contact with the eye is needed to record the electrical potentials. As the amplitudes arising from ERG are very small, a signal amplifier is additionally placed and the amplified signal passed to analogue-to-digital converter (ADC) for digitalisation of the analogue signal [36]. The converted digital signal is passed to the personal computer (PC) either directly or through a data buffer. This entire set-up, which acts as an interface between the analogue signal and the PC, is hereafter referred as the data acquisition system (DAQ). Refer to Figure 6 for a visual overview of a DAQ.

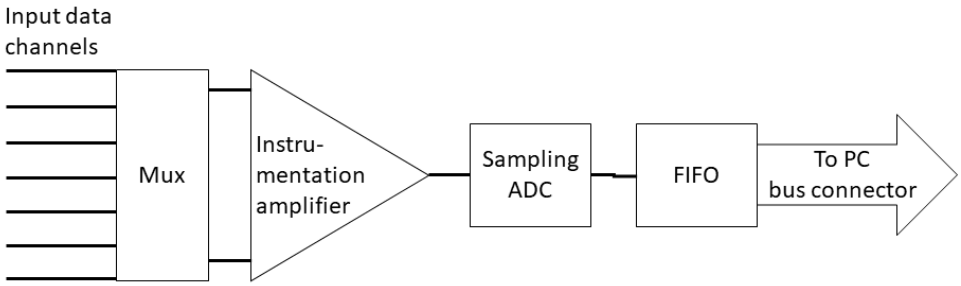

Figure 6. A typical DAQ device consists of a multiplexer (mux), instrumentation amplifier, analogue-to-digital converter (ADC) and first-in, first-out (FIFO) buffer. All of this connects to a PC through a bus, such as universal serial bus (USB) or peripheral component interface (PCI).

Many commercial DAQ devices are built for handling both analogue and digital input/output (I/O). As the DAQ hardware needs to interface between the sensors and a computer, they are either connected directly to the computer's motherboard through peripheral component interface (PCI) or through other serial ports or buses such as universal serial bus (USB). One provider of DAQ hardware solutions is National Instruments™ (NI) which also provides a software design platform LabVIEW™ optimised for use with their hardware. These were also utilised in this thesis.

The NI DAQ devices selected for the use in this thesis contain a multiplexer (mux), instrumentation amplifier, sampling ADC and first-in, first-out (FIFO) buffer, similar as presented in Figure 6. Since multiple analogue and digital input wires are connected to the DAQ device, the mux is used for selecting one or more of several wires to be passed onward [36,37]. The instrumentation amplifier takes these input lines and, by using a set value of gain, amplifies the signal. In NI DAQ devices the amplification with

programmable gain minimises the so called settling time, yielding the highest possible temporal resolution for sampled signals [37]. The amplified signal is then sampled by the ADC into a format readable by the PC. The sampled signal is passed to the FIFO buffer which wires the digital values to a PC through a connector [36]. The FIFO buffer is an element that stores in its memory the values from ADC and outputs them to the PC one by one in the order in which they were passed to the buffer. This prevents data loss in case of a low read rate of a PC [36].

DAQ hardware normally contains the ADC [38]. The accuracy of recorded electrical potentials is limited not only by the recording electrodes, but the DAQ hardware and its ADC. The ADC resolution is characterised by its bit number and sampling rate. A good descriptive value for the bit number is the value of least significant bit (LSB). It dictates whether a number is even or odd. It also describes the accuracy of an ADC, as any input signal within set range can have 2^n discrete digital values [36]. For instance, a binary number consisting of 12 bits can contain precision of 2^{12} = 4 096 in decimal system or 4 096 discrete levels to describe a value. Therefore, an input value converted by ADC can receive one of 4 096 possible digital values [37]. If the ADC was working with a range of -10 V to +10 V (i.e. scale of 20 V in total), the resolution of 1 LSB is 20 V ÷ 4 096 [37], hence ±4.88 mV. For a converted signal, this means that any value within the sampled signal in digital form can fluctuate in steps of 4.88 mV. Adjusting the input voltage range also affects the precision by changing the resolution of 1 LSB. Therefore, when evaluating purchase of DAQ hardware, the bit precision of the device is an important factor to take into consideration.

Another important note when collecting data is to consider the frequency range of the expected signal. According to the Nyquist-Shannon sampling theorem, the ADC sampling should take place at more than twice the rate of the highest frequency component of the recorded signal to avoid aliasing [38]. Therefore, the sampling frequency needs to be set high enough and the DAQ hardware needs to be able to perform at this rate. Especially sampling multiple inputs with high frequency can be a bottle neck for an ADC.

2.5. Software Engineering

In this thesis, the focus is in the development and implementation of a new software capable of processing ERG signals and interacting with the I/O devices of the retinal

heating setup. This chapter will cover theoretical background in software engineering and software process models.

2.5.1. Development Process of Software

The development process of a new software, also called software development life cycle model, includes three different models: the waterfall model, incremental development, and integration and configuration. Each of these describe a different approach that can be taken in the development of a software.

The waterfall model breaks down the process into distinct phases which are implemented one by one. These phases can include, for instance, planning, implementation, and validation. For businesses, this model is one of the most cost-effective solutions. [39] Although this seems slow at first, the waterfall model becomes time-effective by taking into account potentially arising issues which would present themselves easier than directly tackling the software implementation.

The incremental development relies on concurrently occurring activities and gathering feedback on the system, while building new versions on top of this until final build is reached [39]. Hence, this model is more pivotal for end-user feedback. The benefits of development processes, where end-user feedback is emphasised throughout the process, include mapping features that would have otherwise been omitted from the end-product. This type of process model does not require meticulous prior planning but can engage a developer into the work right away.

Finally, the integration and configuration process model is centred in reuse of components, for instance from another software, and integrating them in the new software with required changes [39]. This is especially beneficial when developing a new version of a software or parts of it, as the developer can avoid using unnecessary effort and time in development of pre-existing features from scratch.

Out of the different process models, I have identified the most useful to be the incremental development for the basis of this thesis. This allows much more freedom in consulting the end-users of the software and identifying better possible additions the software still lacks. At the current stage there already exists one older version of software like this in the department of Neuroscience and Biomedical Engineering of Aalto University. Therefore, some parts of this old software might be utilised partly in the new one. For this reason, the process of integration and configuration is used to some extent as well.

2.5.2. Software Activities

The process of an entire software engineering project can be broken down in separate activities (or phases). In his book on software engineering, Hirsch [39] has listed four distinct activities: software specification, software design and implementation, software validation, as well as software evolution. In a waterfall model, this order of activities would be kept, and proceeded to the next stage once the previous activity has been concluded. However, since the approach in this thesis is incremental development, the order of these activities is not set and can take place in any order, returning to each activity multiple times.

The very first stage of any project starts with planning. In the case of software phases, this would be the software specification part. It involves identifying various requirements for the software, eliciting existing solutions, and a check on the feasibility of the desired new features [39]. In companies, this first stage activity alone would resolve whether or not to construct the software at all, depending on the expected costs, feasibility and planned use. The software developed in this thesis also included the planning phase as the very first activity. Nearly all the desired features were identified, and a non-functional prototype of the software was constructed to receive feedback from the team on the intended direction of the development. This activity also involved investigating potential DAQ devices to assess possible need for device investments, and the effects the chosen data acquisition system would bear on the software architecture.

After concluding the software specification phase, the software design and implementation should follow. This activity upholds, only in parts relevant to this project, the architectural planning of the software, interface design and selection and design of components for the software. [39] In regard to the thesis, this activity involved evaluating what kinds of structures would be used in the software, and what types of interfaces would be required with involved devices or other system components. Additionally, it was evaluated which components of previous software versions could be reused for the new software.

The validation phase could be undertaken as chronologically the third activity in a development process like the waterfall model. Since many new features were planned to be added to the final product, validating seems more rational at the same time with the development. Validating the software involves three parts that are also very pertinent in this thesis: component testing, system testing, and customer testing [39]. The component testing involves testing separately each component, such as functions (or subdiagrams)

and objects, making up the final product. This was in majority achieved by testing each function after their creation, and finally through unit testing as described later in Chapter 3.3.7. The system testing part involves testing the interactions between components. This was also in most parts validated during the software implementation by testing functions interacting with other functions. The system testing is later in the thesis referred to as bench testing. The customer testing is the last part of validation and was so also in this thesis. This involves the system being tested by a customer, or in this case by the research team members using the software. The purpose is to show any potential errors that the developer might not realise. How well the software manages to fulfil end-user's needs and expectations is also part of this. The customer testing, or in this case end-user testing, is covered in Chapter 4.

The final activity is the software evolution. As any software can experience changes at any time during and after the software implementation [39], maintainability (evolution) of the system needs to be considered. Maintainability is allowing developers later to expand the capability of the software. One of the reasons why this thesis builds a completely new software is because the previous one is not fully suited for allowing expandable features without sacrificing its functionality or by a developer using tremendous amount of time. The maintainability is discussed in depth in Chapter 3.3.1.

3. Materials and Methods

This thesis encompasses the development and implementation of new features for the existing RPE heating device. The end goal was to achieve a software that can control the heating safely, record ERG-data reliably and perform continuous analysis on the acquired data in real time. This chapter introduces the devices and methods exerted for this goal. This covers the overview of the programming platform and hardware (Chapter 3.1), the experiment set-up (Chapter 3.2), and identified features and methods relevant for software implementation (Chapter 3.3).

3.1. Programming Platform and Hardware

The software was programmed with National Instruments LabVIEW™ (Version 16.0, Software Platform Bundle Fall 2016) [40]. LabVIEW is based on graphical programming syntax. The programs in LabVIEW are known as virtual instruments (VIs), each one containing a front panel, a block diagram and a connector pane. The front panel of a VI is the user interface, while the block diagram contains all the source code in graphical syntax. The connector pane defines the VI's inputs and outputs, so that it may be called by another VI. In LabVIEW, a VI in another VI is known as subVI, which is analogous to subroutines in text-based programming languages [41].

LabVIEW enables an interface for using MathWorks MATLAB® functions, which were in parts also utilised in this project (MATLAB version R2017a) [42]. The retinal heating device and ERG-recording set-up comprise of various devices, which the LabVIEW driven software interacts with. These devices need to be connected to the PC running the software. These devices are discussed in Chapter 3.1.1.

The software was implemented on a PC running Microsoft's Windows 7. The PC contains 8 GB of DDR3 random-access memory (RAM), an Intel® Xeon® CPU X3450 processor with a quad-core and eight threads clocking at 2.67 GHz, as well as a regular hard disk drive. The number of cores and threads in a processor dictates the number of tasks that the PC can run concurrently. For instance, simultaneous execution of two parallel for-loops in LabVIEW assigns one thread for each, either within the same core or different cores of a multicore processor [43]. The operating system, RAM, processor and storage type are the most relevant features for the undertaking of this software project. Although LabVIEW does also support computing with the graphics processing unit (GPU) to accelerate arithmetic computation of large data sets [44], it was not utilised in this thesis.

It was considered that adding GPU computing would complicate the source code, while the gain in computational efficiency would remain insignificant.

3.1.1. Device Set-up

The device set-up in this thesis is composed of the DAQ hardware, I/O equipment (transducers) and the PC with its software. In this thesis, the device set-up includes two DAQ devices: NI PCI-6221 [45] and NI PCI-6024E [46], both purchased from National Instruments™. Each DAQ device is directly connected to the PC's motherboard as peripheral component interface (PCI) and both form the interface needed for all the transducers. The input sensors include: a thermistor for body temperature recordings, a photodiode for monitoring the optic power of the heating device, two ERG sensors and a piezo sensor for movement tracking. Additionally, NI PCI-6024E controls the ERG amplifier reset by setting baseline of the instrumentation amplifier to zero. In addition to DAQ-connected transducers, one USB 3.0 port is used for video recording of the fundus of the patient. An overview of how these devices interact with each other, as well as all the connected I/O equipment can be observed in Figure 7.

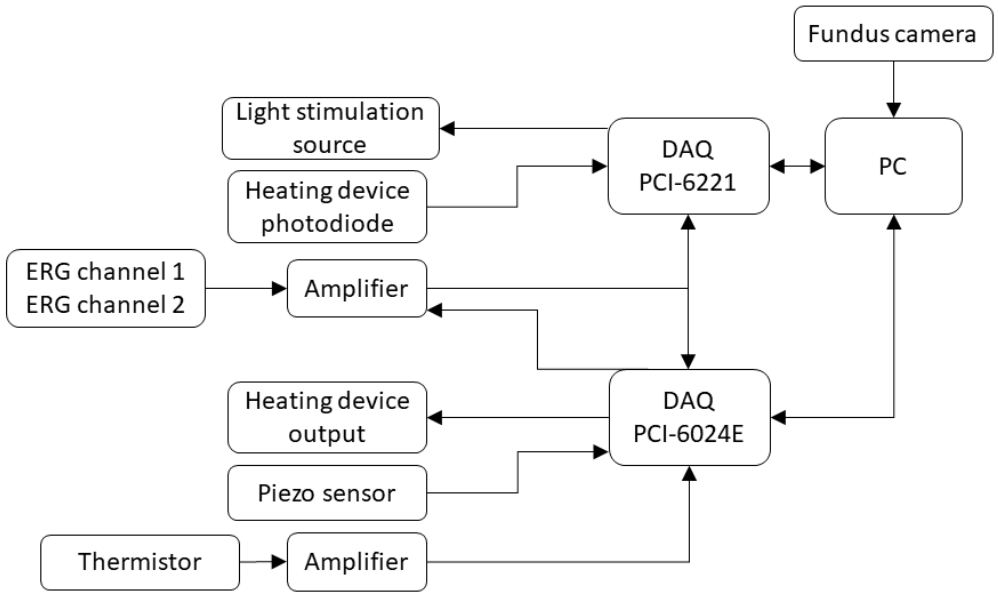

Figure 7. A diagram displaying the simplified layout of the different I/O devices and their interaction with the DAQ devices and the PC. The direction of the arrows describes the direction of signal flow: arrows towards DAQ devices or PC are input signals and vice versa.

The two acquisition devices contain internal timing engines and a 68-pin layout for analogue and digital I/O connections. The timing engine in each DAQ device is necessary in acquiring input signals or generating output, since the engine controls when to start and stop signal acquisition or generation [47]. The resources of one timing engine can be utilised by one task at a time. A task is an event of signal acquisition or generation, which takes place at a set sampling rate and defined time interval by the timing engine. As most NI devices have only one common timing engine for all analogue input channels, it means that all signals acquired simultaneously need to share the same sampling rate and start and stop times. Hence, recording multiple distinct analogue signals under the same DAQ device can become cumbersome if the user needs to acquire the signals at different frequencies or time intervals. Both selected DAQ devices were available in the thesis environment of which the NI PCI-6221 was used in the most recent device set-up. Two timing engines were identified as necessary in this thesis. Thus, the NI PCI-6024E was brought in to facilitate an additional timing engine to the device set-up.

The NI PCI-6221 is connected from the output-end to a shielded 68-pin connector block [48]. The shielding improves signal-to-noise ratio (SNR). The NI PCI-6024E, on the other hand, is connected from the output-end to an unshielded 68-pin connector block [49]. The product details of the two DAQ devices have been summarised in Table 3.

Table 3. Summary of the DAQ hardware properties for the NI PCI-6024E [46] and the PCI-6221 [45]

Property	**NI PCI-6024E** [46]	**NI PCI-6221** [45]
Maximum sampling rate	200 kS/s	250 kS/s
I/O resolution	12 bits	16 bits
I/O resolution at input range of -10 … +10 V	4.88 mV	0.305 mV
Settling time	5 µs for ±1 LSB of step	7 µs for ±1 LSB of step
Crosstalk at 100 kHz	-60 dB	Adjacent channels: -75 dB, Non-adjacent channels: -90 dB
Number of analogue outputs	2	2
Output rate	10 kS/s	833 kS/s
Number of digital I/O channels	8	24
Number of counters/timers	2, 24-bit	2, 24-bit
Internal base clocks	20 MHz, 100 kHz	80 MHz, 20 MHz, 0.1 MHz

As can be noted, the PCI-6024E is significantly inferior to the PCI-6221 when it comes to resolution and sampling rate. Hence, in this project, the PCI-6221 is configured for quantifying the ERG responses and heating device photodiode voltage as well as for controlling the light stimuli, because temporal and I/O resolution are important for all of these. As some actions need to run in parallel with the high-resolution acquisition, the other DAQ device (NI PCI-6024E) is utilised in continuous background acquisition of the ERG signal and thermistor voltage as well as in monitoring the breathing of the mouse via a piezoelectric sensor. Additionally, the heating device power output is controlled with the PCI-6024E. The I/O equipment presented in Figure 7 together with their interactions with the DAQ devices are introduced in more detail in Table 4 and Table 5.

Table 4. Interaction between digital and analogue I/O of the PCI-6221 [45]

NI PCI-6221 [45]	**Type of channel**	**Purpose**
ERG channel 1	Differential analogue voltage input	Main ERG signal for quantifying rising electrical potentials in the retina
ERG channel 2	Differential analogue voltage input	Secondary ERG signal for quantifying electrically rising potentials in the retina
Heating device photodiode	Differential analogue voltage input	Evaluation of the actual power output of the retinal heating device
Light stimulation source	Counter voltage output (digital)	Generation of a pulse or PWM signal

Table 5. Interaction between digital and analogue I/O of the PCI-6024E [46]

NI PCI-6024E [46]	**Type of channel**	**Purpose**
Piezo electrode	Differential analogue voltage input	Monitoring breathing rate
Continuous background ERG signal	Differential analogue voltage input	Acquiring data of the ERG channel for the duration of the entire experiment
Heating device output	Analogue voltage output	Control of the retinal heating device light source power by adjusting its voltage output
ERG amplifier reset	Counter voltage output (digital)	Resets the ERG signal baseline to zero
Thermistor	Differential analogue voltage input	Quantification of the rodent body temperature

The analogue input acquisition is vulnerable crosstalk. Crosstalk is an event where one measured input signal affects the value and shape of another input signal measured from another channel of the same DAQ device. This is influenced by the settling time of the instrumentation amplifier of the DAQ hardware [37]. Settling time is described as the amount of time it takes for the amplifier output to reach and stay within the limits of a defined accuracy [50]. When the sampling interval becomes short in comparison to the settling time, the accuracy of the recording decreases and crosstalk might take place [37]. Since this project utilises multiple different analogue input recordings, it is important to keep the settling time as short as possible. This is the case especially because the originating ERG signal is in the range of tens to hundreds of microvolts, while the smallest amplified ERG signals recorded by DAQ can be in the order of tens to hundreds of millivolts, and therefore susceptible noise.

National Instruments has listed four ways of ensuring low settling times in their DAQ M Series user manual (includes PCI-6221) [37]. Firstly, and most importantly, the signal source impedance should hold a maximum impedance of 1 kΩ. Using as short and high-quality cables as possible for the sensors is the second most crucial factor. This decreases noise as well as minimises crosstalk and transmission line effects. Thirdly, when scanning multiple channels, the channel scanning order should be inspected to avoid transitions from high voltage input channel to lower voltage input channel, because the settling time increases in these cases. Finally, and least importantly, the input signals should not be scanned at rates faster than necessary. [37] These were accounted for in this thesis by rearranging the channel scanning order to occur from lowest voltage input to the highest. New sensors added to the recording set-up were also connected with coaxial cables that were selected as short as possible. Finally, as explained before, those sensors not requiring high scanning rates were split into a different DAQ device (PCI-6024E) than the sensors sampled usually at higher rates (PCI-6021).

3.2. Experiment Set-up

In this thesis, the implemented software (related features described further in Chapter 3.3) was tested under two circumstances. First, the software was validated by a bench test, which included unit testing of smaller pieces of the software. Second, an *in vivo* experiment with a live mouse was conducted to test the software's performance under its intended experiment set-up, which is explained below.

For the *in vivo* experiment a mouse is anaesthetised with isoflurane gas and laid down on a water-circulated heating pad equipped with a nose cone for continuous isoflurane anaesthetic delivery. A piezoelectric sensor is then placed in skin contact just below the diaphragm of the mouse, relaying information on the mouse movement, and hence providing a tool for estimating its respiratory rate. The core body temperature is monitored by a thermistor placed in the rectum of the mouse. The light stimuli are defined from the software, which controls a green laser apparatus emitting light through the mouse pupil into the retina. The generated ERG response signal is recorded with a corneal silver-silver chloride electrode and relayed through the DAQ device to the computer, where the software performs retinal temperature estimation on the recorded response. The estimation is performed with MATLAB functions utilising the extracted kinetic features of the responses (see Chapter 2.3.2). The retinal temperature is estimated continuously while the user initiates retinal heating with an IR laser whose output power is controlled from the software. The user can adjust the heating power output until desired target temperature is reached. The ERG response based retinal temperature estimation is run continuously to evaluate the retinal temperature and to adjust the heating power to reach the target. Once the desired temperature has been reached and kept constant for a desired amount of time, the experiment is ended, and the mouse is revived from anaesthesia or sacrificed.

3.3. Key Requirements for the New Software

The planning stage of the software implementation included identifying key changes and requirements desired for the new software. These were inspired by the availability of the same features (although more primitive) or lack thereof in the previous software. I identified in total eight different key features that were considered in this thesis. These are presented in Table 6, and their implementation in the software is discussed in the upcoming Chapters 3.3.1–3.3.8.

Table 6. Key identified features and requirements for the software to be engineered

Feature type	Chapter
Maintainability	3.3.1
Dynamicity	3.3.2
Accuracy and precision	3.3.3
Pulse-width modulation and pulse sequences	3.3.4
Automation	3.3.5
Safety, error handling and user-friendliness	3.3.6
Verification and validation	3.3.7
Documentation	3.3.8

3.3.1. Maintainability

The maintainability aspect was catered to by designing the software as easy as possible to adapt to changing conditions. Such conditions could be newly identified needs or features for the software, or other factors affecting the software architecture, such as new hardware. For this reason, it was important to develop each aspect of the program and its functions so that changing individual parts in the software is made as easy as possible. Documentation plays a large role in maintainability as well, by allowing the developer to gain better understanding of the underlying features of the software. Thus, also a documentation was considered as a complimenting part of maintainability (see Chapter 3.3.8 and Appendix A for developer documentation).

MATLAB functions, which are part of factoring in the maintainability, were added as an interface in the new program. As most of the post-processing and data analysis in the project is performed with MATLAB, it suits the needs of the software to also perform these activities in real time with MATLAB functions. In LabVIEW itself, many functionalities were designed as subVIs that can be easily reused or revised in different segments of the software. Devising these subVIs in a way that allows them to be implemented frequently is the best use of the hardware resources of the computer by minimising the computational effort and memory needed.

The developed software contains many subVIs. The main program instance of the software I will refer to as the Main VI. Since the Main VI is a large and complex part of the software that interacts with various subVIs, its maintainability is one of the most difficult ones. For testing purposes, two kinds of additional features were created for the

Main VI in aim to increase its maintainability: simulated testing and unplugged I/O testing. These appear as options for the user when starting the software. The simulated testing offers an option to run a past experiment again, reusing the data acquired in that experiment. This is especially helpful when developing features that depend on receiving actual experimental data, since the stimulated testing allows the developer to run tests on the code without having a live mouse subject. The unplugged I/O testing on the other hand is meant for testing the software without any I/O input or output. The benefit of such testing proves to be useful when the DAQ devices are not connected. When the DAQ devices are not connected, attempting to acquire signal input or generate output produces an error in LabVIEW. The error is circumvented with unplugged I/O testing by not allowing acquisition of inputs or generation of outputs. Instead, programmatically random generated data is acquired as input data.

While implementing the new software, it was noticed that many variables had no interdependency across different VIs in the previous software used. Lack of interdependency can cause problems when changes are made to the variables. An example of such variable is the cluster control that contains stimulus input parameters in Main VI (Figure 8). It is used additionally in four subVIs. If a developer, for instance, added one more stimulus type to the drop-down list, they would need to address this change in all subVIs separately. Thankfully, LabVIEW has a feature called type definition which was utilised in the software implementation. In LabVIEW, a type definition is a variable which is saved to a separate file. It is possible to reuse a type definition in any part of the software. Doing so ensures that all instances of the variable are referred to the same file; thus, preserving its properties across all the subVI instances. In the example of a cluster variable, the elements within a cluster automatically update in all instances of the software when the cluster is edited in any VI containing the variable.

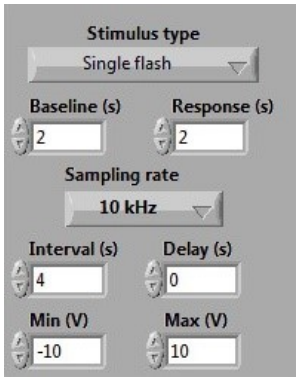

Figure 8. Cluster control of the implemented software containing the stimulus parameters. This control was implemented as a type definition and is used in four subVIs in addition to Main VI.

3.3.2. Dynamicity

The second identified need, dynamicity, has to do with the memory optimisation of the software. Since this software project is relatively large in size and is constantly handling vast amounts of data, the RAM (also known as the working memory) of the PC is put to the test. Each time the user loads a LabVIEW project, all functions and related parts are loaded to the working memory by default. This can occupy most of the available RAM capacity. The probability of running out of RAM during an experiment set-up is real if the memory usage is not optimised properly or the amount of RAM is too low to begin with. If LabVIEW runs out of memory, it can crash, and thus potentially disrupting any ongoing recording set-ups.

If the available RAM saturates, the computer will start using the hard drive space for "virtual memory". This does allow some buffer. However, the hard drive works much slower than RAM, which causes the PC to run slower when more virtual memory is used from a hard drive [51]. LabVIEW can use up to 2 GB of virtual memory by default, but the amount can be increased if necessary. The virtual memory was increased to 4 GB in this thesis by accessing Windows 7 system settings of the PC.

To combat insuperable crash of LabVIEW due to memory filling up and ensuring efficient RAM usage, dynamic functions and dynamic memory allocations were implemented. This calls for planning rarely-used functions to be called dynamically, instead of allowing LabVIEW to load them all statically into the working memory upon each start-up. Dynamically called functions only reserve RAM for the duration they are actively in use. Additionally, some variables can be reserved dynamically by not having their values

constantly in the memory, but instead by dynamically allocating them RAM capacity and releasing the associated memory slot after they are not in use anymore.

3.3.3. Accuracy and Precision

Precision is a measure of random variability in results and how close each recording is to the average of all values. Accuracy on the other hand is a descriptor for how well the measured results correspond to the true value. [29] The precision could be said to be a measure for reproducibility of recordings.

The hardware is one of the most fundamental factors influencing the accuracy and precision. With robust sensors and electrical components interfacing between the sensors and the PC, it is possible to minimise instrumental noise and artefacts. The DAQ devices were carefully evaluated (as covered in Chapter 3.1.1) to assure the accurate transmission of all input signals, especially the ERG signal, to the PC. The DAQ devices have an effect mainly on the precision described by the LSB of the DAQ device. A device with higher bit-number will have a smaller LSB resolution. For instance, NI PCI-6221 has 16-bit analogue input resolution and NI PCI-6024E has 12-bit input resolution. Thus, the former is more suitable for measurements requiring more precision. Accuracy can also be affected by the DAQ devices and was catered by minimising the crosstalk effects (as discussed in Chapter 3.1.1). The channel scanning order of multiple analogue inputs was adjusted from low voltage input to high voltage input, and an empty channel was sampled right after high voltage input channel to account for settling time of the high voltage input channel. An example of high voltage input is the thermistor, which records voltage in the range of a few volts.

In addition to input data, output needs to coincide as accurately as possible with user-set values. The internal base clocks of the DAQ devices dictate the temporal resolution in which signals can be generated as output. Since the intensity of the light stimuli is modulated by adjusting the duration of an on/off pulse to the laser, it needs to generate pulses in the range of fractions of a millisecond. The highest frequency internal base clock that NI PCI-6221 has is 80 MHz. It can theoretically produce a light pulse half of the frequency's reciprocal [37], hence, 25 ns. On the contrary, NI PCI-6024E, has to work with a base clock of 20 MHz, lowering its minimum pulse duration down to 100 ns. Because of the difference, NI PCI-6221 has been selected for the device in charge of generating light stimuli, due to its better performance for shorter duration light pulses. In both cases, an inaccuracy of one base period exists. Thus ±12.5 ns for PCI-6221 and ±50 ns for PCI-6024E.

3.3.4. Pulse-Width Modulation and Pulse Sequences

A possibility to produce different types of light stimuli was implemented for the new software. Although a flash or pulse stimulus shows clearly the characteristic ERG waveform discussed in Chapter 2.3, it might be by no means the best or only option for a light stimulus used for retinal temperature estimation. It is also unclear which type of light stimulus would be optimal for prospective clinical trials. Therefore, two auxiliary pulse stimuli were added: sine wave and square wave options. As the light source, which LabVIEW needs to communicate with, only accepts digital information, the light stimuli can be augmented through a method called pulse-width modulation (PWM).

The difference between analogue and digital light sources is that an analogue light source can produce any output power in its range, whereas a digital light source can only produce discrete values, in some instances just on and off values. In the case of this thesis, the stimulus light source is a laser that receives only on and off values. PWM tackles this problem by producing an observed wave-like output signal by varying the width of a pulse temporally [52]. When the frequency of the PWM pulses is selected high enough, it can exceed the critical flicker frequency of the perceiving eye, causing the light to appear as continuous instead of individual on and off pulses. The critical flicker frequency is at most approximately 60 Hz in any kind of lighting [53]. In the software, sine and square wave PWM signal output possibilities were added. Hence, a sine wave modulated PWM signal is produced by varying the width of the digital pulses. The resulting illumination observed by the eye is a continuous light output varying in intensity like a sine wave. Both sine and square waves were selected to be produced at a default switching rate of 10 kHz, far higher than needed to exceed the critical flicker frequency. For an illustration of how the PWM signal works in electronics, see Figure 9. The sought effect is similar to that of attempted with the stimulus light.

An additional light stimulus type identified for more human needs is pseudorandom sequence. In this option, a pulse stimulus was used, but instead of repeating the stimulus at exact time intervals, it was coded in to be done in a pseudorandom manner. Once the user inputs a time range for the pseudorandom stimulus, the software, using random seed, generates an arbitrary decimal number belonging to the user-inputted time range. This number defines in seconds the amount of time the software waits between two consequent light stimuli. The advantage of this might include a more comfortable experience for patients, as the flashes of light would appear to occur randomly.

In addition to providing different wave options for a light stimulus, it was also identified that it would be useful to produce planned and timed light stimulus sequences. These consist of any of the stimuli types and can be saved and loaded from a file in the interest of reusing earlier experiment set-ups. Readily reusable pulse sequences allow the end-user to save time and standardise the experiments.

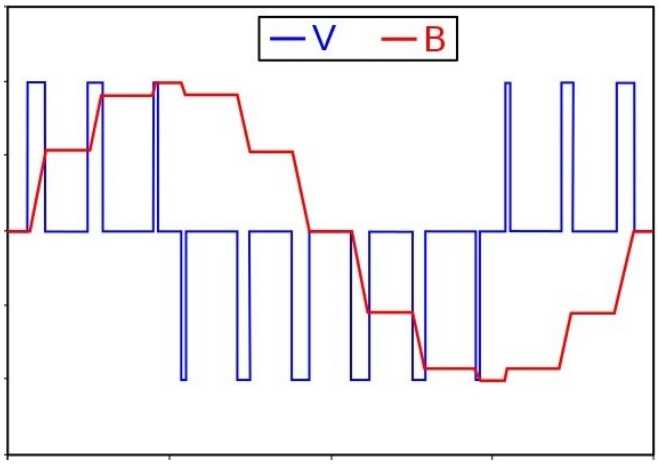

Figure 9. A sine wave modulated PWM signal. The blue plot (V) visualises the output from a voltage source as series of rectangular on and off pulses modulated in width. The red plot (B) is the induced sine-like current in the inductor. When a light source is PWM modulated, similar effect is sought by ensuring the switching rate exceeding the critical flicker frequency. [54]: modified.

3.3.5. Automation

The automation of some of the software features was considered in desire of reducing the amount of attention required by the end-user. A few automation features were included: estimation of depth of anaesthesia with respiratory rate, implementation of a PID controller, signal amplifier reset, and acquiring and saving snapshots from a fundus camera during stimuli. These are further discussed in this chapter.

During the *in vivo* animal experiments, anaesthesia is used to stabilise the animal subjects. Using anaesthesia has also required monitoring the animal with a piezoelectric sensor that records movement of the subject. From the movement it is possible to determine the respiratory rate which is desired to be kept between 1.5 and 3 Hz during any experiment. When the breathing falls within this range, the mouse is not in too deep anaesthesia to risk being deceased nor in too shallow to wake up. As this evaluation is done manually in the previous set-up, it was automated to determine respiratory rate and notify the user if it is out of the range. The respiratory rate was inferred through passing the voltage

output of the piezoelectric sensor to a MATLAB function which performs a frequency analysis. The piezoelectric sensor data is recorded in real time in segments of two seconds at a rate of 1 kHz and loaded into a buffer of 10 seconds of data and then passed on to a MATLAB function. The function computes a discrete Fourier transform through an algorithm known as fast-Fourier transform (FFT) on the data from past 10 seconds. The respiratory rate is then extracted with a peak finding function and the highest peak in the low-frequency range (< 50 Hz) is selected as the rate. Additionally, the function computes from given user-defined range (1.5 to 3.0 Hz by default), whether user needs to be notified with a warning. The warning can be either: no warning, too low a frequency or too high a frequency, which is passed forward to LabVIEW and displayed on a front panel LED to warn the user and prompt into adjusting the anaesthetic delivery.

Heating the retina was another factor that was automated. Although automating the retinal heating has very little purpose in early experiments due to the uncertainty in the retinal temperature model, it does serve a purpose in the future if clinical trials become reality. Therefore, a rudimentary version of this automation was included in the software. The automation involves tools for giving the user to set a selected temperature for the retina, and through a closed feedback loop the software adjusts output voltage of the heating device to reach the set target temperature. This process is steered with a controller that adjusts the output based on the computed difference of expected setpoint temperature and measured retinal temperature estimation.

A well described and widely used controller is so-called proportional-integral-derivative (PID) controller. For example, many thermostats rely on PID controllers. The PID controller takes in an error term as an input and adjusts an output based on three terms: proportional, integral and derivative terms. Some variations exist, sometimes only accounting for one or two of these three terms. For example, P-controller utilises only the proportional term and PD-controller the proportional and derivative terms. The proportional term is relative in its output value to the input error. The integral term takes into account past output values by integrating the error term. The integral term contributes to the overshoot of reaching the setpoint value. Finally, the derivative term predicts future error by evaluating the rate of change of the error term. All three terms are given gain values, K_p for proportional, K_i for integral and K_d for derivative. The gains adjust the relative effect of each term. [55] A visual illustration of the PID controller diagram is shown in Figure 10.

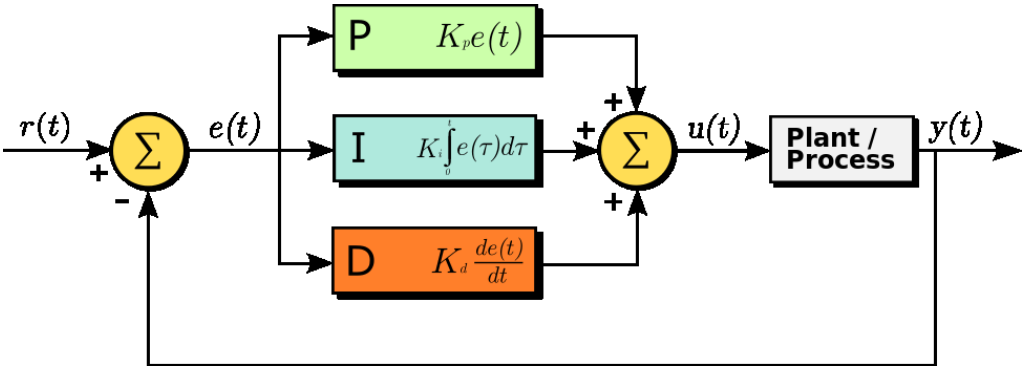

Figure 10. PID controller illustrated. In the case of the RPE heating device *r(t)* is the setpoint temperature and *y(t)* is the estimated retinal temperature. The difference of *r(t)* and *y(t)* produces the error term *e(t)* on which the proportional (P), integral (I) and derivative (D) terms are applied on and summed to produce the process variable, *u(t)*. The process variable is effectively the output voltage of the heating device. The plant contains an equation describing the relation between the controller output (input of the plant) and output of the system, *y(t)* (retinal temperature estimation). [54]

In this thesis, a P- or proportional-controller was implemented as a rudimentary solution. P-controller is a variation of the PID controller, only considering the proportional term as presented in Equation (1). However, it is possible to add the integral and derivative terms to this solution at any point. Building a complete PID controller would require good knowledge of the plant (see Figure 10). Building a well descriptive model for the plant, on the other hand, would require much more reliable and accurate data, as well as numerous experiments before being able to precisely describe the plant with a mathematical equation. Minimising the error of the temperature estimation would be of paramount importance in this. On the other hand, tuning all three terms of a PID controller would still require multiple experiments. The equation describing P-controller is presented in Equation (1).

$$u(t) = K_p e(t) \tag{1}$$

Since the typical voltage range of the heating device has been around 0 to 1 volts in heating experiments, the K_p gain was set to 0.2 arbitrarily. The controller outputs zero voltage when the set temperature has been reached and otherwise oscillating temperature based on the error term. This rudimentary controller mostly serves as a placeholder for future implementations.

Two automated operations featured during ERG recording, in parallel to piezo recordings and P-controller, are the amplifier reset and fundus camera snapshots. The former sends

digital output signal to the ERG signal amplifier each time an iteration of a recording loop reaches its end, but before the next iteration begins. As a result, the amplifier adjusts the signal baseline to zero level. The amplifier also contains a physical button for the reset which has been the means of manual adjustment previously. During ERG recordings, the fundus camera snapshots are acquired and saved as Portable Network Graphics files (PNG file extension). The acquisition takes place at the start of each stimulus.

3.3.6. Safety, Error Handling and User-friendliness

In order to prevent accidental turn-on of the IR heating laser, the software was equipped with a safety feature: user must first turn on the heating with a "start heating" button, then select an output power and confirm the choice by clicking "set". Safety becomes especially relevant in future when clinical trials are pursued, since the developed software deals with heating the retina with a laser capable of producing tissue damage. Therefore, safety is introduced also in the discussion part of this thesis in Chapter 5.2.

The error handling of the software was in large parts catered by including as many conditional structures as possible which check for errors and inform the user on these. For instance, if no input channel for thermistor was selected by the user, the subVI in charge of thermistor recording prompts the user to select a channel. If the subVI was not supplemented with such a conditional structure, LabVIEW would instead yield a cryptic error message without any information on which I/O channel was faulty. Another way to improve error handling capacity was to set an acceptable range of input values for various controls to prevent crashes due to illogical user input values. One of these is the voltage range at which ERG is recorded. The DAQ device is capable of recording a range of -10 to +10 V. Thus, the user can modify the input range of the ERG recordings within this range.

The considerations taken in designing a user-friendly software were to keep the front panel of LabVIEW as pithy as possible. The need for each control and indicator being displayed were carefully evaluated, and less useful ones were either completely hidden or moved behind a tab, where the user would need to navigate to see the additional information. The controls, indicators and other visible items on the front panel were named with the aim of making them as self-explanatory as possible. Information was also provided for some of the front panel items by placing textboxes next to them or by adding the information into their properties. The properties of any control or indicator in LabVIEW contain an option for documentation which is displayed to the user when they hover the mouse cursor over the control or indicator.

3.3.7. Verification and Validation

To prove that the software works, it is important to validate and verify it. Validation is concerned with confirming the source code functioning as intended by the developer. Verification, on the other hand, involves making sure the software satisfies end-user expectations. [39] Before implementing any source code, a prototype was created by designing how the software would look like in the perspective of an end-user. At this time, no functionality was yet implemented. The prototyping was designed using LabVIEW VI's front panel objects. A walkthrough of the planned functionality of the front panel objects was done with the prospective end-users and feedback gathered. The purpose of the prototype was to gather direct input from the research team end-users on the types of features which should be included in the final build and if the planned direction of the implementation was the desired one.

Considering that most of the validation takes place in conjunction with the development process, I have also identified the importance of unit testing to make the validation of the software better suited for future changes. Unit testing is a validation method where a unit testing function is given a set of inputs and expected outputs. The unit test then runs the program (VI or subVI) under test by wiring these inputs and comparing resulting actual outputs to the pre-defined expected ones. If they do not match, an error report is generated, thus letting the developer know of the issue in order to start meticulous debugging.

LabVIEW is equipped with a unit testing possibility: LabVIEW Unit Test Framework Toolkit. Therefore, a unit test for several subVIs of the software were created. These unit tests can be run at any point during the development of the software to validate the results. This is especially useful once new developers overtake the software project later, as running the same unit tests with one click is possible.

In addition to the unit testing and other validation during the project implementation, also long-running trials are required to fully benchmark the software's capabilities. In this project, this is handled with bench testing and actual experiment with a real murine subject (described in Chapter 3.2). The bench testing includes running the software as if there was a subject involved, but without getting any meaningful recording results. This is useful in benchmarking the software's general functionality. Unit testing can only detect problems within VIs, but these kinds of benchmarking tests validate the entire software and the interdependencies between various VIs. When conducting the benchmarking with an actual patient, such as mouse, also meaningful results can be

achieved and the software's capability of performing correctly in real time analyses is validated.

3.3.8. Documentation

In any software, one of the most important considerations is proper documentation which can close the knowledge gap between the previous developer and a new developer attempting to build more functionality on the software. The documentation includes logically named labels for objects visible in the software for end-user, as well as, documenting the code itself. The latter includes in-code comments in the block diagram of the VIs which allows developers to understand rationale for the need of the segment of code. Separate documentation help files were created to provide more comprehensive information of various functions and objects. The help files include list of all inputs and outputs of the functions, used variables, as well as snapshots of the code and the front panel display.

Separate documentation files were composed of the entire software for both end-user and developer. The end-user documentation entails detailed instructions on the function of each front panel object of the developed LabVIEW program. The file also includes general depiction of the software and instructions for getting more out of the software's various functionalities. For developers, more thorough documentation was written. This involved overall depictions of various subVIs and dependencies, as well as recommendations and a brief list of a few sources of literature found useful during development. The developer documentation is attached in appendix A.

4. Software Implementation Results

In this chapter the results of the software engineering are discussed. The emphasis here is in qualitative review of the problems and goals the software addressed. Therefore, this section is divided into three principal chapters. The first one, Chapter 4.1, investigates the visible user-interface; in this case the front panel. The second principal topic, Chapter 4.2, discusses the results of bench testing and qualitative evaluation. Finally, Chapter 4.3 entails the outcomes of the experiment set-up where an actual mouse subject was purposed.

4.1. User-Interface

The user-interface visible to an end-user contains the front panels of Main VI and a VI referred to as Second screen VI, which is opened by Main VI on a secondary monitor of the PC. If only one monitor is in use, the Second screen VI opens cascaded on the primary monitor. For a front panel look of the Main VI and Second screen VI, one can refer to Figure 11 and Figure 12, respectively.

The software's most important features are giving stimulus outputs, recording ERG responses, and controlling the RPE heating device. The stimulus output details can be seen in the upper-left portion of the front panel of Main VI ("stimulus parameters", see Figure 11: A). User can choose the stimulus type from four options: single flash, PWM sine- or square-wave, or pseudorandom (as explained in Chapter 3.3.4). Additional controls include adjusting the duration of stimulus as well as selection of the light source, voltage range and sampling rate. All of this can also be preprogramed by the user through the "design stimuli" menu, as explained in Chapter 3.3.4. The RPE heating device controls are located just below the "stimulus parameters" (see Figure 11: B). User can select through a multiple-choice box to either initiate heating immediately, synchronised with next ERG response, or at specified time point. User can also choose from the drop-down list either "crude" or "sophisticated" heating. "Crude" heating outputs user selected voltage values, while "sophisticated" heating contains the P-controller implementation described in 3.3.5. The recorded ERG responses appear on "list of responses" multicolumn listbox on the bottom right corner of Main VI (see Figure 11: C) and are drawn to the large "running responses" graph on the top right of the front panel (see Figure 11: D).

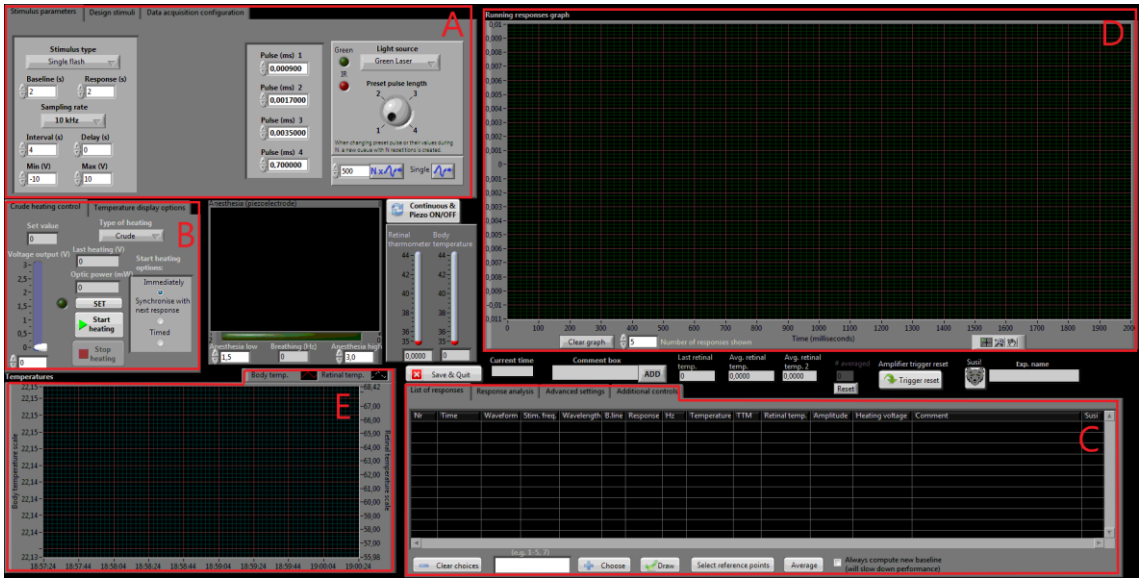

Figure 11. View of the Main VI front panel. Highlighted are five regions enclosed by red boxes. A) the "stimulus parameters" are contained here, as well as tabs for "design stimuli" and "data acquisition configuration". B) contains the RPE heating controls. C) highlights the "list of responses" multicolumn listbox. D) responses are drawn on this "running responses graph". E) body temperature and retinal temperature are graphed here.

The software user interface is also accompanied by a set of graphs, in addition to the mentioned "running responses" graph. On Main VI, one can observe the body temperature and retinal temperature graph located in the left-bottom corner (see Figure 11: E). It draws, using a separate scale for each, the recorded body temperature and estimated retinal temperature over a duration of three minutes. The latter temperature can be chosen to be drawn either by each estimated value, as average of past five values, or as running average.

Various real-time analyses on the data are performed before each response is plotted in the graphs. These analyses include possible user-defined filtering, baseline correction, as well as retinal temperature estimation. Auxiliary graphs related to the ERG responses are found on the Second screen VI (Figure 12). These include graphs for the baseline correction, real-time response, averaged response, and response amplitudes. The real time response graph (left-most, middle; see Figure 12: A) displays the signal in real time during its acquisition. Two graphs are located right below it (see Figure 12: B). The left one ("baseline prediction") is where the baseline prediction is plotted. The baseline correction methods include either simple linear baseline correction or an autoregressive baseline correction. This is then subtracted from the filtered ERG response and plotted to the

"corrected response" graph on the right side. The only filter option included in this version of the software is a 50 Hz Notch filter which effectively filters out 50 Hz mains frequency. More detailed functionality of both baseline correction and filtering is given in Chapter A.2.7.2 of the Appendix A. In the graph of upper-left corner (see Figure 12: C), averaged signal over defined number of responses is drawn. User can also select the x-scale value range and select to display estimated t_p values on it. Lastly, on right-bottom corner (not visible in Figure 12), resides the amplitude graph. It keeps track of the amplitude values of selected number of responses. Inferring downward or upward trends of amplitude is hence possible from the graph.

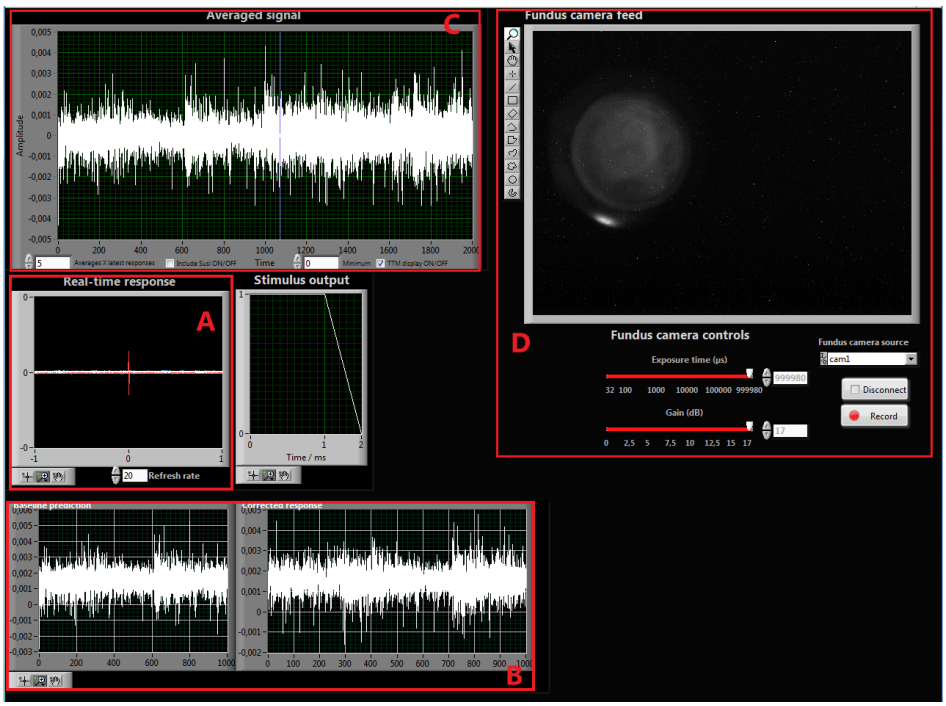

Figure 12. View of the Second screen VI front panel. Highlighted are four regions enclosed by red boxes. A) contains the "real-time response" graph. B) entails "baseline prediction" and "corrected response" graphs. C) "averaged signal" graph where an average of ERG responses is plotted. D) fundus camera controls and the "fundus camera feed" showing view from the fundus camera.

Fundus camera feed is displayed on the Second screen VI on the upper-right corner (see Figure 12: D). During each ERG response, a snapshot is automatically acquired and saved to the data file directory with a timestamp. The user can also manually acquire a snapshot by clicking "Record". The feed quality is affected by user-selected gain and exposure time values, encoded to the software and for which controls are located just below the camera feed.

4.2. Bench Test and Qualitative Evaluation

The software was bench tested without any live subjects. The purpose was to unveil any potential errors and sources of error before proceeding to the *in vivo* experiment set-up. This verification process included validating individual VIs through unit testing as well as the entire software by comprehensive testing of the primary functionalities in Main VI. Finally, for the purposes of quantifying the successfulness of the software implementation, values for key metrics were attained. The metrics included evaluation of execution speeds, the software's full size, cyclomatic complexity, modularity index, size of the documentation, and overall complexity of interdependencies between functions. The results of the quantitative evaluation have been gathered to Table 7 and are based on Sommerville's book on Software Engineering [39].

Table 7. Results of quantitative bench testing

Description of quality	Results
Start-up time	19.32 s (N = 5)
Software size	15.8 megabytes
ERG recording loop mean delay // standard deviation	117.962 ms // 3.009 ms (N = 1812)
Total documentation size	17 111 words
Unit testing code coverage	46.3 %
Number of nodes	3982
Total code size on disk	945.6 kB
Cyclomatic complexity of Main	168
Fan Out	49
Modularity index	1.88

The most crucial structure of the software was considered to be the while-loop performing ERG recording (located in Main VI). Slow performance of the *ERG recording loop* can cause delays in case multiple consequent stimuli are planned. For instance, if the user wishes to repeat each three second a recording lasting for two seconds, then a delay of more than a second will cause the loop to perform in over three seconds, falling short from the intended three-second stimulus interval. Another important structure is the *real-time analysis loop*, as explained further in Chapter A.2.4 of Appendix A. This loop can handle slightly longer delays. However, significantly long delays can present themselves as slowed down refresh rates of the various graphs and indicators, causing the user to receive delayed information of the stimuli. In the end, only the *ERG recording loop* was

bench tested for mean delay and standard deviation of the delay. The loop had a mean delay of 117.962 ms and a standard deviation of 3.009 ms. The results were acquired by using a two-second repetition interval with one second of baseline and one second of response recording (altogether two seconds) at 10 kHz sampling rate. The amount of repetitions for this was N = 1812. The delay means that when a user initiates multiple recordings consecutively, the interval between each subsequent response is on average at least 117.962 ms higher than the set duration of the recording (two seconds).

The quantitative evaluation also revealed that the entire software size is 15.8 MB, which includes custom graphics. With only source code, the software sized at 4120.5 kB. This is relatively large, which is why the software takes an average of 19.32 seconds (N = 5) to load when starting it. In LabVIEW, nodes can be described as the block diagram objects, such as functions, VIs, sequences, and loops [56]. The size of the software and number of nodes, or in text-based coding languages the number of lines of code, is a good indicator of expected error occurrence and complexity, since more code signifies higher probability for mistakes [39]. The number of nodes was 3982, which corresponds approximately to same number of lines of code in text-based coding languages.

The documentation was in total 17 305 words, which can be broken into two parts: end-user documentation (or user manual, 6 971 words) and developer documentation (10 334 words). The documentation in Appendix A includes only the developer documentation. The end-user documentation is accessible together with the software though.

The unit testing covers an estimated 46.3 % of all the project source code, revealed by LabVIEW's unit testing framework. The percentage is acquired by dividing the number of executed subdiagrams by the total number of subdiagrams in the entire software project [57]. Subdiagrams are the structures and loops in the VI block diagram [57]. For example, a case structure of two cases contains two subdiagrams, a for loop one subdiagram, and so on. The code coverage is only 46.3 % due to the vast amount of subdiagrams in Main VI, for which it would not be feasible to develop a unit test within reasonable amount of time. Additionally, some of the subVIs containing I/O functionality do not have associated unit tests due to the difficulty of implementing unit tests with functions that require I/O connectivity.

Cyclomatic complexity is a common metric used for evaluating the complexity of a software and it is related to the maintainability, reliability and comprehensibility of the software. It measures the number of linearly independent of paths through the program's source code. In principle, decision-points and conditionals, such as if-statements, increase the number of linearly independent paths and thus the cyclomatic complexity. [39] In

LabVIEW, an additional metric for maintainability is the modularity index, which is a division of the total number of VIs by the number of nodes in the entire source code of the software project [58]. The cyclomatic complexity for the Main VI was evaluated to have an index value of 168, and the modularity index was 1.88. In LabVIEW the default recommended upper limit for cyclomatic complexity is 20, although the original developer of cyclomatic complexity, Thomas McCabe (1976), suggested in his publication [59] to keep the complexity at less than ten by splitting more complex programs into more subroutines. Software with values higher than the recommended upper limit can be difficult to maintain, comprehend and debug. The modularity index, on the other hand, is recommended to be three or higher in the LabVIEW style book [58]. An index of 1.88 is lower than the recommended, hence implying a need to create more subVIs for more efficient functioning of Main VI.

4.3. Testing in an Experiment Set-up

The practical testing, which included actual experiment set-up, was conducted on two different occasions by anaesthetising mouse subjects (see Chapter 3.2). The first trial was more focused on the testing of basic functionalities during an actual experiment on a murine subject. Whereas, the second trial was aimed at validating the controlled RPE heating using the IR laser and testing performance under a usual experiment set-up.

On the first experiment, nearly all features were discovered to function as desired. A few minor errors were identified and fixed after this experiment. Additionally, a few small changes and improvements were performed based on the user-feedback received. These, though, did not have a significant impact to the software's architecture or features. Examples of such changes include changed front panel layout as well as adjusting the position and visibility of controls and indicators.

The second experiment, where the heating device control was tested, occurred approximately two months after the first one. Additionally, the use of a higher wavelength (IR range) stimulus light and the functionality of the respiratory rate estimator were tested. All these features worked mostly flawlessly, although more improvement points were attained through user-feedback and errors detected that were not noticed during bench testing. Additionally, it was discovered that the temperature recording was heavily impacted by 10 kHz sampling rate during ERG signal acquisition due to crosstalk effect.

The changes done after the second experiment were as following. The heating control received an additional safety feature: user must first input desired output and confirm it

separately by clicking a “set” button. A calibration file was also bound to the heating device to make its output values more accurate. The stimulus protocol was improved by adding a feature enabling repetition of the set protocol a desired number of times. The thermistor input was also moved from NI PCI-6221 to be recorded on NI PCI-6024E instead (the set-up reported in Chapter 3.1.1), to minimise crosstalk effects. Additionally, the scanning order of channels was adjusted to be from lowest voltage to highest in effort to minimise crosstalk as well. The specific details of all the mentioned added functionalities have been described in the methods Chapter 3.3, and Chapter A.2 of Appendix A. Other amended features were more trivial, such as an indicator for displaying system time and adjusting input ranges for controls.

5. Discussion

In this chapter, mainly suggestions on future directions of the software are discussed. Within the time limits of this thesis, not all built-in features of the software were fully developed, and hence are covered here. The developer documentation (Appendix A) also outlines recommendations for future development of the software, although it is oriented towards best-practice sharing.

5.1. Future Considerations and Suggestions

There are a few identified improvements that would be possible to undertake in the software in years to come. These relate roughly to three distinct categories: the software optimisation regarding the hardware, prospective clinical trials, and improvement on software complexity.

The hardware could be still improved. The current PC runs on 64-bit Windows 7 with 8 GB of RAM and a hard disk drive, which are bottlenecks of the PC. Sometimes the amount of RAM is not enough to smoothly run the software architecture, requiring virtual memory allocation from the hard disk. This tends to slow down the performance significantly, occasionally even crashing LabVIEW as a result. It would be suggested to invest in the hardware by doubling the current RAM capacity and replacing the hard disk drive with a solid-state drive, which has much faster read/write rate than a hard disk. The current processor contains four cores and eight threads at 2.67 GHz. A processor with even more cores and threads could be considered to facilitate fast parallel tasks should the software continue to be developed. Especially real-time analyses require computing power which could be increased with the use of a more powerful processor. Also, PC graphic card's GPU can be utilised to speed up the analysis sequences.

It seems apparent that the high-frequency sampling desired on the current DAQ devices impede the accuracy of recordings. The thermistor input seemed to have notable crosstalk at high frequencies, causing inaccurate temperature recordings. The crosstalk is partially due to high differences in measured voltages across different input channels, not allowing enough settling time for high thermistor voltage. To remedy the situation, it should be considered to upgrade the DAQ devices to allow better accuracy for these multichannel recordings at high frequencies. A DAQ device with high maximum sampling rate and short settling time would be ideal. Alternatively, an additional DAQ device for solely the thermistor input would solve the issue. In this case, the thermistor would also elicit

crosstalk effects on the other input sensors. While considering possible DAQ device upgrades and purchases, it could be beneficial to enable the software to become portable and feasible to run on a laptop computer. The portability aspect would require a modular DAQ device which plugs into a laptop through USB or Ethernet port. Additional benefits of a modular DAQ include the ability of installing various modules to it that can function independently of each other, allowing higher number of I/O channels and timing engines. National Instruments sells a product like this with the name CompactDAQ [60].

A potential issue regarding temporal resolution of repeated stimuli arises from the delay of 117.962 ms between consequent recordings. This is not a problem with the current experiments, since repeat intervals are almost always set higher than the amount of recorded signal. The problem arises when the recorded signal length (baseline and response combined) is less than 117.962 ms apart from the set repeat interval. This problem could be solved in various ways, for example through compactDAQ that could have a dedicated timing engine for the ERG signal and stimuli, instead of sharing the engine with various other sensors and essentially requiring synchronisation.

As was discovered in bench testing of the software, the cyclomatic complexity is unusually high: 168. It stems from several if-conditionals arising from error-handling functionalities and allowing simulated user-testing. The complexity could be reduced by implementing more of the code in subVIs which can be each independently unit tested. The cyclomatic complexity is also a measure of maintainability, and hence an indicator of how easy it is to understand the source code, develop the software and detect mistakes. Testing such a large and complex software is difficult and nearly impossible in such a way that mistakes would be uncovered. In addition, the modularity index was 1.88 which also falls short from the recommendation of three or higher. This implies, similarly to the value of cyclomatic complexity, on the need of subdividing larger segments of source into subVIs.

5.2. Safety Concerns

With murine recordings lie different safety requirements than with human-oriented clinical trials. If the experiments are to be brought to clinical trials in the future, few safety concerns would need to be tackled. First of all, it would be likely that anaesthesia would not be applied at all. As a result, the patient would be more unpredictable, causing themselves potential risks during the RPE heating therapy. One of these risks is eye

movement, which can occur by accident and cause the heating laser's target spot on the retina to abruptly change or even relocate outside the retina.

One potential way to combat eye movement is to either switch off or correct the laser's path to follow the desired spot on retina. For both methods, it is necessary to be able to automatically infer the changing eye position. Two potential ways can be foreseen to do this. The first option is inferring the eye movement by recording the eye movement externally, by e.g. following the movement of pupillary. The second option is to utilise the fundus camera feed information. The fundus camera approach is more favourable in this case, since it already exists in the current device set-up. However, the device set-up might change for clinical trials and not contain the same fundus camera. Hence, implementing this method in the current device set-up might not reflect the set-up in clinical trials. Additionally, implementation of any kind of eye movement correction requires far more data, and potentially should be undertaken with machine learning algorithms to automate the recognition of retinal features. For instance, a method of optic nerve localisation has been implemented already by Hoover *et al*. [61], where the optic nerve can be recognised fairly accurately from retinal images by examining converging blood vessels. Using the optic nerve as a point of reference in relation to the IR heating spot, an adjustment system for the laser spot could be developed.

Currently the software does not include any feature which would prevent retinal temperatures from exceeding beyond safe levels. This can be implemented through more reliable retinal temperature estimation model. If the retinal temperature was to rise over a pre-defined safe limit, the software could automatically switch the heating laser off or enforce much lower power. A more robust PID controller would also optimise the adjusted power output. Possible defects also exist with the retinal temperature estimation which might not work ideally on all individuals. Poor contact with electrodes or incorrectly placed heating laser and stimulus light could also contribute on the retinal temperature estimation yielding incorrect values. Therefore, the output power could be programmatically limited to only certain safe values that could then be only exceeded once a healthcare professional performing the treatment manually approves it.

5.3. Feature Extraction

Currently the only extracted feature for retinal temperature estimation in the implemented software is the time-to-peak (t_p) of b-wave. It could be advisable to also implement other features to the retinal temperature estimation methods, such as the possibility to include

a-wave and spectral sensitivity information. In the current *in vivo* experiments, only dark-adapted ERG recordings using dim flash stimuli have been performed. This is due to the better SNR of rod responses, making it a simpler starting point for developing models for retinal temperature estimation. For this reason, cone-dependent features have not been refined. If moving to clinical trials, it would be paramount to include models more suited for background light conditions in which cone responses are predominant. As was discussed in Chapter 2.3, the S-cones would be, in theory, the most isolated cones to stimulate. However, the number of S-cones on the retina is small in comparison to the other cone types, making it difficult in practice to isolate S-cone responses. In reality a single cone type isolation might not be entirely feasible. Elimination of rod components is possible, though, by using bright background lighting or high enough stimulus frequency in fERG.

In AMD, the macular region is degraded. This can cause ERG signal's SNR to decrease, making recordings more difficult and less precise. This calls for more development of ERG features to account for it, as now the software will be used in mouse experiments of healthy retinas.

The current software contains a subVI called "Feature extraction". It receives as input the used stimulus light source (green or IR wavelength), acquired ERG signal, acquisition frequency and b-wave peak fit parameters. It then passes these to a MATLAB function that performs currently only a b-wave peak finding and retinal temperature estimation based on this. However, a developer can adjust the subVI and the related MATLAB function to also perform different analysis if the light source is different. Additional features, such as a-wave based retinal temperature estimation can also be added by modifications to the source code of these two subdiagrams.

6. Conclusion

RPE heating therapy is a potential method for treating AMD and possibly other detrimental eye diseases. The research on the topic calls for even more experimentation and repeated trials. ERG is used in this to estimate the retinal temperature to maintain the RPE heating therapy within the therapeutic levels. A powerful and advantageous software is needed to fulfil all the needs for precise ERG-recordings as well as to account for all the identified needs of this set-up. Prior to the start of this thesis, an existing software solution was present but lacking in features and maintainability. The goal was to develop a new software for an RPE heating device set-up as well as to consider future needs. Eight primary needs were identified and implemented, ensuring more automated control over the software, as well as added reliability and maintainability.

This thesis has produced an engineered software solution, which proves to be well validated and fitting for the purposes defined in the introduction of this thesis. The software, furthermore, fits all the current needs for trials with mice, and takes into account future potential demand towards the maintainability, performance, safety and other features. Although issues exist in current maintainability of the software, vast amount of documentation has been written to aid prospective developers. Additionally, some modifications might be necessary to increase the performance, many of which concern the hardware if more functionality of the software is demanded.

Some features that would need to be added to allow even preliminary tests of human transpupillary thermotherapy, are automation and safety features as well as extraction of features from ERG responses. Feedback controlled RPE heating, using a PID controller, would be advisable to implement after more data on experiments is available, since proper plant identification and more fine-tuned controller gains are needed before the controller can be safely used. Safety mechanism dealing with eye movement is also a feature that should be implemented. Machine learning methods exist for recognition of retinal landmarks, which can be utilised in correcting eye movement.

7. References

1. Mariotti S, Pascolini D. Visual impairment, vision loss and blindness 2010 global estimates, and VI and blindness causes [Internet]. Global Data on Visual Impairments. 2010 [cited 2017 Sep 21]. Available from: http://www.who.int/blindness/data_maps/VIFACTSHEETGLODAT2010full.pdf?ua=1

2. Wong WL, Su X, Li X, Cheung CMG, Klein R, Cheng CY, Wong TY. Global prevalence of age-related macular degeneration and disease burden projection for 2020 and 2040: A systematic review and meta-analysis. The Lancet Global Health. 2014;2(2):e106-16.

3. Brown GC, Brown MM, Sharma S, Stein JD, Roth Z, Campanella J, Beauchamp GR. The burden of age-related macular degeneration: A value-based medicine analysis. 2005;103:173–86.

4. Galloway NR, Amoaku WM, Galloway PH, Browning AC. Common eye diseases and their management. 4th ed. Postgraduate Medical Journal. Springer International Publishing; 2016. 196 p.

5. Facts about age-related macular degeneration [Internet]. National Eye Institute. 2015 [cited 2017 Sep 22]. Available from: https://nei.nih.gov/health/maculardegen/armd_facts

6. Mainster MA, Reichel E. Transpupillary thermotherapy for age-related macular degeneration: long-pulse photocoagulation, apoptosis, and heat shock proteins. Ophthalmic Surgery, Lasers and Imaging Retina. 2000;31(5):359–73.

7. Haldin C, Nymark S, Aho A-C, Koskelainen A, Donner K. Rod phototransduction determines the trade-off of temporal integration and speed of vision in dark-adapted toads. Journal of Neuroscience. 2009;29(18):5716–25.

8. Heikkinen H, Nymark S, Donner K, Koskelainen A. Temperature dependence of dark-adapted sensitivity and light-adaptation in photoreceptors with A1 visual pigments: A comparison of frog L-cones and rods. Vision research. 2009;49(14):1717–28.

9. Pitkänen M, Kaikkonen O, Koskelainen A. A novel method for mouse retinal temperature determination based on ERG photoresponses. Annals of Biomedical Engineering. 2017 Oct;45(10):2360–72.

10. Rodieck RW. The first steps in seeing. 1st ed. Sutherland M, editor. Sinauer Associates; 1998. 546 p.

11. Schunke M, Schulte E, Schumacher U. Atlas of anatomy: Head and neuroanatomy. 1st ed. Wright K, Queenan B, Paltzer V, editors. Stuttgart: Thieme Medical Publishers; 2010.

12. Young B, Woodford P, O'Dowd G. Wheater's functional histology e-book: A text and colour atlas. 6th ed. Hall A, editor. Elsevier Health Sciences; 2014. 452 p.

13. Strauss O. The retinal pigment epithelium in visual function. Physiological reviews. 2005;85(3):845–81.

14. Lang GK. Ophthalmology: A short textbook. 1st ed. Stuttgart: Thieme; 2000. 604 p.

15. McCulloch DL, Marmor MF, Brigell MG, Hamilton R, Holder GE, Tzekov R, Bach M. ISCEV standard for full-field clinical electroretinography (2015 update). Documenta ophthalmologica. 2015;130(1):1–12.

16. Raitta C, Karhunen U, Seppälainen AM, Naukkarinen M. Changes in the electroretinogram and visual evoked potentials during general anaesthesia. Albrecht von Graefes Archiv für klinische und experimentelle Ophthalmologie. 1979;211(2):139–44.

17. Nair G, Kim M, Nagaoka T, Olson DE, Thulé PM, Pardue MT, Duong TQ. Effects of common anesthetics on eye movement and electroretinogram. Documenta ophthalmologica. 2011;122(3):163–76.

18. Veleri S, Lazar CH, Chang B, Sieving PA, Banin E, Swaroop A. Biology and therapy of inherited retinal degenerative disease: insights from mouse models. Disease Models & Mechanisms. 2015 Feb 1;8(2):109 LP-129.

19. Remtulla S, Hallett PE. A schematic eye for the mouse, and comparisons with the rat. Vision Research. 1985;25(1):21–31.

20. Panda-Jonas S, Jonas JB, Jakobczyk M, Schneider U. Retinal photoreceptor count, retinal surface area, and optic disc size in normal human eyes. Ophthalmology. 1994;101(3):519–23.

21. Ding X, Patel M, Chan C-C. Molecular pathology of age-related macular degeneration. Progress in retinal and eye research. 2009;28(1):1–18.

22. Oyster CW. The human eye: structure and function. 1st ed. Sinauer Associates; 1999. 766 p.

23. Meyer LM, Dong X, Wegener A, Söderberg P. Light scattering in the C57BL/6 mouse lens. Acta Ophthalmologica. 2007;85(2):178–82.

24. Forrester J V, Dick AD, McMenamin PG, Roberts F, Pearlman E. The Eye E-Book: Basic Sciences in Practice. 4th ed. Elsevier Health Sciences; 2015. 568 p.

25. Morimoto RI. Proteotoxic stress and inducible chaperone networks in neurodegenerative disease and aging. Genes & development. 2008;22(11):1427–38.

26. Kaarniranta K, Salminen A, Eskelinen E-L, Kopitz J. Heat shock proteins as gatekeepers of proteolytic pathways—implications for age-related macular degeneration (AMD). Ageing research reviews. 2009;8(2):128–39.

27. Boettner EA, Wolter JR. Transmission of the ocular media. Investigative Ophthalmology & Visual Science. 1962;1(6):776–83.

28. Perlman I. The electroretinogram [Internet]. Kolb H, Fernandez E, Nelson R, editors. Webvision:The Organization of the Retina and Visual System. Salt Lake City (UT): University of Utah Health Sciences Center; 2001 [cited 2017 Aug 2]. Available from: https://www.ncbi.nlm.nih.gov/books/NBK11554/

29. Webster J, Clark J. Medical instrumentation. 4th ed. New Jersey: John Wiley & Sons; 2010. 713 p.

30. Bowmaker JK, Dartnall HJ. Visual pigments of rods and cones in a human retina. The Journal of Physiology. 1980 Jan;298(1):501–11.

31. Frishman LJ. Origins of the electroretinogram. In: Heckenlively JR, Arden GB, editors. Principles and practice of clinical electrophysiology of vision. 2nd ed. Cambridge, Massachusetts: MIT press; 2006. p. 139–83.

32. Kong J, Gouras P. The effect of body temperature on the murine electroretinogram. Documenta Ophthalmologica. 2003;106(3):239–42.

33. Mizota A, Adachi-Usami E. Effect of body temperature on electroretinogram of mice. Investigative Ophthalmology and Visual Science. 2002;43(12):3754–7.

34. Lachapelle P, Benoit J, Guité P. The effect of in vivo retinal cooling on the electroretinogram of the rabbit. Vision Research. 1996;36(3):339–44.

35. Armington JC, Adolph AR. Temperature effects on the electroretinogram of the isolated carp retina. Acta Ophthalmologica. 1984;62(3):498–509.

36. Austerlitz H. Data acquisition techniques using PCs. 2nd ed. San Diego, United States: Elsevier Science; 2002. 416 p.

37. DAQ M series: M series user manual [Internet]. DAQ M Series. National Instruments; 2016. Available from: http://www.ni.com/pdf/manuals/371022l.pdf#G7.33344

38. Kularatna N, Institution of Electrical Engineers. Data converters. In: Digital and analogue instrumentation: testing and measurement. 1st ed. Institution of Engineering and Technology; 2008. p. 57–112.

39. Sommerville I. Software engineering, global edition. 10th ed. Hirsch M, editor. Pearson Education M.U.A.; 2016. 816 p.

40. National Instruments LabVIEW (version 16.0) [Internet]. 2016. Available from: http://www.ni.com/product-documentation/53222/en/

41. Introduction to virtual instruments [Internet]. 2011 [cited 2017 Dec 28]. Available from: http://zone.ni.com/reference/en-XX/help/371361H-01/lvconcepts/intro_to_vis/

42. MathWorks. Mathworks MATLAB (version R2017a) [Internet]. MathWorks; 2017. Available from: https://se.mathworks.com/products/matlab.html

43. Differences between multithreading and multitasking for programmers [Internet]. 2014 [cited 2017 Dec 29]. Available from: http://www.ni.com/white-paper/6424/en/

44. Introduction to GPU computing with LabVIEW [Internet]. 2013 [cited 2017 Dec 28]. Available from: http://www.ni.com/white-paper/14077/en/

45. NI PCI-6221 [Internet]. [cited 2017 Aug 11]. Available from: http://www.ni.com/fi-fi/support/model.pci-6221.html

46. NI PCI-6024E (legacy) [Internet]. [cited 2017 Aug 11]. Available from: http://sine.ni.com/nips/cds/view/p/lang/fi/nid/10968

47. Timing engines [Internet]. 2017 [cited 2017 Aug 11]. Available from: http://zone.ni.com/reference/en-XX/help/370466AD-01/mxcncpts/timingengines/

48. NI SCB-68A [Internet]. [cited 2017 Aug 11]. Available from: http://sine.ni.com/nips/cds/view/p/lang/fi/nid/210777

49. NI CB-68LP [Internet]. [cited 2017 Aug 11]. Available from: http://sine.ni.com/nips/cds/view/p/lang/fi/nid/1187

50. Jung W. Op amp specifications. In: Op Amp Applications Handbook. 1st ed. Elsevier Science; 2004. p. 51–88.

51. Spector L. If Windows virtual memory is too low, you can increase it, but there are trade-offs [Internet]. 2014 [cited 2017 Nov 2]. Available from: https://www.pcworld.com/article/2840886/if-windows-virtual-memory-is-too-low-you-can-increase-it-but-there-are-trade-offs.html

52. Holmes DG, Lipo TA. Pulse width modulation for power converters: Principles and practice. 1st ed. Vol. 18. John Wiley & Sons; 2003. 724 p.

53. Kalloniatis M, Luu C. Temporal resolution [Internet]. Webvision:The organization of the retina and visual system. Salt Lake City: University of Utah Health Sciences Center; 2007 [cited 2017 Sep 16]. p. 1067–82. Available from: http://webvision.med.utah.edu/book/part-viii-gabac-receptors/temporal-resolution/

54. No title. By Zureks (Own work) [GFDL (http://www.gnu.org/copyleft/fdl.html), CC-BY-SA-3.0 (http://creativecommons.org/licenses/by-sa/3.0/);

55. Åström KJ, Hägglund T. PID controllers: Theory, design, and tuning. 2nd ed. Isa Research Triangle Park, NC; 1995. 343 p.

56. Estimating code complexity in LabVIEW [Internet]. 2009 [cited 2017 Dec 28]. Available from: http://www.ni.com/white-paper/3324/en/#toc2

57. Code coverage (unit test framework toolkit) [Internet]. 2012 [cited 2017 Dec 28]. Available from: http://zone.ni.com/reference/en-XX/help/372584D-01/lvutfconcepts/utfc_code_cov/

58. Blume PA. The LabVIEW style book. 1st ed. Pearson Education; 2007. 372 p.

59. McCabe TJ. A complexity measure. IEEE Transactions on software Engineering. 1976;2(4):308–20.

60. CompactDAQ [Internet]. [cited 2017 Dec 29]. Available from: http://www.ni.com/data-acquisition/compactdaq/

61. Hoover A, Goldbaum M. Locating the optic nerve in a retinal image using the fuzzy convergence of the blood vessels. IEEE Transactions on Medical Imaging. 2003;22(8):951–8.

A. Appendix: Documentation for Developers

A.1. Introduction

All VIs and subVIs are denoted **in bold** text in this documentation. These files receive a .vi extension in the file name by LabVIEW. Additionally, all variables, i.e. controls, indicators and graphs are denoted as *italic*. For instance:

- **Main.vi** or just **Main** denotes the VI called named Main.vi in LabVIEW directory;
- *Single, Last retinal temp.* and *Running responses graph* are a control, indicator and graph, respectively.

The purpose of this documentation is to provide a thorough insight for a software developer. Having a better understanding of this software project's source code is especially helpful in making further changes to it. This documentation is divided into three principal chapters. This chapter serves as a brief introduction to this documentation. In the second chapter, the block diagram of the software project is inspected. The source code and arrangement of various subVIs are also discussed in the chapter. Lastly, recommendations will be given regarding the future development of the software.

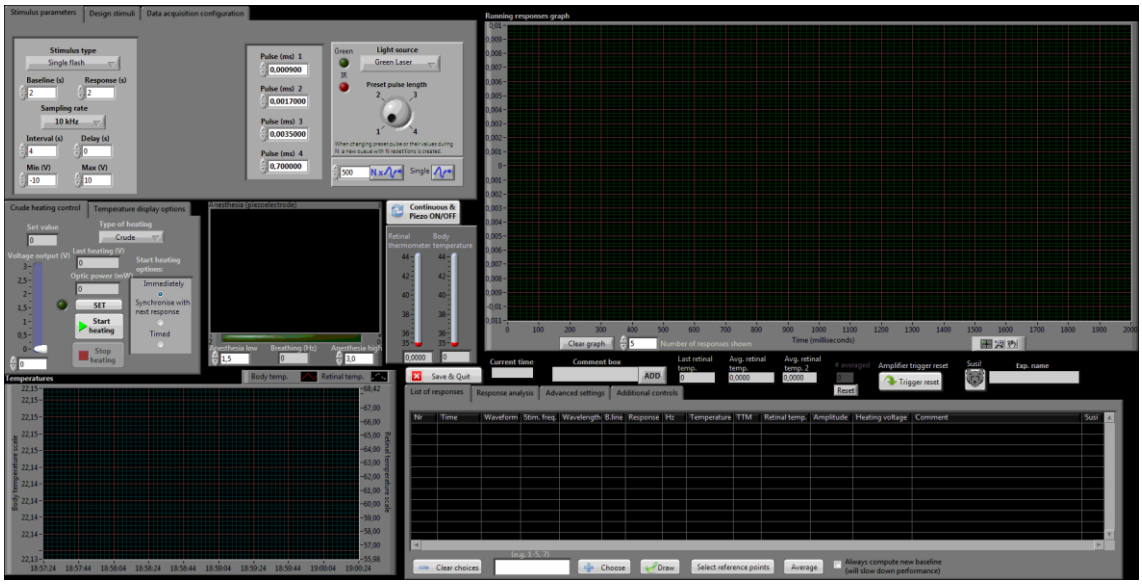

Figure A-1. Front panel view of the **Main.vi**.

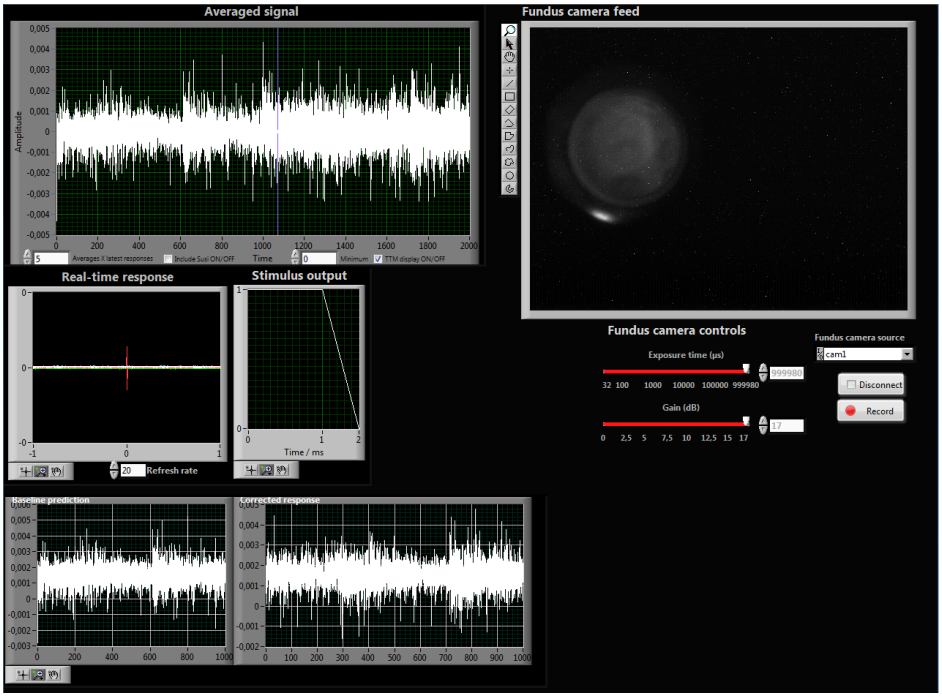

Figure A-2. Front panel view of **Second screen.vi**.

A.2. Block Diagram and MATLAB – Details for Developers

The related LabVIEW VIs and MATLAB functions contain quite a bit of information themselves already. The documentation on both has been implemented as comments throughout the source code. This chapter, however, is more for an overview and understanding the rationale behind larger structures and how they relate to each other.

A.2.1. Start-up and Quit Routines

When **Main.vi** is launched and run, numerous start-up routines take place. First, when launching **Main**, LabVIEW loads all the static subVIs into working memory. Once this step is complete and the user wants to run the program, **Create parameter file.vi** is run dynamically, loading into the memory and releasing it once the user prompt file input parameters have been given. This function sets the data directory location. Depending on if the user opts in for normal run or chooses to create test file with simulated test or not, some front panel options are disabled or enabled. Next, the **Second screen.vi** is launched and maximised on the PC's secondary monitor if one is available. If no secondary monitor is connected, it opens cascaded on the main monitor. After this, **Main** opens a configuration file, if one is available, where various parameters are obtained from, such as last used retinal temperature parameters, channel input configurations and others. The

configuration file, named “Config file” is found on the root of user-defined data directory location.

The front panel of **Main** includes a control labelled *Save & Quit*. Clicking it will result in a dialog box to confirm quitting. Essentially opting in for this option initiates a quit routine. The VI safely waits until the currently ongoing iteration loops finish, and then creates the Config file, if not yet existing, to the data directory. Same parameters as in start-up routine are saved into the Config file, based on currently set values of the software.

Rationale for having start-up and quit routines is to provide the end-user quick access to values used previously by saving them and loading them at the following start-up. A quit routine is also needed to bring the software into controlled stop by closing files, waiting for loops to finish their current iteration and resetting output device values to zero so that they will not be active after the program comes to an end.

A.2.2. In Vivo heating device and ERG.lvproj

When creating a massive library of VIs and functions, such as is the case for this software project, it is useful to build them in a project file. The first step of creating larger software projects in LabVIEW should be creating the lvproj file. It contains all the files and dependencies used for running the program, it facilitates unit testing, and it holds many other useful tools. The project file view of *In Vivo heating device and ERG.lvproj* upon opening the project explorer is illustrated in Figure A-3.

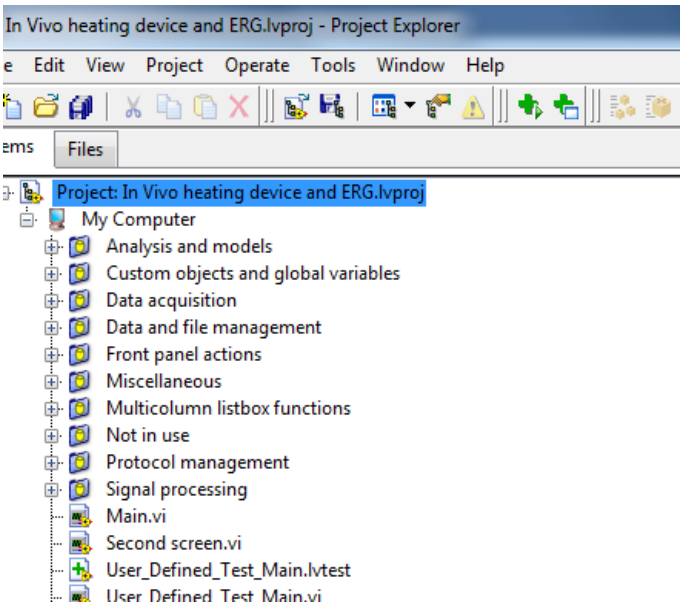

Figure A-3. View of the *In Vivo heating device and ERG.lvproj*.

The project explorer is handy for navigating, renaming or changing the location of different project folders and files. It is also possible to create build specifications and manage dependencies of the project. LabVIEW project explorer will also display any possible conflicts between found files or dependencies and offer solutions. All in all, the project file links everything in a software project together and maintains these links. More details regarding unit testing and the directory management with the project file is discussed in the appendix Chapters A.2.2.1 and A.2.2.2, respectively.

A.2.2.1. Unit Testing

There are many ways of testing one's code to validate its proper functionality. The more familiar method is simply coding first, then trying with certain input values if the expected outcome is true. If not, the developer would then attempt to explore how to fix this. This method includes often meticulous manual labour by the developer, sometimes being remarkably inefficient and thus time-consuming. It is also no guarantee to test all required aspects of the code, leaving possible mistakes undetected.

Unit testing is a time-efficient and cost-effective method of validating entire functions and VIs or subVIs to ensure they are functioning as they should. First, one needs to create a unit test file, which receives an extension ".lvtest" in LabVIEW. Creating one is possible for example through the project explorer. For .lvtest files it is possible to define inputs and expected outputs, which are used in testing the VI. The unit test runs the VI and compares the real outputs with the pre-defined expected outputs and generates a report of it. Testing of tens of different VIs can take place in just a minute or few minutes of time. The report generated often automatically flags incorrect values, notifying the user where debugging might be required.

It is possible to run multiple unit tests sequentially by clicking on the "run unit tests" seen on the ribbon of the project explorer view in Figure A-3. This will initiate to run all unit tests in the project. Once completed, LabVIEW will generate a report with the run time, possible failure information and which tests passed (see Figure A-4 for an example). For running individual unit tests, one needs to right click them on the project view and select "run".

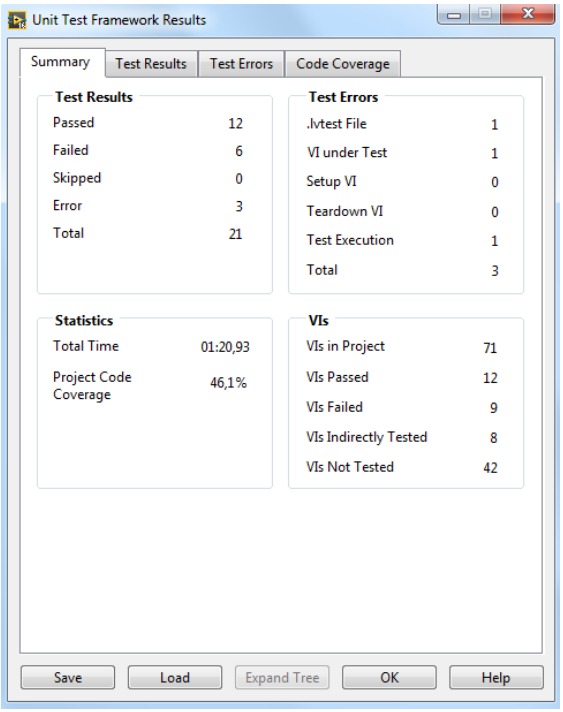

Figure A-4. How unit tests results might look like.

Creating a new unit test is the most time-consuming part of unit testing. For more complex subVIs, one can spend several tens of minutes to create a unit test. However, once created, significant amount of time can be saved if any changes are made and the software is consequently validated. Unit testing is also one of the best ways to prove that the program and its modules work.

Creating unit tests for an individual VI in the project is easy for simple subVIs. However, creating one for VIs which require user action, such as clicking buttons or giving other type of system input, is a bit more time-consuming. For these kinds of VIs, a user-defined unit test is required, where the system inputs, such as click of a button, is achieved programmatically. At least these VIs contain parts requiring system inputs: **Main**, **Second screen**, **CreateParFile** and **Protocol design**. Considering how complex a VI **Main** is, a unit test for it was not created. Creating one for **Main** would have required potentially several tens of hours of labour, which was not deemed necessary during the timeline of this thesis.

One can create either a standard unit test, which runs a VI by giving it inputs and expected outputs and compares the expected outputs to the actual outputs. However, some functions cannot be validated in this way. For instance, **Protocol design.vi** has inputs and outputs, but its outputs are more dependent on what the user does during the call of the

VI, than what was inputted. In other words, this VI requires user to interact with controls and buttons, which can only be tested by a user-defined unit test, where it is possible to create programmatical controls of simulated user interaction. A button for instance can be pressed programmatically to simulate the user clicking “OK” or similar. For more details and a good tutorial by NI on creation of unit tests, see (i) of Chapter A.4.

A.2.2.2. Directory and File Management

The benefit of having a project file in LabVIEW is a better control over LabVIEW’s VI and MATLAB function locations. Setting up auto-populating folders in the project explorer keeps track of changed locations by linking the files, as long as they are contained within the actual project root folder. Additionally, since LabVIEW loads all the project files at each start-up of the program, a much faster loading time is achieved due to LabVIEW project file keeping a link to each file’s location. This way LabVIEW does not need to search the files by their names but can load them directly with the link provided by the project file.

The different VIs and other files related to this project have been placed in various folders in the project folder. Good directory management creates structured and organised view of the files, which in turn can be better in terms of searching for the files. Here, categorisation of the subVIs into their respective folders has been attempted, based on the functionality of the VIs and type of occurrences they are related to.

The root of the project directory contains 10 folders, three project files, a “read me” text file and **Main.vi** and **Second screen.vi**. The folders and their contents are presented in Table A-1. For simplicity, unit testing files (.lvtest extension), folders and files contained under “Help files” folder, as well as .jpg and .png files, are all excluded from the table.

Table A-1. Directory of all folders and containing files in the project directory. The first column indicating “Folder” is the folder contained within the root directory of the project (altogether 10). The second column “Subfolders” indicates a possible folder contained within the “Folder”. The “Files” column indicates which files, either .m or .vi extension, are contained within the subfolder mentioned in the second column. If the subfolder is “(root)”, the files are located at the root of the folder mentioned in “Folder” column

Folder	Subfolders	Files
Analysis and models	(root)	**b_wave_peak.m**
		body_temperature_calculator.m
		Calculate references.vi

Folder	Subfolders	Files
		Feature extraction.vi
		Ignore susi responses from selected references.vi
		Plot of TTM fit.vi
		Retinal temperature averaging.vi
		Retinal temperature estimation.vi
		retinal_temperature_estimation.m
Data acquisition	Probes	**Create piezo task.vi**
		ERG channel set up.vi
		Probes recording outside stimuli.vi
		Probes set up.vi
	(root)	**Change heating method.vi**
		DAQ device error -200220.vi
		Get Terminal Name with Device Prefix.vi
		Heating device control.vi
		Heating set up.vi
		heating_device_control.m
		Laser set up.vi
		Optical power.vi
		Select channel prompt.vi
		Signal reset.vi
		Single ERG-recording.vi
		Single ERG-simulated test.vi
		Unplugged IO ERG-recording test.vi
		volts_to_optical_power.m
Data and file management	(root)	**Add comment.vi**
		Add susi to custom row.vi
		BL Save.vi
		Combine into parameter string.vi
		Create parameter file.vi
		Default data folder (contains the path to root directory of data files)
		Fetch BL estimates from file.vi
		FolderExists_sub.vi
		OpenParFile.vi
		Read data file.vi
		Save continuous data.vi
		Save data to file.vi
		Save image.vi

Folder	Subfolders	Files
		Save references to file.vi
Front panel actions	(root)	**Define new Susi button.vi**
		Enable and disable controllers.vi
Help files	(root)	(contains folders for all help files)
Miscellaneous	(root)	**Adjust number of running responses.vi**
		Updating temperatures to graph.vi
Multicolumn listbox functions	(root)	**Choose rows, list of responses.vi**
		Clear rows, list of responses.vi
		Column index look up.vi
		Fetch responses from file using symbols.vi
Protocol management	(root)	**Change protocol order.vi**
		Delete protocol item.vi
		Open protocol.vi
		Protocol design.vi
		Protocol to queue.vi
		Save protocol.vi
Signal processing	(root)	**Anesthesia monitoring.vi**
		anesthesia_monitoring.m
		AR_baseline_correction.m
		Average signal.vi
		Baseline correction.vi
		Baseline fix and filter.vi
		Filtering MATLAB interface.vi
		filtering.m
		Find if responses are sized different.vi
		Split signals.vi
		Track amplitudes.vi
(root)	(root)	**In vivo heating device and ERG.aliases**
		In vivo heating device and ERG.lvlps
		In vivo heating device and ERG.lvproj
		Main.vi
		Read me
		Second screen.vi

In addition to the directory of LabVIEW files needed to run the software, the directory and file management is found in another location of the hard drive selected by the user.

The other location includes all data files recorded during the recording sessions, as well as configuration files for storing information across different LabVIEW sessions.

A.2.3. Event Structures

In NI LabVIEW, the event case structure is a convenient tool to monitor if a defined event takes place within the VI. It is most efficient for short-timed events cases, as opposed to heavy and long running loops. The event cases are individual event instances inside the event structure which are triggered by defined actions and upon being triggered will execute whatever is inside the event case. Some of these actions could be for instance the user clicking a control, double clicking a front panel item, pressing a keyboard button, etc. The event cases can also be activated programmatically. The event structure needs to be inside a continuously executing loop, such as a while loop, to allow the structure to execute same or different events more than once. One event structure, enclosed by a single while or for loop, can only execute one event case at a time, but can have multiple events in queue.

In this project, event structures have been implemented in various places, such as in these VIs: **CreateParFile.vi**, **Main.vi** and **Second screen.vi**. For many of these, the documentation and their function has been covered well enough in the VI event structures as comments. For **Main** however, the documentation is a bit more extensive due to the VI containing 35 event cases. These are briefly introduced in Table A-2. From development point of view, the aim has been to create this event structure as concise as possible. It contains only events that execute on average within a fraction of a second. Some a bit heavier computation loops might take several seconds but have been mostly avoided in order to improve execution efficiency.

Table A-2. Event case structure in **Main.vi**. The table names all the event cases, what triggers them, which SubVIs are involved and a brief explanation on their function. First column is formatted as: [number of the event case given by LabVIEW] “name of the control that triggers the event”: type of trigger.

Event case	subVIs	Function
[0] Timeout	-	When a numeric value is connected to this event, the entire event structure times out after this amount of inactivity in milliseconds.
[1] “Stimulus parameters.Stimulus type”: Value Change	-	Hides and shows controls related flash or wave stimuli when stimulus type is selected.

Event case	subVIs	Function
[2] “Stimulus parameters.Response (s)”: Value Change	-	When changing value of *Response (s)*, the maximum duration of a flash or PWM wave is coerced to be shorter in duration than *Response (s)*.
[3] “Single”: Value Change	**Enable and disable controllers.vi; Signal reset.vi**	i) Resets amplifier baseline. ii) Disables and greys out various controllers and sets “stimulation on” status to ON. iii) Waits 300 ms to ensure ongoing measurements to take pause. iv) Passes values of front panel controls to “Recording” –queue.
[4] “Multiple”: Value Change	**Enable and disable controllers.vi**	Similar to that of [3] “Single”, but adds set multiple instances to queue, instead of one. Also contains a pseudorandom functionality (if opted in by user), which randomises amount of delay from set range between consequent stimuli.
[5] “Continuous & Piezo”: Value Change	-	When switched on, the trigger button will blink, and various continuous and piezo acquisition controllers are enabled. Opposite is done when the button is switched off.
[6] “Add protocol item”: Value Change	**Protocol design.vi**	Opens the SubVI’s front panel and options for recording setup. Updates the protocol list once SubVI execution finishes.
[7] “Delete protocol items”: Value Change	**Delete protocol.vi**	Deletes protocol items in the list marked with a tick or deletes all protocol items, depending on user selection. Shifts all protocol list items below deleted ones upward in the list and changes their “Nr” to correspond to new order.
[8] “Save protocol”: Value Change	**Save protocol.vi**	Opens a dialogue to save the protocol list as file. Saves all the items on the list for later usable file.
[9] “Open protocol”: Value Change	**Open protocol.vi**	Opens a dialogue for file location. Opens selected file and updates the protocol list to match the file.
[10] “Protocol list”: Double Click	-	Marks selected items on protocol list with symbols or erases the marking on those that have already symbol.
[11] “Protocol list”: Drag Ended	**Change protocol order.vi**	When clicking and holding and dragging an object in the protocol list, the dragged item will be repositioned in the list and reorders items above that to reorder to keep the rest of the list logical

Event case	subVIs	Function
[12] “Start protocol”: Value Change	**Enable and disable controllers.vi**; **Column index look up.vi; Protocol to queue.vi**	i) Sets the protocol list to disabled. ii) Adds protocol list items to “Recording” –queue in chronological order according to “Nr”. iii) Highlights the first row in protocol list. iv) Sets values to auxiliary indicators, such as *Max stretch, Protocol index* and *Stretch nr at protocol start*.
[13] “Stop protocol”: Value Change	**Enable and disable controllers.vi**	i) Enables and disables various controls. ii) Flushes the “Recording” –queue, effectively preventing execution of the rest of the items in protocol list.
[14] “List of responses”: Double Click	-	Adds or clears symbols to items on list of responses.
[15] “Make choices”: Value Change	**Choose rows, list of responses.vi**	The SubVI takes as input the “choose items” string control and adds symbols to these items.
[16] “Clear choices”: Value Change	-	Clears all symbols from list of responses.
[17] “Clear graph”: Value Change	-	Clears all plots from “Running responses graph”.
[18] “Draw”: Value Change	**Fetch responses from file using symbols.vi**; **Baseline fix and filter.vi**	Takes as input selected items (symbols) from list of responses and fetches responses from the experiment files. These are passed to baseline fix and filter.vi, which performs the baseline fix and possible filtering and passes these to the “Running responses graph”, effectively plotting selected items.
[19] “Number of responses shown”: Value Change	-	If user selects less plots to be shown in the “Running responses graph” than currently shown, the event will delete chronologically the oldest plots from the graph.
[20] “Start heating”: Value Change	**Enable and disable controllers.vi**	Enables “Stop heating” and disables the “Start heating” controls.
[21] “Stop heating”: Value Change	**Enable and disable controllers.vi**	Sets the voltage output to zero and enables “Start heating” and disables “Stop heating” controls.

Event case	subVIs	Function
[22] "Average responses": Value Change	**Fetch responses from file using symbols.vi**; **Baseline fix and filter.vi**; **Find if responses are sized different.vi**; **Feature extraction.vi**; **Plot of TTM fit.vi**	Fetches first all selected responses from files and performs baseline fix and possible filtering on these. The modified responses are then compared if they are all the same sized. If yes, they are passed onwards for extracting features, such as b-wave-peak fit. The responses are then passed to *Runnin responses* graph and drawn together with b-wave-fit using **Plot of TTM fit.vi**.
[23] "Add comment": Value Change	**Add comment.vi**	Opens main experiment folder's .par file, adds comment and empties the comment box on **Main** front panel.
[24] "Susi! keyboard": Value Change	**Column index look up.vi**; **Add comment.vi**	If latest response's "Susi" column contains a 0, it is replaced by a 1 on *List of responses* and the main .par file. Else, nothing is done. Hot key set up.
[25] "Susi!": Value Change	**Fetch responses from file using symbols.vi**; **Column index look up.vi**; **Add susi to custom row.vi**	Similar to above, but the operation is instead done for all selected when the "Susi!" was clicked. Additionally, responses are redrawn in *Average signal* and *Running responses* -graphs by excluding the selected responses. Retinal temperature estimates are marked NaN in *Latest retinal temperatures* array for responses that were selected.
[26] "Define new Susi shortcut": Value Change	**Define new Susi button.vi**	Opens a dialog box and awaits for the user to press a new hot key button on the keyboard. Passes the new value to **Main**.

Event case	subVIs	Function
[27] "Confirm references": Value Change	**Ignore susi responses from selected responses.vi**; **Fetch responses from file using symbols.vi**; **Baseline fix and filter.vi**; **Column index look up.vi**; **Calculate references.vi**; **Plot of TTM fit.vi**	Fetches responses from selected ones (ignoring "Susi" ones though). Baseline fixing and filtering is applied and the responses then undergo averaging and b-wave-fit to the averaged response. Retinal temperature is estimated through different characteristics, such as b-wave fit and thereafter TTM. This becomes the reference TTM value. In the meanwhile, also body temperature of selected responses (ignoring "Susi" ones) is averaged. This becomes the reference temperature.
[28] "Reset avg temp 2": Value Change	-	Sets # *Averaged* to zero, effectively resetting the response from which onwards to average retinal temperatures.
[29] "Temperature and heating input.Temperature input": Value Change	-	Changes the temperature input channel by stopping the recording for 500 ms and then changing the channel and enabling the recording thereafter.
[30] "Type of heating": Value Change	**Change heating method**	Changes the *Heating device out* range and caption to correspond to the chosen heating method: either crude or sophisticated.
[31] "Pulse presets", "Preset pulse length", "Light source", "Stimulus parameters", "Pseudorandom info", "Sine wave info", "Square wave info": Value Change	-	If any of the controls is changed in value (pulse pre-set, their length, light source, stimulus parameters or waveform information), a false signalling value is sent to control *Multiple*. This ends execution of multiple stimuli, after which a true signalling value is sent to *Multiple*, adding multiple stimuli to execution with the newly selected values.
[32] "Amplifier trigger reset": Value Change	**Signal reset.vi**	Runs the **Signal reset** subVI, which sends a short digital pulse to the ERG amplifier, setting its offset to zero level.

Event case	subVIs	Function
[33] “Set voltage”: Value Change	-	Sets the output voltage of the heating device to correspond to a value given by user. This event is an added layer of safety to assure the voltage will not be accidentally adjusted too high by the user without confirming the new voltage first.
[34] “Quit”: Value Change	-	Aborts **Second screen.vi**. Additionally, all loops in **Main** should stop running, and the VI proceeds to next frame on the stacked sequence enclosing all functions and loops in **Main**. Config file is updated.

A.2.4. Queue Structure

For controlling events that must be repeated multiple times during one program execution, LabVIEW comes in handy with its intuitive Queue Operations. These operations have been utilised in this project in two ways in the **Main** VI: input and execution of stimuli (Queue name: *Recording*) and passage of recording results to a real time analysis-oriented structure (Queue name: *ERG analysis*).

In LabVIEW, a queue can have a preallocated size, or it can continue to grow indefinitely. The difference is made in memory management. In this project, the queues are not preallocated, but the user can instead input as many items to the *Recording* and consequently *ERG analysis* -queues as wished. In LabVIEW, the queues work on the basis of first in, first out (FIFO) buffering. When removing elements from the queue by dequeuing, the elements are extracted from the queue in the same order in which they were added: chronologically the first one added is the first one to be dequeued. If the number of elements in the queue reaches zero, the loop will pause executing until more elements are in queue.

The first and most important queue, *Recording*, is highly convenient and can be seen executed in various events. In previous software versions, the event structure (see Table A-2) would have contained the actions now put under *Recording* queue. The actions in *Recording* queue are slowly executing data acquisition functionalities, which would clog the event structure, hampering the execution of features that need to execute quickly. Thus, the actions under *Recording* queue are now set separate from the event structure, allowing more efficient execution in overall. For this reason, events like “Single”, “Multiple” and “Start protocol” all act to enqueue elements into *Recording* queue. The elements enqueued dictate the order, parameters and timing of stimuli and recording. The

Recording queue is dequeued just below the event structure, where the dequeued elements are passed on to a subVI: **Single ERG-recording**, **Single ERG-simulated test** or **Unplugged IO ERG-recording test**. The subVI, dictated by user selection during start-up, initiates the stimulus and performs the data acquisition.

Another queue structure, designated as *ERG analysis*, is found on **Main**. This queue exists to not clog the time-sensitive *Recording* queue's stimuli and recording execution. Hence, once *Recording* queue has dequeued an element and its execution is complete, the acquired data is enqueued to *ERG analysis*. The elements in the queue are executed in a while loop structure by dequeuing and analysing the results of the ERG recording. The queue structure here also contains methods for saving data to file, performing the retinal temperature estimation and updating relevant graphs.

Having these two independently dequeued queue structures allows to squeeze more performance out of LabVIEW. The goal is to prevent slowing down the event structure (see Table A-2) or the ERG recording. Having the structures implemented inside one single loop would result in the opposite. Instead, two queues and dequeuing loops are implemented to allow the computer's processor to run them independently and simultaneously. Especially the *ERG analysis* queue is convenient for ensuring real time analysis of results, which could otherwise significantly slow down the recording set up, if it had to be done in the same loop structure.

A.2.4.1. Editing the Queue Structure at Any Parts of the VI

Changing the queue structure is possible by keeping in mind a few matters. Editing the queue structures could be necessary if more elements are wanted to be passed to the queue for stimulation set-up or analysis for instance.

When passing new or changed inputs to the queue by enqueueing, one needs to change the type definitions **Recording queue element.ctl** and **ERG analysis queue element.ctl**. These type definitions are specified in the block diagram at **Obtain Queue** for each of the queues but are also found in the project directory under "Custom objects and global variables". One needs to simply add more controls to either of these type definitions (which are clusters of controls) and give a label for each of the controls. This way, the clusters can be unbundled or bundled by name when dequeuing and queuing elements. When changing the type definition control, it will be automatically updated at its all instances: in event structures of **Main**, as well as, in various subVIs. LabVIEW will automatically give an error also if any inconsistencies are detected.

A.2.5. Second Screen

The **second screen.vi** is a VI that opens up in a second monitor of the computer when running **Main**, if there is a second monitor in use. It runs in parallel with **Main** and is thus not referred as subVI. If a second monitor does not exist, it opens cascaded on the main monitor. **Second screen** contains useful graphs, such as an averaged signal graph, baseline fix related graphs, real time response information and amplitude in function of time. **Second screen** also contains *Fundus camera feed* and controls for displaying options on the mentioned graphs and camera feed.

The fundus camera (Allied Vision Mako U-503B) is connected by USB 3.0 connection to the PC. It requires IMAQ package of LabVIEW to be installed. Some tuning options were included in the implementation of the *Fundus camera feed*. These include the adjustment of exposure time and gain. Additionally, it is possible to acquire snapshots with the *Record* button, which are saved to a file and are named systematically after the experiment name. During stimulation and ERG recording, the *Acquire* control is set to receive an input by a reference from other functions. When the *Acquire* receives a true value, the camera acquires an image.

A.2.6. SubVI Documentation

Table A-3 provides a brief overview of each subVI's main function, as well as, the subVI's input and output parameters. A more thorough documentation of all the subVIs can be found in the help files of each one. Additionally, block diagram of each of the subVI contains explanatory comments.

Table A-3. All the subVIs used in the software project, their function, as well as, input and output parameters through the connector pane of the subVI

VI name	Main function	Input parameters	Output parameters
Add comment	Opens data file and adds user-typed comment string to it.	*ColHdrString, Experiment name, Comment box, ItemNames in, error in*	*ItemNames out, error out*
Add susi to custom row	Marks a recorded response as bad.	*Experiment name, Response index, error in*	*error out*
Adjust number of running responses	Changes how many responses are shown in *Running responses* graph.	*Running responses in, Number of responses shown, ERG response*	*Running responses out*

VI name	Main function	Input parameters	Output parameters
Anesthesia monitoring	Interfaces for a MATLAB function computing respiratory rate from piezo signal.	*data, Fs, low_limit, high_limit, min_peak_distance*	*error, frequency, warning*
Average_signal	Takes in defined ERG responses and outputs a modified average signal based on processing options selected by the user.	*Number of averagings for plot, Stretch nr, Include Susi, Filtering, New baselines, Baseline fix parameters, Item names, Experiment name, Column headers, Display TTM*	*Averaged signal, Sampling rate, Averages, Minimum, Average, TTM*
Baseline correction	Performs selected baseline correction and outputs adjusted ERG signal.	*Baseline, Sampling rate, Response, Baseline fix parameters, Update graphs*	*BL prediction, Just response, BL + corrected response*
Baseline fix and filter	Performs both baseline correction and filtering on the signal if opted in by the user. Outputs modified ERG signal.	*New baselines, Update graphs, Experiment name, Filtering, Responses to draw, Response list item names, Response list column headers, Response indexes, Baseline fix parameters*	*BL array, Aq_frequency array, BL prediction, Just response, BL + corrected response*
BL Save	Saves baseline correction related data to a file.	*Experiment name, Stretch number, Voltage output, Predicted BL, Baseline fix parameters*	-
Calculate references	Computes the reference temperature and t_p values for user-selected responses. User can select the method of computation. Responses marked as "Susi", i.e. "bad", are excluded.	*B wave parameters, Responses item names, Response indexes, Acquisition frequencies, Index for temperature, BL + corrected response, Array of baselines, Temperature model, Susi responses*	*ttm, T_ref, Amplitude, polyparams, breakerror, Text to display*

VI name	Main function	Input parameters	Output parameters
Change heating method	Changes the range for heating device input and its caption according to selected input method.	*NewVal*	*Caption.Text, Scale.Maximum, Scale.Minimum, Value*
Change protocol order	Adjusts the order of protocol items in the list.	*Obtained item names*	*Updated item names*
Choose rows, list of responses	Marks a tick on those responses in *List of responses* which user types in a text box: e.g. “1-5” or “1, 2, 3”.	*Choose items, error in, ItemSyms in*	*ItemSyms out, error out*
Clear rows, list of responses	Clears all selections/ticks in *List of responses*.	*ItemSyms in, error in*	*ItemSyms out, error out*
Column index look up	Returns the column index that contains defined header name.	*Multicolumn list headers, Search for*	*Column index*
Combine into parameter string	Takes in variables and combines them into a string that can be saved to a data file.	*List of responses item names, cluster, Retinal temp., Amplitude, TTM*	*New item names, Parameter string*
Create parameter file	Opens when **Main** is started and prompts for user input.	-	*File name, Test options output*
Create piezo task	Creates DAQ task for piezo sensor.	*error in, Sampling rate, Piezo channel, ERG Ch*	*task out, error out*
DAQ device error -200220	Handles the data acquisition related error code 200220.	*error in*	*error out*
Define new Susi button	Lets user select new hot key for “Susi“ button.	-	*New button*
Delete protocol item	Deletes selected protocol items from list.	*ItemSyms in, ItemNames in*	*ItemSyms out, ItemNames out*

VI name	Main function	Input parameters	Output parameters
Enable and disable controllers	Either enables or disables controllers given in as an array of references.	*error in, Array of items, Disabled status*	*error out*
ERG channel set up	Creates a DAQ task for ERG channels and heating device photodiode.	*Sampling rate, ERG channels, Min (V), Max (V), heat input channels*	*Task out, error out*
Feature extraction	Interfaces for MATLAB function, which computes b-wave t_p and amplitude for selected signal.	*Light source, response, B-wave fit parameters, aq_frequency, bl_duration*	*ttm, amplitude, breakerror, polyparams*
Fetch BL estimates from file	Acquires baseline estimates of selected ERG response from file.	*Experiment name, Response indexes*	*BL estimates, error out*
Fetch responses from file using symbols	Acquires selected ERG responses from the data file.	*Selected Items, Experiment name, Ch2 in use?*	*Responses, error out, Responses indexes*
Filtering MATLAB interface	Interface for MATLAB filter function.	*BL + corrected response, Sampling rate, Filter type*	*Output signal*
Find if responses are sized different	Compares array of ERG responses and returns true if their sizes do not match. Otherwise returns false.	*Responses, Response indexes*	*Different sized?*
FolderExists_sub	Displays an error message if user-selected folder exists when prompted by **Create parameter file**.	*Folder name*	-

VI name	Main function	Input parameters	Output parameters
Get Terminal Name with Device Prefix	Acquires the prefix for a device defined by *Local Terminal Name*. LabVIEW's internal VI.	*Task in, Local Terminal Name, error in*	*Task out, Terminal Name with Device Prefix, error out*
Heating device control	Interfaces for MATLAB function, which takes a setpoint temperature or voltage as input and adjusts it into an actual voltage output.	*Type of heating, Last heating (V), Optic power (mW), Heating device out, Retinal thermometer*	*Actual heating device V*
Heating set up	Creates DAQ task for the heating device.	*Channel*	*Refnum out, error out*
Ignore susi responses from selected references	Modifies indexes of selected responses to exclude responses marked as "Susi".	*Selected responses, Responses item names, Responses column hdrs*	*Modified responses, Indexes for susi responses*
Laser set up	Sets parameters for stimulus light task.	*phase, Duration, Time/Frequency, Stimulus type, Channel, Stimulus (s), Baseline (s), Sampling rate*	*Stimulus waveform, Waveform array, Task output, error out*
Open protocol	Loads a saved protocol list from file.	-	*Obtained item names, Column headers*
OpenParFile	Acquires and outputs details from the defined parameter file.	*Experiment name, Stretch number*	*Wavelength, Temperature, Stim. freq., Stimulus parameters, all rows*
Optical power	Interfaces for MATLAB function computing optical power for photodiode voltage.	*Heating device voltage, pd calibration polynomial*	*Optic power*
Plot of TTM fit	Plots the curve which was done as a fit to find t_p of a response.	*Average response, Baseline, Aq frequency, ttm, Poly width, polyparams*	*Running responses*

VI name	Main function	Input parameters	Output parameters
Probes recording outside stimuli	Records data from thermistor and heating device photodiode.	*Temperature and heating input*	*data, error out*
Probes set up	Creates DAQ task for thermistor and heating device photodiode.	*Temperature and heating input*	*Task out, error out*
Protocol design	Prompts user to add items to protocol list.	*Obtained array, Column headers*	*Updated array*
Protocol to queue	Adds protocol items to *ERG recording* queue.	*Repeat protocol, Temperature and heating input, Protocol list, Recording queue, Recording element, Stimulus parameters, Stimulus type StringsAndValues, Stimulus type Strings, ColHdrs, ERG channels*	*Protocol help array*
Read data file	Loads responses from data file and appends them array of responses.	*Data file path, Responses in*	*Appended responses out, error out*
Retinal temperature averaging	Computes running retinal temperature average of defined number of latest responses.	*Latest retinal temperatures, Averaged over # items, Temperature tolerance*	*Averaged temperature estimate*
Retinal temperature estimation	Interfaces for MATLAB function performing the retinal temperature estimation.	*T_ref, ttm, ttm_ref*	*T*
Save continuous data	Saves continuously acquired data to file.	*Parameter string, Experiment name, ERG Ch1*	*error out*

VI name	Main function	Input parameters	Output parameters
Save data to file	Saves Ch1 and possible Ch2 data to files and saves string to parameter files.	*Experiment name, Parameter string, Ch1 ERG, ERG Ch2, Ch2 in use?*	-
Save image	Saves acquired snapshots to files.	*Fundus camera feed, Experiment folder, Last response nr*	-
Save protocol	Saves protocol list to a file.	*Column headers, Obtained item names*	-
Save references to file	Saves reference values to a file.	*T_ref, ttm_ref, Indexes for susi responses, Exp. name*	-
Second screen	VI running in parallel with **Main** on PC's second monitor. Technically not a subVI.	-	-
Select channel prompt	Prompts user to select a new channel for a DAQ input.	*Enable, error in*	*DAQmx Physical Channel 2, error out, OK*
Signal reset	Readjusts ERG signal amplifier's baseline to zero.	*Signal reset, ERG trigger reset channel*	-
Single ERG-recording	Performs a single light stimulus and ERG recording.	*ERG channels, Temperature and heat input channels, Stimulus parameters, Stimulus input parameters, Combined input signals, Light source*	*ERG Ch1, ERG Ch2, Temperature, Heating device voltage*
Single ERG-simulated test	Performs a single light stimulus and simulated ERG recording by using ERG data from a user-selected file.	*ERG channels, Temperature and heat input channels, Stimulus input parameters, I/O features enabled, Stretch nr, Experiment name*	*Wavelength, Stim. freq., Temperature, ERG Ch1, ERG Ch2, Heating device voltage, Stimulus parameters*
Split input signals	Splits combined signal input into distinct ones.	*Combined input signals*	*ERG signal, Temperature, Heating device voltage*

VI name	Main function	Input parameters	Output parameters
Track amplitudes	Draws an amplitude graph in function of time.	*Amplitude, seconds since 1Jan1904*	-
Unplugged IO ERG-recording test	Only acquires random-generated data within certain range. Ignores I/O devices.	*Stimulus input parameters, Stimulus parameters, ERG channels, Temperature and heat input channels, Light source*	*ERG Ch1, ERG Ch2, Temperature, Heating device voltage*
Updating temperatures to graph	Updates the retinal temperature and body temperature into the temperatures graph.	*Time, Body temp. Y, Retinal temp. Y, history, clear, Temperature graph in*	*Temperature graph out, X scale maximum, X scale minimum*

A.2.7. MATLAB Functions

A major feature of this software has been an integration of various MATLAB functions by interfacing with LabVIEW and its VIs. Many of these functions could be managed by applying LabVIEW's functions, but considering the efficiency, easy-to-use and prevalence in the field of engineering, MATLAB is a fitting solution. Additionally, some features are actually more efficient on MATLAB than LabVIEW, such as the autoregressive modelling. The data analysis by the research group is also performed on MATLAB after each experiment session. If the user wishes to change the values or the script of any of the MATLAB functions, it can be done directly by replacing the existing .m file in the LabVIEW directory. As long as the input or output values or the name of the function does not change, no other actions are required on LabVIEW's end.

When **Main** is run, its start-up routine also contains a functionality that opens the MATLAB command console and adds LabVIEW directories to MATLAB path. Unit testing any MATLAB related functions thus requires running **Main** at least once before unit testing. Else, an error is produced on the unit tests requiring MATLAB functionality.

All the MATLAB functions, their main functionality, as well as, input and output parameters are listed in Table A-4. More documentation is available directly in the functions as comments. Additionally, Chapter A.2.7.1, explains the anaesthesia monitoring, filtering and baseline correction a bit more in detail.

Table A-4. All the MATLAB functions used in the software project. Contains the function name, its main operation, as well as, input and output parameters

Function name	Main operation	Input parameters	Output parameters
anesthesia_monitoring.m	Computes the respiratory rate from the acquired piezo signal.	*Fs, data, low_limit, high_limit, min_peak_distance*	*error, frequency, warning*
AR_baseline_correction.m	Computes autoregressive (AR) model for baseline and subtracts it from the raw response.	*response, flash_index, model_order*	*newresponse, ar_baseline, error*
body_temperature_calculator.m	Computes the body temperature from thermistor voltage.	*raw_voltages*	*body_temperature*
b_wave_peak.m	Fits a polynomial to find a b-wave peak.	*response, bl_duration, aq_frequency, repetitions, poly_width, light*	*ttm, amplitude, polyparams, breakerror*
filtering.m	Takes in signal and performs defined filtering on it.	*Fs, filter_type, input_signal*	*output_signal*
heating_device_control.m	Takes a setpoint temperature or voltage as input and adjusts it into an actual voltage output.	*heating_device_out, type_of_heating, last_heating, voltage_read, temperature*	*error, voltage_out*
retinal_temperature_estimation.m	Computes the estimated retinal temperature based on reference values and t_p.	*T_ref, ttm, ttm_ref*	*T*
volts_to_optical_power.m	Uses a calibration file to convert recorded photodiode voltage into optic power.	*heating_device_voltage, pd_calibration_polynomial*	*optic_power*

A.2.7.1. Anaesthesia Monitoring

Anaesthesia monitoring was introduced as part of tracking the mouse respiratory rate during anaesthesia and thus to estimate the depth of anaesthesia. The implementation is in **Main**, where a circular buffer was implemented to include respiratory data from the

piezo sensor to be passed on to the **anesthesia_monitoring.m** MATLAB function. Additionally, each recorded data point is saved to a data file for later use.

The function **anesthesia_monitoring.m** gets as an input the data recorded at the piezo sensor for the past 10 seconds, as well as the range of acceptable respiratory rate limits and other technical settings. The function then calculates the FFT for the input signal, in order to attain the frequency spectrum. Since the breathing frequency of a mouse is somewhere in the range of few hertz, it is relatively easy a deduction to assume there should be a frequency component peak in the low end of the frequency spectrum produced, typically around 1-4 Hz in the experiments. Automatically finding this peak can be achieved by MATLAB's peak finding function, setting the distance between consecutive peaks large enough and looking for the peak that is in the lower end of the spectrum. This way MATLAB will output a hertz value corresponding to the respiratory rate.

The computed respiratory rate is compared against user-defined limits for too shallow and deep anaesthesia frequencies. If the breathing frequency does not satisfy either of these limits, MATLAB will output a variable indicating which condition was not met. The LabVIEW interface will then light up the LED on the front panel to indicate that the anaesthesia agent delivery speed needs to be modified.

A.2.7.2. Filtering and Baseline Correction

The user can opt in for filtering of the ERG signal during an experiment session. This was brought in as a MATLAB function, **filtering.m**, which contains currently only one filter option: 50 Hz Notch filter. 50 Hz is the mains frequency in Europe and majority of the world, which can sometimes present itself as noise in recorded signals if proper precautions are not taken. The implemented filter, infinite impulse response Notch filter, is a stop-band filter. When selecting 50 Hz as the stop-band, the filter attenuates the 50 Hz component to remarkably low levels, effectively removing the frequency component. The MATLAB function can be modified to include more filters if a need is identified. User can also choose not to perform filtering at all.

Baseline correction is often necessary, as the recorded ERG signal can drift to non-zero baseline. LabVIEW function **Baseline correction** contains the means to adjust the signal baseline. The options include simply subtracting the average value of the signal baseline, i.e. pre-stimulus time, from the signal itself. User can adjust the length of baseline used for taking an average value. As a result of this method, the signal is moved approximately to the zero of the vertical axis. The second option includes autoregressive (AR) baseline

adjustment performed through the MATLAB function **AR_baseline_correction.m**. Its more in-depth functionality is explained in the function itself. Briefly, user can adjust the method by adjusting the *model_order* value, which is associated with the number of data points used as prediction of the signal. An interested reader is invited to learn more about the function by reading on the used Burg's method AR modelling: see (ii).

A.2.8. Type Definitions and Block Diagram Readability

LabVIEW contains functionalities for making controls or indicators type definitions. Type definition enabled variables are saved as separate files in the project directory, where they can be obtained for use in any part of the program. Instead of duplicating a variable from subVI into another subVI and passing the values through connector pane, creating a type definition ensures that the variable keeps same structure across all its subVI instances.

As LabVIEW is a graphical programming platform, all the source code is viewed as a block diagram containing blocks connected by wires. The blocks are functions or modules and the wires show data flows. At some point, when the software becomes large enough, it can become difficult to follow what is going on, mostly due to the apparent complexity caused by overlapping of multiple wires. The complexity of overlapping writes was considered in making the code easier comprehendible for future developers. Instead of passing, for instance, different types of data across functions, which would create more wires, clusters were used. Clusters contain multiple variables packed into one structure, which can be passed as one single wire, and unpacked to its contents when desired. In addition to clusters, creating subVIs was given emphasis. Some parts of the code can become so large, that they occupy vast amount of space from the graphical interface. Compressing this into a subVI can be useful in order to display all of it as one single block. Creating a subVI out of blocks of code should be done carefully though, as there are risks to performance (see Chapter A.3.2 on execution efficiency).

A.3. Recommendations for Future Software Development

I would like to discuss here how the development should optimally take place in the future when adding more modular pieces to the project. These are anywhere between smaller memory management aspects to bigger structural considerations. In principle, it would be essential that any new development would fit the original block diagram as well as possible, especially in case of **Main** which already has a large block diagram.

A.3.1. Memory Management Considerations

The memory management starts from very small. Usually memory issues are easy to avoid in a sophisticated platform like LabVIEW. These can be generally divided into two kinds of categories: firstly, the small pieces that gradually make the project bulkier slowing down performance, and secondly, the bigger memory leaks.

When LabVIEW loads a VI, it generally loads all the contents of it to memory (RAM, in specific), no matter if particular functions, subVIs or blocks are used during a specific run. For this reason, it is good to consider how often certain parts of the software need to run and at which points in order to consider loading and releasing them dynamically. This chapter consists of smaller subunits, which all have an effect on the RAM usage of the software project.

A.3.1.1. Number Representation

The very first, yet relatively insignificant, memory optimisation method is changing the representation of various numbers to correspond to the memory needs and actual use of the number. In computing, numbers are composed of bits. One byte is eight bits, and usually in LabVIEW numbers are in the accuracy of a multiple of bytes, such as: U8, U16, U32, U64, which all are unsigned integers. The number suffix after the letter "U" represents bits used for expressing the number. For example, an unsigned integer of 8-bit accuracy (U8) means that this integer can be a number anywhere between the range of 0 and $2^8 = 256$. In this case, the number consumes 8 bits or 1 byte of RAM during LabVIEW execution.

When designing new implementations for the software project, it is good to consider the range of values the number can receive. For instance, a dropdown menu with three options numbered 0, 1 and 2 (known as enumerator in LabVIEW) can easily be assigned an accuracy of U8 representation, instead of using for instance U64, which would take 7 bytes more memory. In the case of a single variable, a consumption of 7 extra bytes of memory is negligible in comparison to gigabytes of RAM usually present in computers. The effect is more notable and amplified when multiple variables are assigned unnecessarily the higher bit representation. It is especially important to select the representation appropriately when it is used in multicolumn listboxes, arrays or in structures that amplify its memory usage effect. For instance, *ERG recording* queue can amplify the memory usage of a number allocation by 1000-fold when executing 1000 repetitions of a recording.

A.3.1.2. Display of Graphs and Controls

In general, each graph, control and indicator displayed on the front panel that needs to be updated can slow down performance. This is why it is good to consider which variables need to be visible on the front panel. Those which are not needed can be hidden. Additionally, graphs have a setting "smooth updates" on by default. This can put more strain on the processor and can slow down performance as well. Turning smooth updates off will result in a more noticeable flicker, as the contents of the graph are erased and redrawn after each iteration, but this will also improve performance. It would be recommended to always consider which graphs need to have smooth updates and turn the feature off if needed by accessing the graph's settings.

A.3.1.3. Close References

Whenever pointing a reference to a variable or even a separate subVI, LabVIEW reserves RAM for this dynamically. If the reference remains open, it might lead into a memory leakage. As this can be easily overlooked, it is recommendable to be extra cautious of closing all references as soon as they are not needed anymore. Separate **close reference** function exists in LabVIEW for this. Leaking memory can result in shutting down of the entire software.

A.3.1.4. Dynamically Loading and Calling VIs

It is important to note that when subVIs are part of the block diagram of any VI, they are all loaded up into the physical memory when starting up the VI. However, some subVIs need to be called constantly, some a bit less often and some subVIs might be called only once, not at all or only conditionally. For this reason, it is good to consider if the subVI needs to be inserted into the block diagram statically, or if it could be called dynamically, to facilitate lower memory consumption. See an example view of a dynamically called subVI in Figure A-5.

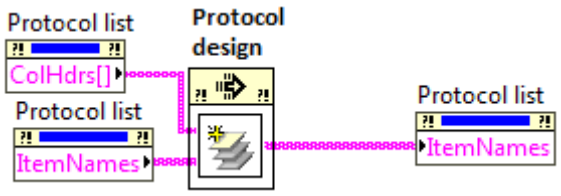

Figure A-5. Dynamically loaded and called **Protocol design** as an example of dynamically called subVI. The box enclosing **Protocol design** is distinctive for dynamic calling.

The options for dynamically calling subVIs include calling them by “Load with callers”, “Reload for each call” or “Load and retain on first call”. The latter two are mostly utilised in this project. Loading a VI into memory when calling it, whether it be the first time or not, can cause small delay due to the loading taking a few milliseconds. Hence, the use of these options should be considered for time-sensitive operations. A complete list of dynamically called VIs in **Main** is presented in Table A-5.

Table A-5. Dynamically called subVIs in **Main**.

VI name	Location on **Main**	Method
Change protocol order	Frame 2: event [6] “Add protocol”	Reload for each call
Create parameter file	Frame 0	Reload for each call
Define new Susi button	Frame 2: event [26] “Define new Susi button”	Reload for each call
Delete protocol item	Frame 2: event [7] “Delete protocol item”	Reload for each call
Open protocol	Frame 2: event [9] “Open protocol”	Reload for each call
Protocol design	Frame 2: event [6] “Add protocol item”	Reload for each call
Save protocol	Frame 2: event [8] “Save protocol”	Reload for each call
Single ERG-simulated test	Frame 2: dequeuing loop for “Recording”	Load and retain on first call
Unplugged IO ERG-recording test	Frame 2: dequeuing loop for “Recording”	Load and retain on first call

For example, **Create parameter** file is ran with the method “reload for each call” when running **Main** for the first time. It is loaded once and released immediately after its execution finishes subVI. In case the subVI was to be called again, it would have to be reloaded to memory again each time. Similar has been done for various other subVIs as well, which are expected to run only once or few times and are not time-sensitive.

Dynamically calling with the method of “Load and retain on first call” is used in this software project for subVIs that might not need to be run at all. If they do need to execute, and it is expected that they need to run multiple times or their execution is time-sensitive on subsequent runs, it is useful to have it loaded permanently to memory on the first run. For instance, the various ERG recording testing functions, such as **Single ERG-simulated test** occupies needlessly RAM if the user opts out from this test method at start-up. Hence, it is dynamically loaded on the first time it is called and it will be retained in the memory until **Main** is aborted.

All the calling methods can be easily set-up by navigating to Application Control in the Programming palette and selecting **Call By Reference**. By right clicking and selecting "Call Setup…" it is possibly to easily search for the desired VI and select the option for how to call for it.

A.3.2. Execution efficiency

VIs can be executed in a variety of fashions. The execution method of VIs is by default "non-reentrant execution" with "normal priority". There is some amount of overhead associated when calling subVIs in this fashion and perhaps the default execution method is not always the most preferred when it comes to memory usage and available alternative methods. Calling a subVI with its default execution settings, a few milliseconds overhead occurs. This can be especially cumbersome if the subVI is called several times within a loop or if there is a need for time-sensitive actions with one or more subVIs involved. In both cases, significant effects to execution speed might occur. See (iii) in Chapter A.4 for further reading.

Some of the easier adjustments for reducing subVI overhead is to consider the placement of the subVI. Is there for instance a way to place it outside a loop it might be part of? Another consideration would be to turn the subVI into a subroutine priority or inlining subVIs into their calling VIs. Subroutines and inlining subVIs were used in this project implementation and are covered in the following chapters (A.3.2.1 and A.3.2.2). In order to access the execution settings of a VI, select File > VI Properties > Execution.

A.3.2.1. Subroutine subVI Execution

The execution priority affects the execution speed. Considering the use of subroutine is appropriate for VIs which are frequently executed, not requiring user interaction and are short in amount of source code. Subroutines run faster due to them executing immediately once their input values are available. The trade-offs of using a subroutine is lack of visible front panel data. There is also lack of dialog, call timing, debugging and automatic error-handling. See (iii) in Chapter A.4 for further reading.

It is possible to also adjust the execution properties to allow a more accurate execution of the subVI. For instance, in **Single ERG-recording**, the execution priority has been switched to "time critical priority (highest)" and the preferred execution system to "data acquisition". These properties ensure the best performance of the subVI and its execution.

A.3.2.2. Inlining subVI to calling VI

Inlining a subVI into calling VIs means that the source code in the subVI acts as if it was pasted on the calling VI's block diagram. This is due to the different way the code is compiled. Normal execution of a subVI causes a few milliseconds of delay, but this delay is greatly diminished when the subVI code is inserted on the calling VI's compiled code either through removal of the subVI and pasting the code to the calling VI or inlining the subVI.

A.3.3. Development and Structure

When developing new features to the software, it is recommended to look into the maintainability aspect of the software. Whenever creating new subVIs to be implemented in this software, it would be useful to consider whether an existing VI can already cater part of the needs, making it more feasible to utilise existing VIs in the creation of new features. Additionally, type definitions, introduced in Chapter A.2.8, are important for maintainability of the software.

The structure of the software should follow most preferably the current one. The number of while and for loops, as well as event and queue structures are at the current moment quite numerous enough for **Main** and **Second screen**, which essentially are running continuously. It is possible to implement many new functionalities in consideration of the existing structures, and avoid creating new processor-splitting loops, queues or other structures. For instance, all data analysis can be contained within the loop dequeuing *ERG analysis*. Also, all front panel related events click events or other quickly executable actions should be placed within the event structure of **Main**. It is more efficient to have all the fast executable events in one event structure, rather than scattered over different ones.

A.3.4. Unit Testing

As was covered earlier, unit testing is a great way to validate the functionality of each of the involved functions or units of the software. For this reason, it would be recommended to continue creating unit tests for new (sub)VIs. Additionally, **Main** does not have any unit testing, due to its wide range of functionalities. Creating a unit test for **Main** would be highly beneficial but creating a thorough test would most likely require several tens hours of work. It is possible to create new unit tests from the project explorer.

For the benefit of unit testing, many MATLAB functions were interfaced under separate subVIs. This way, the MATLAB functions can also be tested with the LabVIEW unit

testing, instead of running another other unit testing program. All the functionalities get tested in one place, the LabVIEW project explorer.

Currently the project code coverage of unit testing is 46.3 %. Of the 75 custom subVIs created for this software, 33 have an associated unit test. For the time constrains of this thesis, the coverage was not able to be increased above these numbers due to extensive test protocol needed for **Main** that contains most of the project code. Although as many as 42 subVIs were not tested, the importance of them is miniscule. Most of them are I/O functions or file management functions, which interface with many functionalities of **Main**, making their unit testing more appropriate in parallel with testing of **Main**.

A.3.5. Documentation

Most of the important documentation is written on the LabVIEW structures themselves. Analogically to text-based coding, it is good to add comments in the code. In LabVIEW this is achieved as adding text boxes on the block diagram and placing them next to the parts that should require explaining. This is the most important way to do proper documentation in LabVIEW.

Another important aspect is to write description for the subVIs used by accessing the VI properties (File > VI Properties > Documentation). The description written in the text box appears when hovering over a subVI in the block diagram while Context Help is enabled. It is advisable to do this for all subVIs and include a brief description of each of the inputs or outputs of the VI. Additionally, it is possible to create a help file and link it to the VI in the same menu.

Main has been configured to prompt for a comment every time it is saved. This is added as an entry and can be accessed any time later from File > VI Properties > Revision. This allows a relatively fast and easy way of recording small changes that have been done between revisions of **Main** or its subVIs. Debugging is also easier if comments are straight written to revisions.

It would be recommended, in addition to everything mentioned above, to create documentation files separately for end-users and developers. For developer it is easier to receive an overall picture of the program when explanations of, for instance, data dependencies between different structures are explained. For an end-user it is more important to understand how the front panel functions.

A.3.6. Ideas

LabVIEW supports object-oriented programming and contains easily comprehendible instructions for doing this. It might be worth looking into whether there are any structures that would be essential to carry out with objects. The stimulations and ERG recording might be some of the features which could be implemented with object controls. For instance, *Stimulus type* could be the parent class, which could be passed to the **Single ERG-recording**. Or perhaps the subVI itself could consist of classes and methods, instead of **Main**. Object-oriented programming entails many benefits, such as better maintainability of large programs, easier understanding of their function once created, as well as better reuse of the objects in other programs. See (iii) in Chapter A.4 for further reading.

Currently the depth of anaesthesia is being estimated with FFT of the acquired signal. However, this is most likely not the optimal way of computing the respiratory rate. As can be seen already in current build, the estimated rate is sometimes off by a few hertz if the signal is very noisy. Other more accurate, but computationally heavier, methods could be investigated to combat the occasional issues. For instance, autoregression could be more accurate but also computationally heavier.

A.4. Further Reading

(i) Prove It Works: Using the Unit Test Framework for Software Testing and Validation. [Internet]. 2017 [cited 2018 Jan 29]. Available from: http://www.ni.com/white-paper/8082/en/

(ii) MathWorks Documentation: arburg [Internet]. [Cited 2017 Dec 17]. Available from: https://se.mathworks.com/help/signal/ref/arburg.html

(iii) VI Execution Speed [Internet]. 2011 [cited 2017 Aug 21]. Available from: http://zone.ni.com/reference/en-XX/help/371361H-01/lvconcepts/vi_execution_speed/

(iv) CS 164: Fall 2015 – Introduction to Computer Science Object Oriented Programming: Advantages of OOP [Internet]. 2015 [cited 2017 Nov 16]. Available from: https://www.cs.drexel.edu/~introcs/Fa15/notes/06.1_OOP/Advantages.html?CurrentSlide=3

(v) LabVIEW Data Acquisition Basics Manual [Internet]. 1997 [cited 2018 Jan 30]. Available from: http://www.ni.com/pdf/manuals/320997b.pdf

(vi) LabVIEW User Manual [Internet]. 2003 [cited 2018 Jan 30]. Available from: http://www.ni.com/pdf/manuals/320999e.pdf

(vii) National Instruments Technical Support [Internet]. Available from: http://www.zone.ni.com

(vii) Getting Started with LabVIEW [Internet]. 2013 [cited 2018 Jan 30]. Available from: http://www.ni.com/pdf/manuals/373427j.pdf