The University of Nottingham

UNITED KINGDOM · CHINA · MALAYSIA

DEPARTMENT OF ELECTRICAL AND
ELECTRONIC ENGINEERING

Robotic Control using Eye-Gaze tracking

AUTHOR	Mr Miles Bardon
SUPERVISOR	Dr Steve Greedy
MODERATOR	Dr Stephen Bull
DATE	17/05/2018

Third year project report is submitted in part fulfilment of the requirements of the degree of Bachelor of Engineering.

Abstract

Through the course of this project an application has been developed that can control robotic devices such as a car or manipulator by a user's eye movements. It can be scaled up to machines that can aid the user in day-to-day life. This presents a platform for further innovation in the realms of capturing more gestures, and opens up new technology and its benefits to those that could not previously utilise them.

The application is designed to have an intuitive user interface such that a typical user can install and use it without reference to a manual, and require only a simple hardware setup. The software will be able to be controlled by the user's eyes, via a commercially available eye tracker with a free developer SDK.

This report will discuss the theory behind eye tracking and follow the development of the application, including how to apply the SDK in practice. Testing and verification procedures will also be detailed, and finally the commercial prospects of this solution will be explored.

Contents

1 Introduction

There are many people around the world with conditions, such as Rett Syndrome which affects 1 in 10-22,000 females [1], that reduce or even eliminate some motor function. Other conditions, such as Amyotrophic Lateral Sclerosis (ALS) causes the death of neurons controlling voluntary muscles [2], rendering those with it unable to move their limbs normally. This is just one of many so called “Motor Neurone Diseases”, and the late Stephen Hawking is one of the most famous examples of a person to have lived with it.
For these people, and their families, it can be difficult to aid them in basic day-to-day tasks that involve grasping or moving objects, as well as simple communication.
Within these families, and indeed their communities, communication is key to sharing personalities, making thoughts known and letting people know what they want. Through the means of Augmentative and Alternative Communication (AAC) [3], non-verbal communication can be interpreted by carers and family. The options for AAC can range from No Tech to High Tech, as demonstrated in Table 1.1.

No Tech	**Low Tech**	**High Tech**
Pointing	Partner Assisted Scanning	Switches used to access a robust communication system.
Gesture	Alphabet Boards	iPads/Tablets
Body Language	Communication Boards	*Eye Gaze devices*
Signing	Aided Language Displays	
Looking/Eye Pointing	PODD (Pragmatic Organisation Dynamic Display) books	
Challenging Behaviour	Communication Books	

Table 1.1 - Examples of AAC suitable for those with Rett Syndrome.

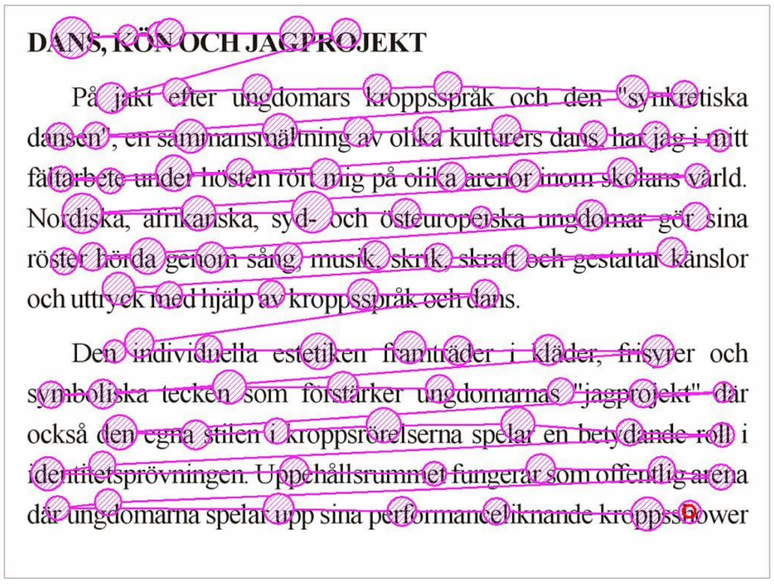
One of the more interesting technologies included in AAC is the use of Eye Gaze devices. These allow the user to have their eyes tracked, usually across a single plane such as a computer monitor or pair of glasses. Currently the commercial market for eye tracking is centred on PC applications that allow communication through selecting words or images that another person can interpret. However, these applications do not offer peripheral support that someone with a disability could use to interact with their immediate surroundings without the direct assistance of a carer. A benefit of this would clearly be increased independence for the user and the potential to increase quality of life.

A survey [4] (See Appendix A) created to gather data for this project was posted on the official Tobii Dynavox (A disability-focussed, eye tracking company) Community Facebook page [5]. One question asked parents or guardians which functions they believed those with disabilities would benefit from. The most popular options were “Picking up items dropped from a wheelchair”, “Turning off and on electrical equipment such as Lights or TVs”, “Grasping a toy or drink to bring towards the user” and “Feeding”. Other similar responses included the desire to use this technology to play with objects. One respondent, who is a teacher, explained that functions of the robotics such as picking up, releasing and throwing would motivate their pupils to interact and learn with these devices. It is clear, therefore, that the potential to develop technologies that will improve the lives and wellbeing of those with these disabilities is very large in this sector.

1.1 History and Applications of Eye-Tracking

The first major observations of eye tracking were made in 1879 by Louis Emile Javal, who observed that reading text was not the smooth sweep we might experience, but in fact a series of jagged movements with stops, known as fixations, and saccades, quick jerking movements. This can be seen in Figure 1.1. In that diagram, the circles show where the eye stopped (the fixations) and the lines show the movement (the saccades). The size of the circle denotes how long was spent at each fixation. The larger circle in the word "syd- och" shows the viewer was particularly interested by the odd gap in the middle of the word.

It wasn't until 1908 that the first eye tracker, a contact lens with an aluminium pointer and hole for pupil was created by Edmund Huey [6].


DANS, KÖN OCH JAGPROJEKT

På jakt efter ungdomars kroppsspråk och den "synkretiska dansen", en sammansmältning av olika kulturers dans, har jag i mitt fältarbete under hösten rört mig på olika arenor inom skolans värld. Nordiska, afrikanska, syd- och östeuropeiska ungdomar gör sina röster hörda genom sång, musik, skrik, skratt och gestaltar känslor och uttryck med hjälp av kroppsspråk och dans.

Den individuella estetiken framträder i kläder, frisyrer och symboliska tecken som förstärker ungdomarnas "jagprojekt" där också den egna stilen i kroppsrörelserna spelar en betydande roll i identitetsprövningen. Uppehållsrummet fungerar som offentlig arena där ungdomarna spelar upp sina performanceliknande kroppsshower

Figure 1.1 - Examples of Fixations and Saccades over a piece of text. [36]

In 1937, the first recording of a user's eye movements was filmed using light beams reflected off user's eyes [7]. This was done by the educational psychologist Guy Thomas Buswell and came to the conclusion that oral and silent reading significantly varies for the same person when following text.

By the 1990s, smaller and less intrusive optical eye trackers were being used to answer questions about human interaction with machines. Large, head mounted trackers (such as the one shown in Figure 1.2) observed the user's eye movement as they viewed various kinds of displays. As the web became more accessible, and eye-catching graphics were being used to draw attention, many analysists wanted to find out how effective different interfaces were for finding the commands users wanted. Additionally, the possibility of aiding those with disabilities begun investigation with the intent of tracking and recording eye movement in real time. [8]

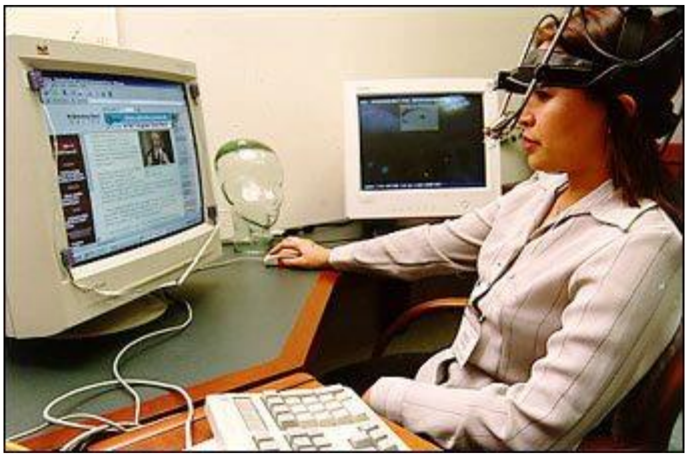


Figure 1.2 - Eye tracking setup and testing in the 1990s. [39]

There are three main categories of eye-trackers;

1. Measurement of the movement of something like a contact lens attached to the eye.
2. Optical tracking by reflecting light off the eye and measuring the change in reflections.
3. Measurement of potentials via electrodes placed around the eye.

Tight fitting contact lenses can have embedded magnetic field emitters whose locations are measured with magnetic search coils placed around the eye. As long as the lens does not slip significantly as the eye moves, researchers can measure eye movement in vertical and horizontal directions, as well as the torsion [9].

Placing electrodes around the eyes is a method that can be used even in complete darkness due to an electric potential field centred on the eye. It can be modelled as a dipole between the cornea and the retina, where the potential difference – changing through time to form a signal – is measured by two pairs of contact electrodes placed on the skin around one eye. This is called an Electrooculogram (EOG), described by Elwin Marg in 1951 and applied by Geoffrey Arden in 1962 [10].

The change in potential comes from each of the cornea and retina moving towards and away from various electrodes changing their potential. EOG is not used to track slow eye movement or gaze *direction* but has proved useful in showing gaze saccades, particularly associated with rapid eye movement (REM) sleep, an example of the signals is given in Figure 1.3. In this figure, the lines labelled LEOG and REOG, which stand for the EOG signals around the left and right eye respectively, show very similar (but mirrored) behaviour when entering REM sleep, as shown in the red box. The reason for the mirrored signals is simply to do with the mirrored orientation of the electrodes on the patient.

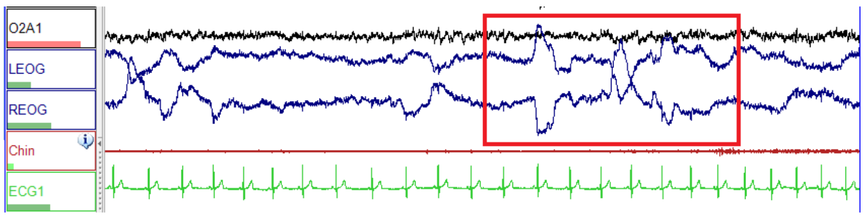


Figure 1.3 - EOG for the left and right eye during REM sleep (in red box). [37]

EOG is also a lightweight solution as opposed to head mounted eye trackers as it only requires electrodes and a measurement circuit. Any computation can be done remotely on a dedicated PC.

Due to the intrusion of the former and latter categories, the most commonly used eye tracker for consumer use are optical trackers. Optical trackers reflect infrared light off the eye which is detected by an optical sensor [11]. Data relating to the exact eye rotation is analysed by comparing the incoming angle of light to the outgoing angle, which is known by the hardware. A more advanced type, the dual-Purkinje eye tracker [12], uses reflection of the front of the cornea and back of the lens to track position.

Most modern eye trackers use cameras to focus on both eyes, specifically the pupil, and emit infrared or near-infrared light to create a corneal reflection. A direction vector is generated by calculating the difference between the centre of the corneal reflection and the location of the centre of the pupil (Figure 1.4). This vector can be extrapolated out to the point at which the eye is looking on a surface in the same geometric plane as the eye tracker [13]. This is known as Pupil Centre Corneal Reflection (PCCR).

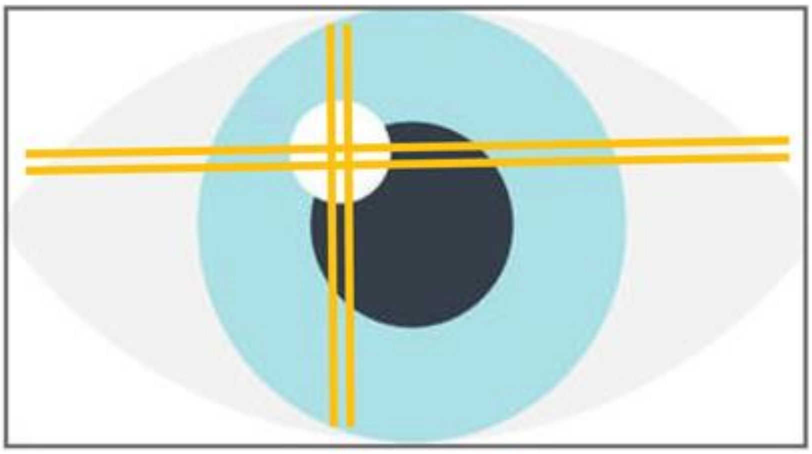


Figure 1.4 - Example of how the corneal centre is tracked [34]

Ordinary light sources luckily do not interfere with this tracking as the accuracy of gaze direction measurement is reliant only on the clear demarcation of the pupil and corneal reflection which is only possible through infrared imaging [14].

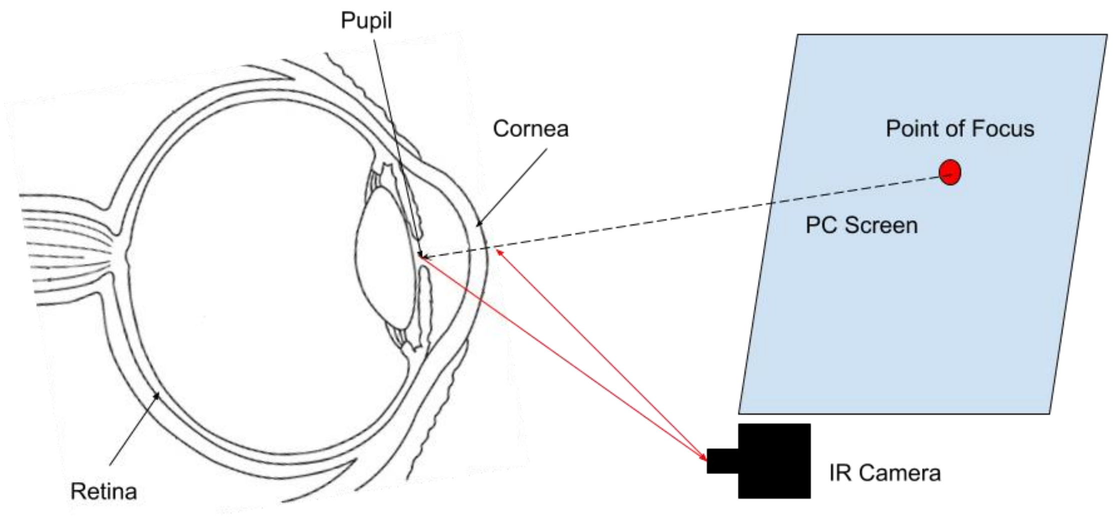


Figure 1.5 - Side-on diagram of gaze fixation [35]

Figure 1.5 shows the side-on view of PCCR. The point on the screen is where the pupil focuses, and the rotation of the eye creates a difference in the pupil position compared to the reflection of the IR light on the cornea.

Two different IR tracking techniques are used; bright pupil and dark pupil. The difference between these two is the illumination source with respect to the optics. If the IR light shines directly along the path of the gaze the optical sensor sees an image much like when camera flash gives people “red eye” in photos. This is a bright pupil effect.

If the IR light comes into the eye at an angle, only the corneal reflection is seen in a dark pupil effect (Figure 1.6). When using the bright pupil method, factors such as size of the pupil, environment of testing and the age of the user can impact the trackability of the eye. Ethnicity is also a factor, as Hispanics and Caucasians work well with bright pupil tracking but Asians work better with dark pupil tracking [15]. A good eye tracker will be able to assess each option on the current user and adapt its tracking technique to provide the most accurate performance for all users.

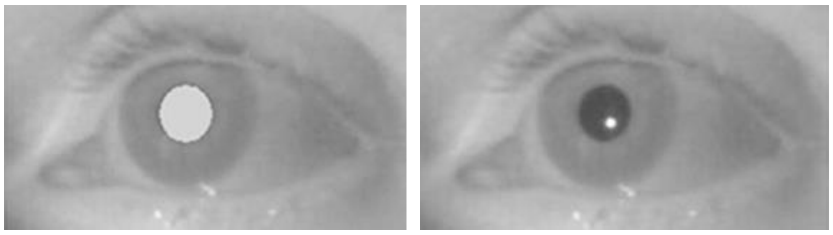


Figure 1.6 - Left: Bright Pupil. Right: Dark Pupil with Corneal Reflection. [42] [43]

1.2 Hardware Comparison

The first consideration to be made with this project is the model of Eye Tracker to be purchased. There are a variety of different options depending on the use of the tracker, as well as the technical capabilities. The search for these were narrowed down to the leading consumer eye-tracking company Tobii, based in Sweden, who provide a large number of tracking options for the general public, those with specific disabilities and those in research and development. The latter is often used for psychological testing [16], where those taking part have their gaze tracked when performing tasks on a screen.

Due to a limited budget, the consumer Tobii eye trackers are considered and compared in Table 1.2.

Device	**Hardware Features**	**PC Requirements**	**Price**
Tobii EyeX Figure 1.7	USB 3.0 40x30cm Headbox at 75cm 70Hz refresh CPU Load 10%	2.0 GHz quad core Intel i5 or i7. 8GB RAM. Windows 7 and up.	€109 [17]
Tobii Eye Tracker 4C Figure 1.8	USB 2.0 40x30cm Headbox at 75cm 90Hz refresh Head tracking (for extra inputs) CPU Load 1% (without Head Tracking)	2.0 GHz quad core Intel i5 or i7. 8GB RAM. Windows 7 and up.	€159 [18]
Tobii PCEye Mini Figure 1.9	USB 2.0 35x30cm elipse Headbox 60Hz refresh	1.0 GHz dual core. 2GB RAM. Windows 7 and up	£890 [19]

Table 1.2 - Eye-Tracker Hardware Comparison

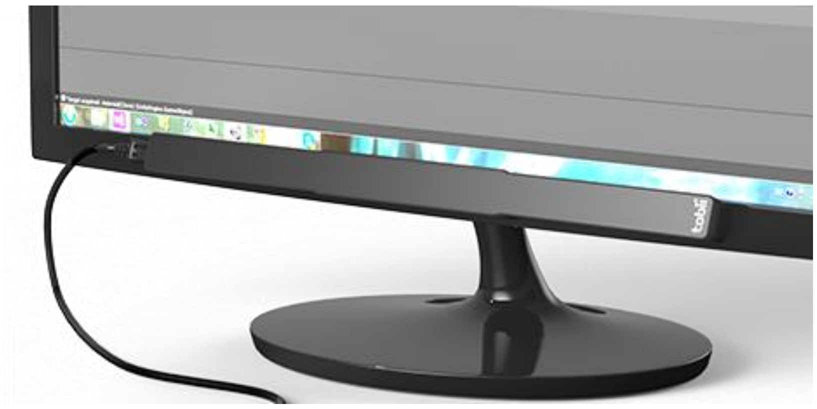


Figure 1.7 - Tobii EyeX Tracker mounted on a desktop PC monitor. [51]

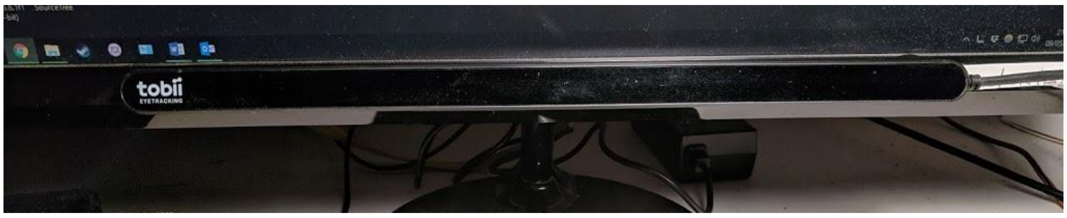


Figure 1.8 - Tobii Eye Tracker 4C mounted on a desktop PC monitor.

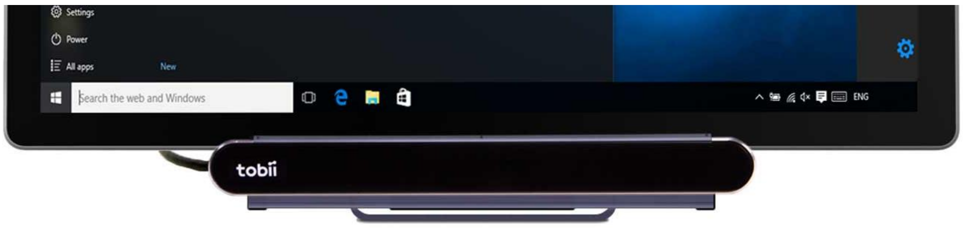


Figure 1.9 - Tobii PCEye Mini mounted on a tablet PC. [52]

From the options discussed, the decision was made to use the Tobii Eye Tracker 4C due to its high refresh rate, large headbox and head tracking capabilities. The headbox is an important factor, as it is the region of 3D space within which the user's head and eyes can be tracked. Outside of this space, it is uncertain as to whether the tracker will recognise a user being present or be able to track their eyes.

To provide a means of demonstrating that the application can communicate with, and control, a robot, a remote controlled vehicle will be built to carry out actions sent from the PC. The vehicle can be controlled by any microcontroller or micro-computer with wireless capabilities. There are a number of different off the shelf options that can be considered; Arduino and Raspberry Pi being the two most well-known brands.

The Raspberry Pi (RPi) is a credit card sized computer that uses a Linux based operating system and has a large number of General Purpose Input/Output (GPIO) pins for interacting with the outside world. The board (revision 3 and up) contains a built in wireless communication interface that supports both Wifi and Bluetooth. The advantages of using a Raspberry Pi to control the robotic vehicle is that the hardware is built onto a single board and is natively supported by the OS. This means that wireless communications do not require any third party libraries to make them work. Whilst Linux supports application development in C++, programs are usually written in Python on the RPi. Python is a high level interpreted language that has a plethora of built in IO libraries, such as those for accessing the GPIO pins and the Wifi module. The disadvantages of using an RPi are that development will have to take place on the device to be tested, and can be slow to process data due to the interpretation of the code instead of pre-compiling it. As well as this, the application to send, receive and parse data sent over wireless will have most of its computational resources used by the overhead of the operating system. For these reasons the RPi will not be considered for control of the robotics.

The Arduino, as opposed to the RPi, is a brand of a microcontroller board with a range of external pins and a small internal RAM and Flash ROM. Each board has a different CPU with varying speeds, sizes and IO capabilities. Two main options exist; the Arduino Uno and the Arduino Mega. The Uno has a smaller form factor than the Mega, though it has fewer pins for IO. The key differences between the two are shown in Table 1.3.

Arduino	**Processor**	**CPU Speed**	**EEPROM Size**	**Digital I/O**	**Price**
Uno	ATmega328P	16MHz	1kB	14	€19.00 [20]
Mega	ATmega2560	16MHz	4kB [21]	54	€35.00 [22]

Table 1.3 - Arduino Hardware Specification Comparison

The benefits that the Arduinos have over the RPi is that the microcontroller is programmed entirely by the developer, with no overhead in CPU usage caused by an operating system. Additionally, through use of libraries installed within the Arduino IDE many of the boards features, such as GPIO and Serial Ports, can be directly accessed through simple lines of code. A disadvantage is the lack of wireless communication present on the board natively. Instead, developers must connect Shields (PCBs that sit on top of the Arduino and extend the pin configuration) that have wireless capabilities. Fortunately, these shields are relatively cheap at around $15 [23] and have libraries included in the Arduino ecosystem. This ease of programming and adaptability for use with different hardware is the reason that Arduino will be used in this project. The Uno will be chosen above the Mega, due to the fact that not many Digital IO pins are required so a smaller form-factor and price makes the Uno the appropriate choice.

Having chosen the microcontroller, the final hardware consideration to be made is the wireless transceivers used with the Arduino. Whilst there is a huge selection of wireless hardware available off-the-shelf, three popular choices were;
The NRF24L01 2.4GHz Wireless Radio Transceiver Module,

Figure 1.10 - NRF24L01 Wireless Module [45]

The HC-12 UART Module,

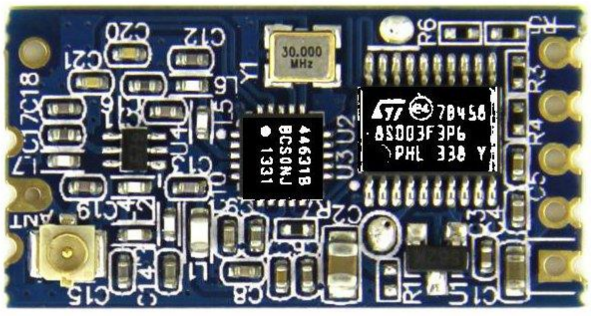


Figure 1.11 - HC-12 UART Module (without antenna coil). [41]

and the Digi Xbee 2.4GHz 802.15.4 RF module.

Figure 1.12 - XBee RF Module (with PCB antenna). [38]

They are compared in Table 1.4.

Module	Frequency	Range	Tx Current	Data Rate	Price
NRF24L01	2.4GHz	1000m	11.3mA	2Mbps	£2.00 [24]
HC-12	433-473MHz	1000m	100mA	5Mbps [25]	£7.00 [26]
Xbee	2.4GHz	1200m	33mA	250kbps [27]	£13.00 [28]

Table 1.4 - Wireless Module Hardware Comparison.

A key factor in the choice of these modules is the form factor they come in. Both the NRF and HC modules are on boards with pins designed to be used on a breadboard or other prototyping circuit base, whilst the Xbee has lower profile pins to be used with custom Arduino shields. This, along with an increased range and fairly low current draw makes it a suitable option for use on a robotic vehicle that could be driving around a large area. The lower bit rate is not an issue, as messages will rarely be bigger than a few bytes so crucial data will not be ignored do to slow speeds.

1.3 Tobii Eye Tracker 4C Deep Dive

The basis of the Tobii Eye Tracker 4C is the IS4 eye tracker board shown in Figure 1.13. This is the prebuilt solution Tobii provides OEMs for complete integration into their own hardware, such as laptops or All in One PCs.

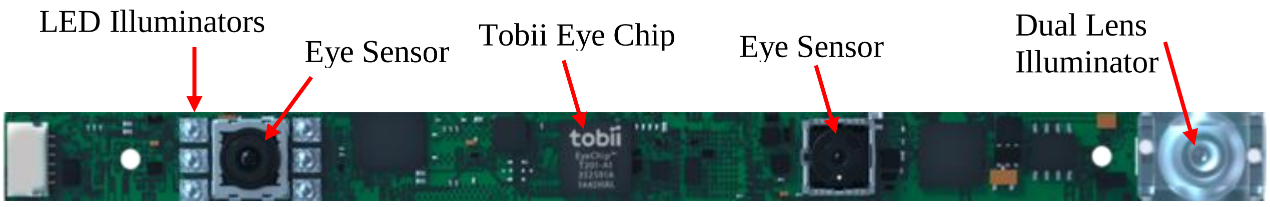


Figure 1.13 - Tobii IS4 Circuit Board [40]

The following key components are present on the IS4;

1. LED Illuminators. These are high density LEDs that form an illumination cluster around the Eye Sensor. They create a pattern of near-infrared light on the eyes which is then captured by the cameras. This is key to actually tracking the differentials between the pupil and cornea, to then calculate the position of gaze.
2. Eye Sensor. A custom made, low power but high resolution camera used for the eye tracking and biometric data collection functions of the IS4 [29]. It captures near-infrared images of the eye and the reflections of the illuminators to then be passed for processing.
3. Tobii EyeChip. This is Tobii's System on Chip Application Specific Integrated Circuit (SoC ASIC) designed to implement their Tobii EyeCore eye tracking algorithm. The EyeChip reduces the CPU load on the host computer by carrying out the calculations itself. It has on-chip Image Signal Processing, which claims to reduce the total power consumption by up to 15 times as compared to a 45 Watt Intel i7 host processor [30]. Dual channel accelerated tracking hardware engines with specific vector instruction sets and large data caches (512Mb LPDDR) allow the EyeChip to outperform standard consumer processors in these specific eye tracking tasks.
   The algorithm supports the following functions:
   - Eye tracking.
   - Head position and tracking.
   - Face recognition, identification and authentication.
   - Facial feature tracking.
   - Gestures.
   - Presence.

   A benefit for OEMs when integrating the EyeChip into their own devices is that it requires a bandwidth of just 100kbps compared to 100Mbps in competitors. Additionally, it supports transferring data over USB 2.0, I2C, SPI, JTAG, RS232 and UART.

4. Another eye sensor. This one is a lower power camera with a Standard Mobile Imaging Architecture (SMIA) socket, modified for enhanced eye and head tracking. This captures light in the visible spectrum for use with facial recognition and other functions that require image processing of facial features.
5. Dual lens illuminator. Provides illumination to the face and eyes with 30% higher efficiency to aid with face and eye tracking. High power illuminators are not required here as a pattern is not being tracked, only the face needs lighting to help the cameras focus on details in imperfect lighting conditions.

All IS4 based devices utilise the Tobii Engine through the Tobii Core SDK, or software development kit [31]. This translates the raw coordinates output from the hardware into useful information about where the user is looking. Using these libraries, developers can access this data and implement it into PC applications – as being attempted in this project. Two interfaces are contained in this engine:

- The Stream Engine – this delivers a stream of data such as the gaze point, eye position and other biometric data.
- The Interaction Engine – this sets events that trigger when gaze enters a certain region with other conditions also being present. This is how games and other reaction-time applications can be implemented.

Within these engines there is functionality that can be applied to elements within a GUI to enhance it with eye tracking capabilities. This will be explained further in the Test Application Development section.

During a discussion with a Tobii Dynavox representative, the issue of Dwell Time was brought up. Dwell Time is the time required for a user to rest their gaze on a particular area of the screen until an action is performed.

They explained that this time can vary hugely between users, a factor of which is their capacity to understand the software they are using. As an example, someone who is unfamiliar with a communication board software will often look at all the buttons to see what they do. If the dwell time is too high (~2s) they may lose interest very quickly and move onto the next button. If all buttons have this high dwell time the user may become frustrated and not want to use the software.

On the opposite extreme, if the dwell time is too low (~0.3s – human reaction speed) the user could activate buttons just by passing their gaze over them. In the case of a communication board this can activate phrases that the user did not mean to say. The representative explained that the best software packages have a setting for this dwell time, which can be tailored to particular user configurations and said that this application too would benefit from this setting.

1.4 Project Aims and Milestones

The aims of this project are not only to develop a physical product but also demonstrate the ability to learn to use new devices and technologies that have not previously been implemented in such a project.

Aims:

- Learn to develop graphical Windows applications using Visual Studio.
- Learn to use the Tobii Eye-Tracking SDK
- Demonstrate the ability to communicate with a wireless Arduino module within a Windows application.
- Learn to control motor drives over wired or wireless communications.
- Learn reverse and forward kinematics to control the gripper end of a robotic arm.

Milestones:

1. Demonstrate ability to track eyes and gaze in a self-contained Windows application.
2. Communicate with an Arduino module from the Windows application to turn an LED on and off.
3. Add functionality to the Arduino module to add gaze-tracking capabilities to a remote-controlled vehicle
4. Implement forward kinematics to robotic control code, and allow control over full arm range.
5. Apply control scheme within the application to control a uArm Metal robotic arm using reverse kinematics.

Deliverables:

1. Windows application that communicates with both the eye tracker and Arduino module simultaneously.
2. Mode within same application to communicate with the robotic arm.

2 Programming Environment Setup

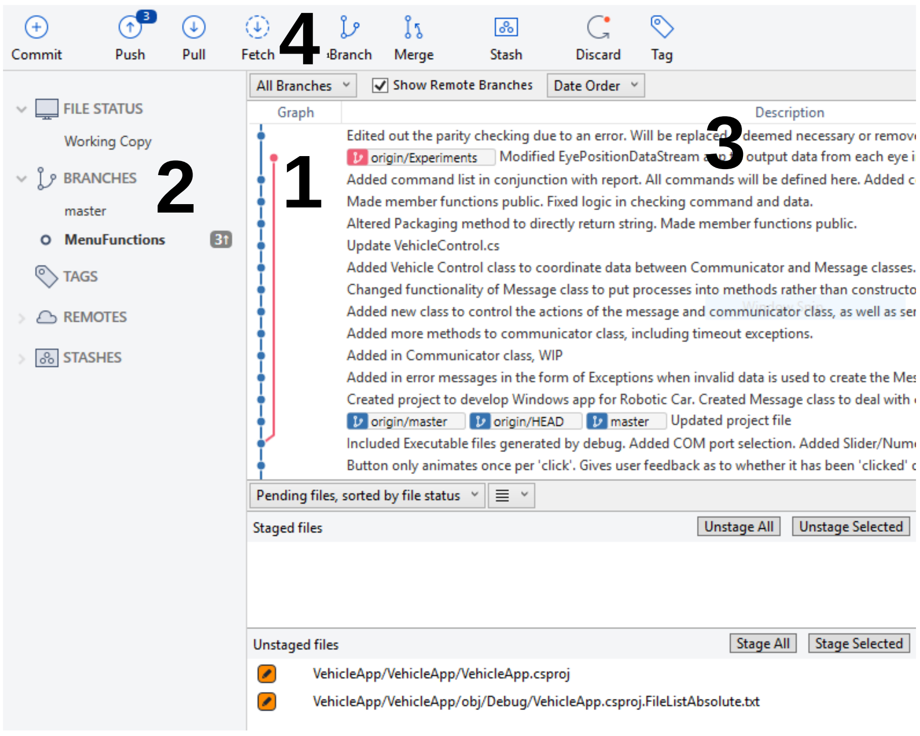
The two pieces of hardware connected to the host PC are the Arduino, via either a wired or wireless (Bluetooth or Wi-Fi) connection, and the Tobii Eye Tracker 4C. Each of these uses a different programming environment due to the requirements of the Software Development Kit (SDK). The code written will also be saved and versioned into a repository to track progress and revert to previous versions if an error occurs.

2.1 Software Version Management

The online GitHub repository service will be used to store the code written. Placing the code online [32] [33] will allow it to be viewed by anyone and protects the developer from computer failures, potentially losing valuable code if hardware fails and local data is lost.

Two separate GitHub repositories will be started: One for the Eye Tracker application and one for the Arduino sketches. These are two separate programs so will be treated as such and development will be carried out in parallel to maximise the efficiency of tasks detailed in the time plan in Figure 5.2.

On the development PC, SourceTree is installed to locally handle Git push/pulls with a GUI, rather than text commands. This is linked to the public GitHub account and has been chosen based on existing knowledge of its operation. The default view is shown in Figure 2.1.



Edited out the parity checking due to an error. Will be replac deemed necessary or remov
origin/Experiments Modified EyePositionDataStream a output data from each eye i
Added command list in conjunction with report. All commands will be defined here. Added c
Made member functions public. Fixed logic in checking command and data.
Altered Packaging method to directly return string. Made member functions public.
Update VehicleControl.cs
Added Vehicle Control class to coordinate data between Communicator and Message classes.
Changed functionality of Message class to put processes into methods rather than constructo
Added new class to control the actions of the message and communicator class, as well as se
Added more methods to communicator class, including timeout exceptions.
Added in Communicator class, WIP
Added in error messages in the form of Exceptions when invalid data is used to create the Me
Created project to develop Windows app for Robotic Car. Created Message class to deal with
origin/master origin/HEAD master Updated project file
Included Executable files generated by debug. Added COM port selection. Added Slider/Num
Button only animates once per 'click'. Gives user feedback as to whether it has been 'clicked'

VehicleApp/VehicleApp/VehicleApp.csproj
VehicleApp/VehicleApp/obj/Debug/VehicleApp.csproj.FileListAbsolute.txt

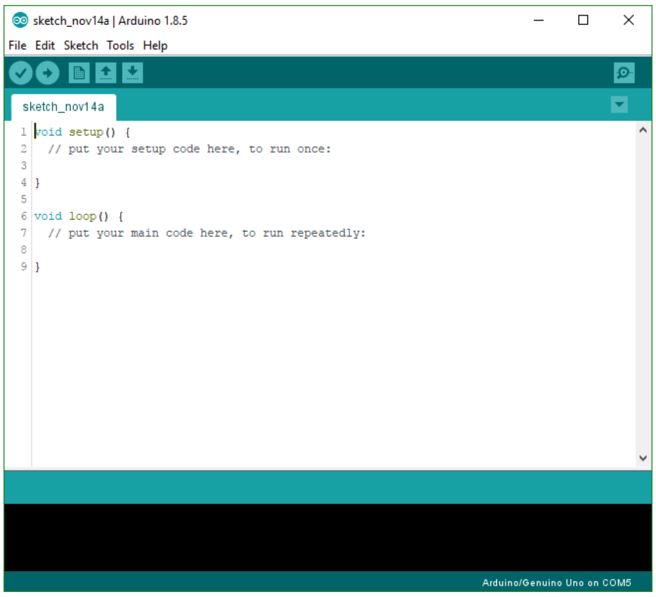
Figure 2.1 - SourceTree Default Git Tree View

The key features of the program are as follows:

1. Development Tree: Shows every code commit made, and the user that made them. If branches have been developed on, these show in parallel to the "origin" branch which shows the current development path. When a feature is merged into the Origin branch, the tree shows this too. On the tree, any commit can be checked out and the files as they were in that commit will replace the current files locally stored on the device accessing it.
2. Branch List: The names of the branches split off from the master branch.
3. Commit information: Shows a list of files changed, which lines were changed and the time/date/author of these committed changes. An explanation by whoever made the changes is also listed.
4. Push/Pull/Fetch: Pushes any local commits, pulls any pending commits from the Git server or fetches the commits without changing any local data in the repository.

2.2 Arduino

The Arduino will be programmed within the Arduino software, Figure 2.2, a basic C IDE and compiler with built in support for all Arduino hardware and access to their library documentation.

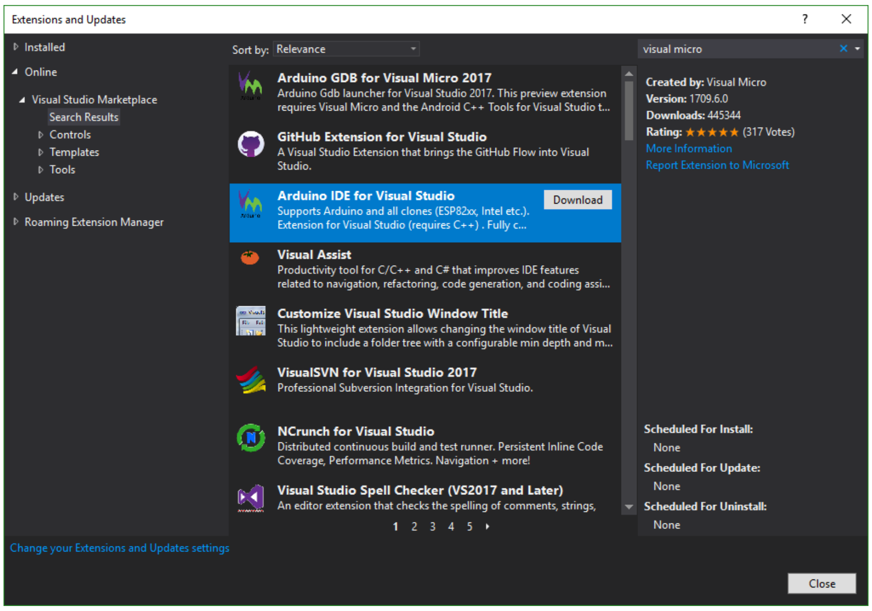

sketch_nov14a | Arduino 1.8.5
File Edit Sketch Tools Help
sketch_nov14a
Arduino/Genuino Uno on COM5

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Figure 2.2 - Arduino IDE Blank Project

A downside to using this IDE is the lack of hardware debugging available. For very basic code, such as blinking an LED based on the input from the serial connection, this is not a big deal. Later in the project an extension to Visual Studio – Arduino IDE for Visual Studio – will be used. This brings advantages of having the same familiar user interface as Visual Studio, with the added benefit of being able to step through critical code, such as that controlling the vehicle or arm.

The extension is installed through Tools > Extensions and Updates as demonstrated in Figure 2.3 and Figure 2.4.


Extensions and Updates

Installed
Online
Visual Studio Marketplace
Search Results
Controls
Templates
Tools
Updates
Roaming Extension Manager

Sort by: Relevance

visual micro

Arduino GDB for Visual Micro 2017
Arduino Gdb launcher for Visual Studio 2017. This preview extension requires Visual Micro and the Android C++ Tools for Visual Studio t...

GitHub Extension for Visual Studio
A Visual Studio Extension that brings the GitHub Flow into Visual Studio.

Arduino IDE for Visual Studio
Supports Arduino and all clones (ESP82xx, Intel etc.). Extension for Visual Studio (requires C++) . Fully c...
Download

Visual Assist
Productivity tool for C/C++ and C# that improves IDE features related to navigation, refactoring, code generation, and coding assi...

Customize Visual Studio Window Title
This lightweight extension allows changing the window title of Visual Studio to include a folder tree with a configurable min depth and m...

VisualSVN for Visual Studio 2017
Professional Subversion Integration for Visual Studio.

NCrunch for Visual Studio
Distributed continuous build and test runner. Persistent Inline Code Coverage, Performance Metrics. Navigation + more!

Visual Studio Spell Checker (VS2017 and Later)
An editor extension that checks the spelling of comments, strings,

1 2 3 4 5

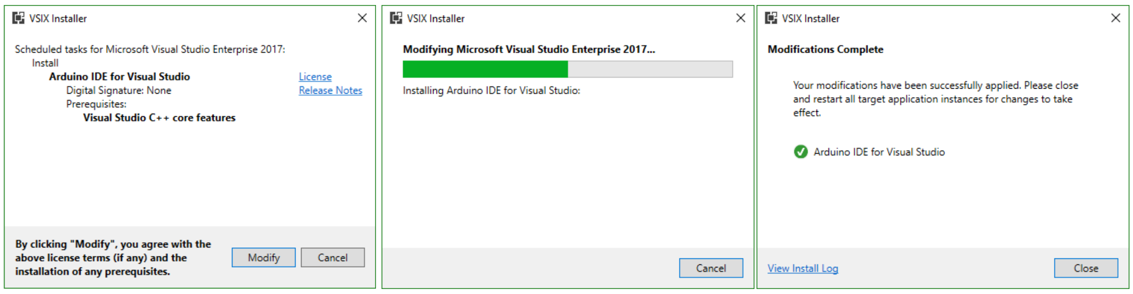
Created by: Visual Micro
Version: 1709.6.0
Downloads: 445344
Rating: ★★★★★ (317 Votes)
More Information
Report Extension to Microsoft

Scheduled For Install:
None
Scheduled For Update:
None
Scheduled For Uninstall:
None

Change your Extensions and Updates settings

Close

Figure 2.3 - Visual Studio Extensions and Updates Dialog


VSIX Installer

Scheduled tasks for Microsoft Visual Studio Enterprise 2017:
Install
Arduino IDE for Visual Studio License
Digital Signature: None Release Notes
Prerequisites:
Visual Studio C++ core features

By clicking "Modify", you agree with the above license terms (if any) and the installation of any prerequisites.
Modify Cancel

VSIX Installer

Modifying Microsoft Visual Studio Enterprise 2017...
Installing Arduino IDE for Visual Studio:
Cancel

VSIX Installer

Modifications Complete

Your modifications have been successfully applied. Please close and restart all target application instances for changes to take effect.

✓ Arduino IDE for Visual Studio

View Install Log Close

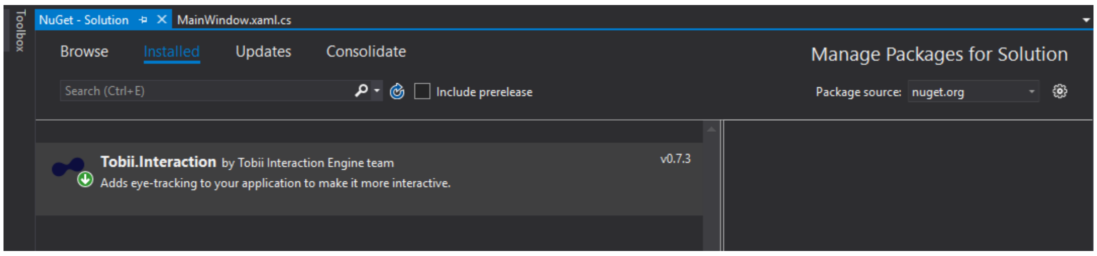
Figure 2.4 - Installation Procedure of Arduino IDE for Visual Studio

These two IDEs will allow the full range of Arduinos to be programmed and tested, meaning their setup should not change if hardware changes are required.

2.3 Tobii 4C Eye Tracker

The Windows application with eye-tracking built in will be developed within Visual Studio 2017 Enterprise. Tobii have provided a “Getting Started” [34] guide for developing console and WPF applications, which also covers the installation of the Tobii Core SDK into Visual Studio.

Upon the creation of the solution, NuGet is used to download the Tobii.Interaction, Tobii.EyeX.Client and Tobii.EyeX.Framework APIs and import them into the solution. To do this, right click on the solution name in the Solution Explorer and select “Manage NuGet Packages for Solution…”. Searching for these APIs in the “Browse” section allows the download of either the current stable release, or a certain version. The version used for this project is v0.7.3 for Tobii.Interaction, v1.8.503 for Tobii.EyeX.Framework and v.1.8.504 for Tobii.EyeX.Client. An example of the Interaction engine installed is shown installed in Figure 2.5.



Tobii.Interaction by Tobii Interaction Engine team
Adds eye-tracking to your application to make it more interactive.
v0.7.3

Figure 2.5 - NuGet Package Installer

A separate application developed by Tobii [35] has also been installed on the development PC which provides calibration features for setting up the eye tracker. It will be assumed that the user of the software will already have this installed and calibrated for all users of that device. It is possible to check this software version within the application, as is demonstrated in the example projects provided by Tobii. This has the use of having a minimum version that allows the software to operate.

3 Test Application Development

Within the Tobii Core SDK, a number of useful functionalities are present within the stream engines. The first is that the eye-gaze point and fixations can be accessed. These are where your eyes are looking and where your gaze rests, respectively. This is used to tell the program where the points of interest to the user are. Fixations are read by a `FixationDataStream` and output the X and Y coordinates on the screen where the fixation has been held. Three event types; `Begin`, `Data` and `End`, tell the program how long the fixation lasted when used in conjunction with time stamps.

The next useful capability is user presence. This is a single variable that tracks whether a user has been detected in front of the eye tracker and, therefore, is ready to use the application. This is particularly useful when an app can pause itself when a user leaves the computer. It also allows the reduction of CPU consumption by turning off the main eye tracking until the user returns. User presence is handled using the `UserPresenceChanged` event handler which notifies the program that a change has been recognised. The program can then invoke a new `Action` that checks the `EngineStateValue` and performs an action based on the user's presence.

In Graphical User Interfaces (GUIs) there are three main interactable 'regions' that can be implemented into an application such as the one discussed in this report. Gaze-Aware regions know when the gaze has entered and left the region, Activatable regions can be clicked or focused by the user's gaze, and Pannable regions can be scrolled or panned by placing the gaze at the edge or corner of a UI element. All three of these are accessed through a `BehaviourMap`, part of the `EyeXFramework` API. For each respective type, a `GazeAwareBehaviour`, `ActivatableBehaviour` or `PannableBehaviour` is added to the behaviour map, linked to a button or other GUI element. When the appropriate gaze interaction is made with that element, the event fires and relevant code is run.

One of potential downsides to using these behaviours is that they only focus the element, not interact with it. This means that a user can look at a button, but that button will not be pressed until another action is performed, such as pressing a key on the keyboard. This is demonstrated in the Interaction Samples available from Tobii [36], and shown in Figure 3.1.

```
private void OnKeyUp(object sender, KeyEventArgs keyEventArgs)
{
 Console.WriteLine("OnKeyUp: " + keyEventArgs.KeyCode);

 if (keyEventArgs.KeyCode == Keys.ShiftKey)
 {
     Console.WriteLine("TriggerActivation");
     Program.EyeXHost.TriggerActivation();
 }
 keyEventArgs.Handled = false;
}

private void OnKeyDown(object sender, KeyEventArgs keyEventArgs)
{
 Console.WriteLine("OnKeyDown: " + keyEventArgs.KeyCode);
 if (keyEventArgs.KeyCode == Keys.ShiftKey)
 {
     Console.WriteLine("TriggerActivationModeOn");
     Program.EyeXHost.TriggerActivationModeOn();
 }
 keyEventArgs.Handled = false;
}

private void OnButtonActivated(object sender, EventArgs e)
{
 var button = sender as Button;
 if (button != null)
 {
     Console.WriteLine("OnButtonActivated");
     button.PerformClick();
 }
}
```

Figure 3.1 - Key Press Trigger Example

In the code snippet, `Program.EyeXHost.TriggerActivationModeOn()` sets the program up to accept triggers, whilst `Program.EyeXHost.TriggerActivation()` actually performs the trigger. Earlier in the code the buttons were linked to the `OnButtonActivated` method, meaning this is what the trigger calls. This then calls the `button.PerformClick()` method to have the button click and perform its action like a normal click would. The key to making a program fully accessible through eye tracking methods lies in how this click action is activated.

The disadvantage of using this SDK is that the app developed will only be compatible with Tobii 4C Eye Trackers, not those made under the Tobii Dynavox Brand. From interactions with Tobii Developer Support, the required SDK and test units are more expensive than those being used in this project, due to a limited budget. As a result this project will serve as a proof of concept that can later be ported to the Dynavox trackers.

3.1 Headbox Testing

The main limitation of screen mounted head trackers is their ability to track eyes and other facial features within a limited three-dimensional region called the Headbox. It is useful to know the practical limits of this box, and what happens to the data received by the eye tracker when these limits are exceeded.

To collect this data, a program was modified from the examples provided by Tobii. This program collects the absolute and normalised 3D position of each eye and saves it to individual .csv files, “OutputLeft.csv” and “OutputRight.csv”. The absolute 3D position consists of X, Y and Z coordinates expressed in millimetres in relation to the centre of the screen. The normalised coordinates are expressed in relation to the headbox, as a fraction of the full volume. These are visually demonstrated in Figure 3.2 and Figure 3.3.

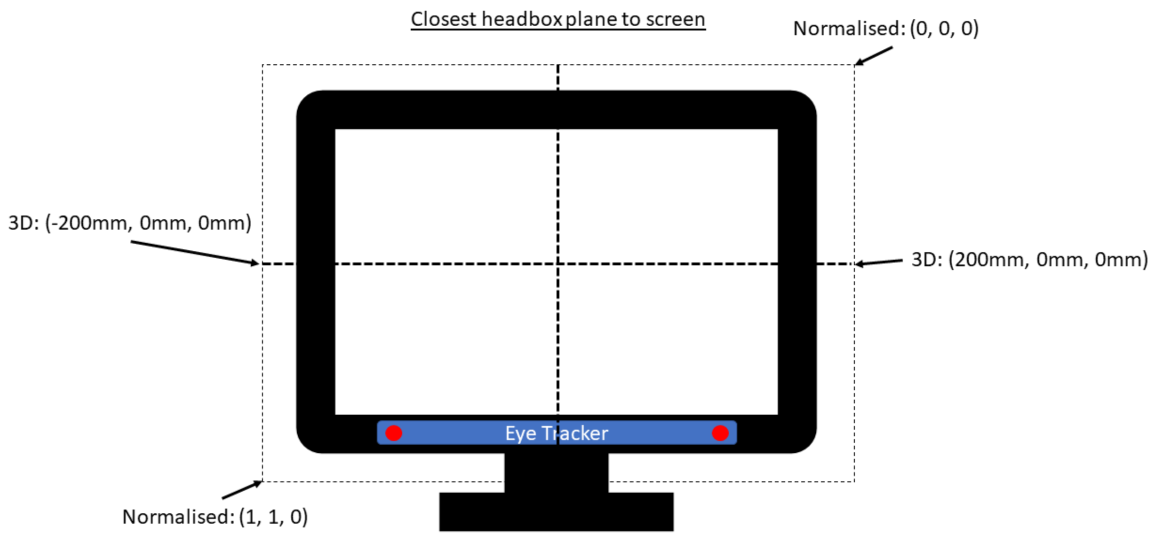


Figure 3.2 - Front view of screen with headbox.

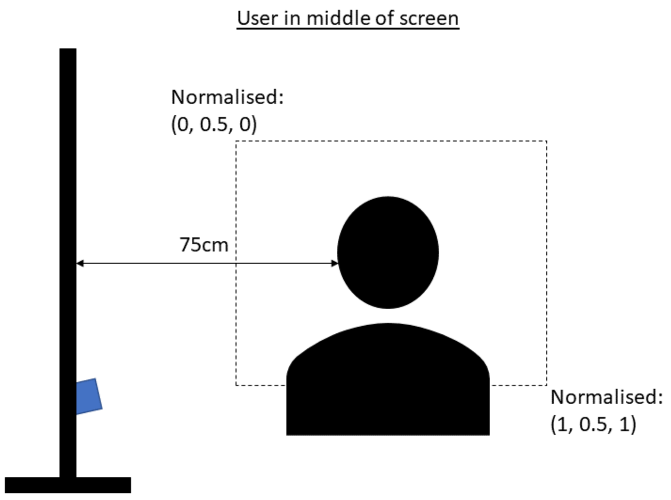


Figure 3.3 - Side view of screen with headbox.

The headbox boundaries were tested by a user moving their head around whilst the PC collected the data on their eye positions. Three tests were performed; one where the head was moved towards and away from the display (along the Z axis), one where the head was moved left and right along the display plane (along the X axis) and one where the head was moved up and down along the centre of the display (along the Y axis). Investigating these axes separately makes it easier to assess the data in each output file, and compile it into a single file to get the complete range of movement.

In the following graphs, only the normalised data is used to assess positions. It was found that the absolute 3D positions returned data without the power of ten component making the data recorded fluctuate wildly when exiting the headbox (the next value after 699mm was 7mm, when it should have been 70cm).

3.1.1 Results

Moving along the X axis of the display, the changes in Figure 3.4 were observed. It can be seen that the entire theoretical headbox can almost be fully explored as the left eye can go almost to the very left of the box, and the right eye can go almost to the very right of the box. It is also interesting to note that the eye spacing is not constant throughout, there are some anomalies where the eyes seem to get closer together. This of course is not the case and a maximum fluctuation of only around 0.05 is observed.

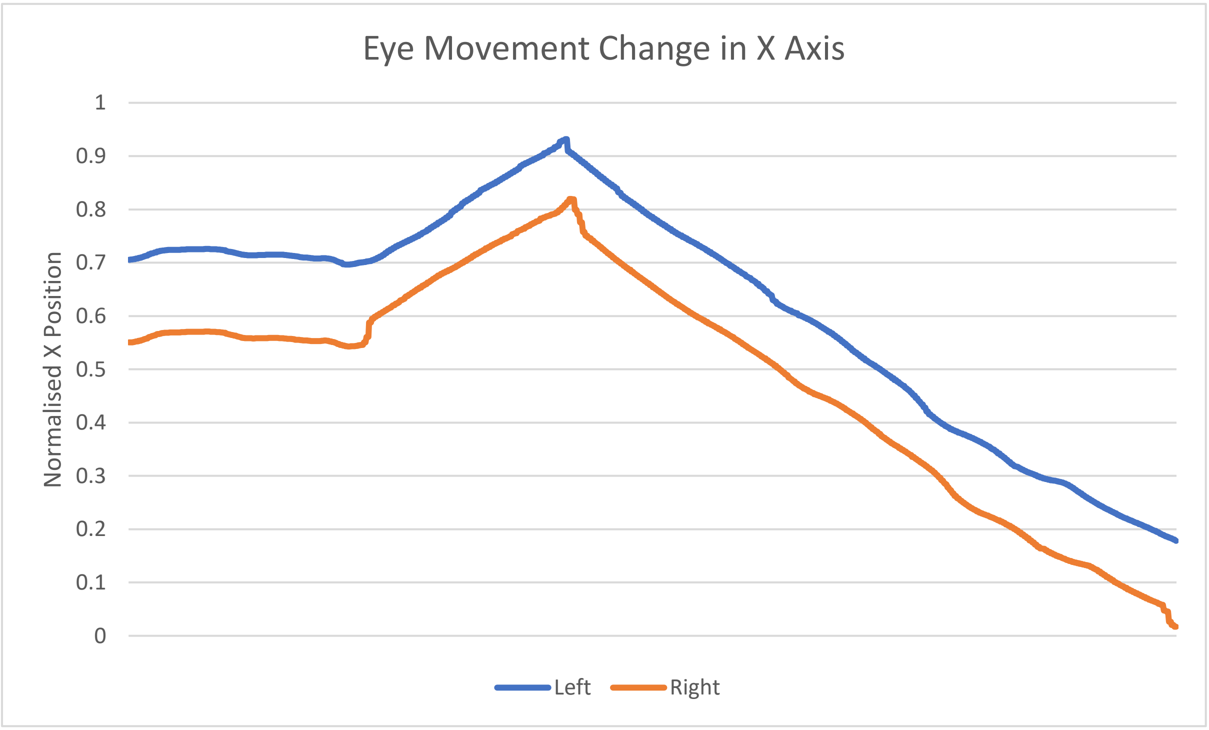


Figure 3.4 - Movement across X Axis

The Y axis shows a much tighter range of movement in Figure 3.5. Eyes could not be read any higher than 50% up the headbox, and not lower than 20%. This is a sensible reading however, as a user will be seated at the same height for the duration of them using an application. Movement of more than 30% of the headbox, which is 9cm, is very unlikely.

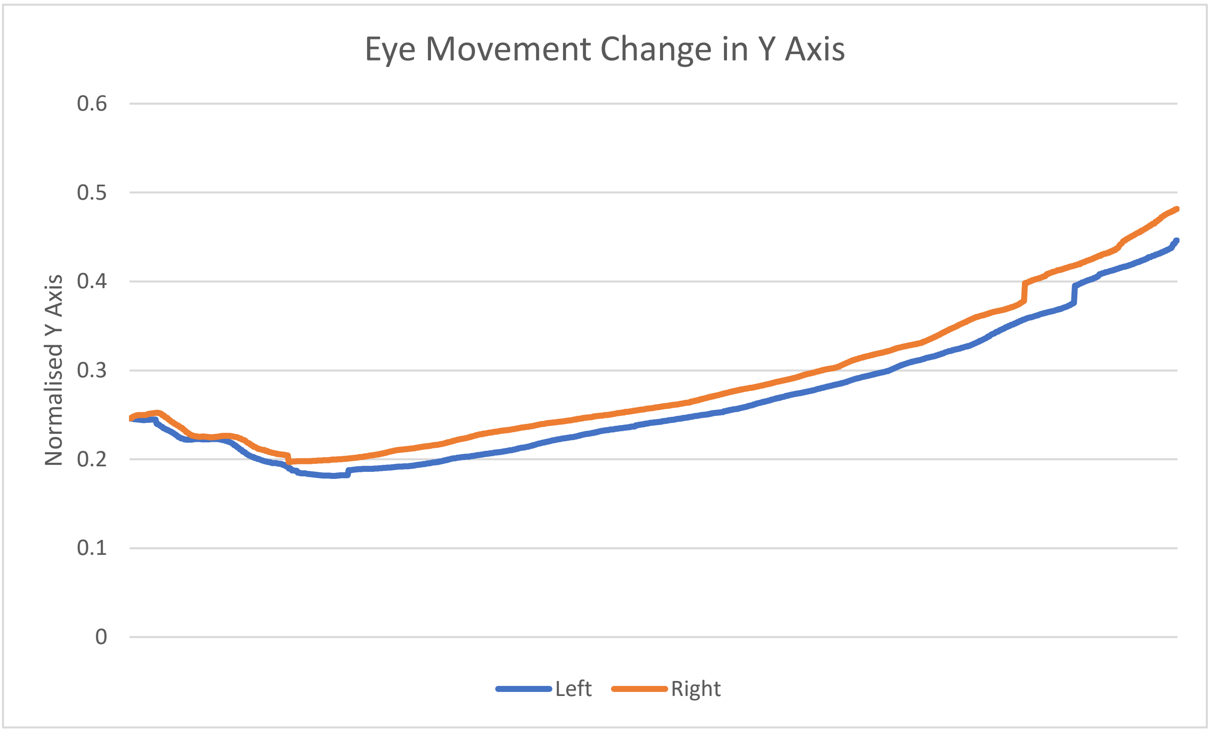


Figure 3.5 - Movement across Y Axis

On the contrary to the Y axis, the Z axis shows a much greater range than expected as shown in Figure 3.6. This shows it is possible to be further away from the screen than the recommended 75cm, up to 105cm. It wasn’t possible to get closer than half of the headbox, which is around 30cm. This is due to the cameras being far apart on the Tobii 4C and the eyes leaving their range of view.

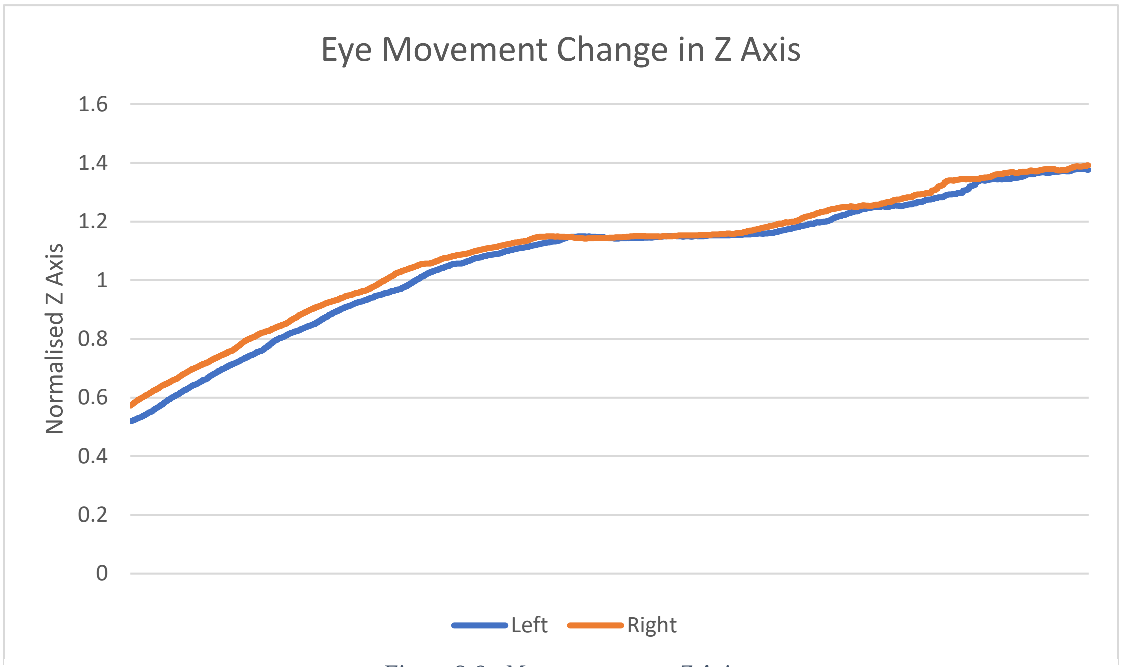


Figure 3.6 - Movement across Z Axis

It is unlikely that a user will move back and forwards from the screen very often, but it is good that the distance a user can be from the screen is large enough to be comfortable for most people.

3.2 Basic Program Demonstration

Using the guide provided by Tobii, a sample Windows Presentation Foundation (WPF) was created to show off how an application can react to the gaze being focused on the window. Once calibrated, a user can change the colour of the window from White to Magenta by looking at it, seen in Figure 3.7.

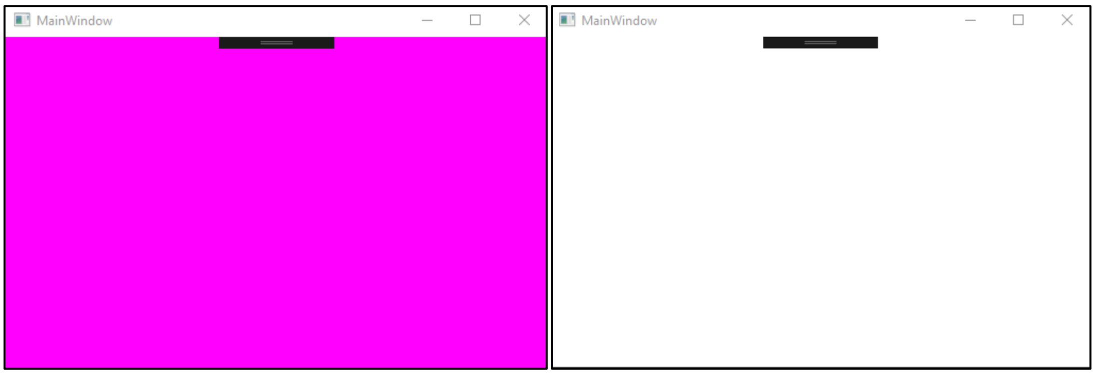


Figure 3.7 - Left: Gaze on Window. Right: Gaze not on Window.

This demonstrates the behaviour `IsGazeAware` and the `HasGaze` flag. Figure 3.8 shows the code snippet that responds to these flags.

```
01 <Window x:Class="TobiiWPFTest.MainWindow"
02     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
03     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
04     xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
05     xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
06     xmlns:tobii="clr-namespace:Tobii.Interaction.Wpf;
                 assembly=Tobii.Interaction.Net"
07     xmlns:local="clr-namespace:TobiiWPFTest"
08     mc:Ignorable="d"
09     Title="MainWindow" Height="350" Width="525">
10     <Grid x:Name="LayoutRoot"
11           tobii:Behaviors.IsGazeAware="True">
12         <Grid.Style>
13             <Style TargetType="Grid">
14                 <Setter Property="Background" Value="White" />
15                 <Style.Triggers>
16                     <Trigger Property="tobii:Behaviors.HasGaze" Value="True">
17                         <Setter Property="Background" Value="Magenta"/>
18                     </Trigger>
19                 </Style.Triggers>
20             </Style>
21         </Grid.Style>
22     </Grid>
23 </Window>
```

Figure 3.8 - XAML code for WPF Demonstration.

Line 6 imports the Tobii Interaction library into the project, which allows the inclusion of `Gaze Aware Behaviours`. The first example is on Line 11, where the program gives the XML Grid element named `Layout Root` its `IsGazeAware` behaviour. This means that the form is now checking the co-ordinates of the user's gaze and comparing it to the co-ordinates of the Grid element, which is contained within the window. As the window is moved around the screen, those co-ordinates change. However this is all dealt with by the Tobii Interaction engine in the background, so a developer only needs to focus on what happens when the gaze is placed within that window.

A trigger is placed within the Grid element, and linked to the property `HasGaze`. As the name suggests, it checks if the element has the users gaze and then performs an action within the Trigger handle. In the case of this program, the background property of the Grid changes from White to Magenta. When the user's gaze is removed, the background changes back to White from Magenta.

3.3 LED control using Eye Tracking in WinForms

Tobii provides a large set of examples for developers to experiment with, and one such set is found in a Tobii GitHub repository [36]. Within this repository there are older examples from 2014 that use the EyeX.Framework API rather than the Interaction API. These examples utilise the WinForms framework to create applications that are designed with simple drag and drop components and little XAML knowledge required. Using this approach will reduce the complexity of the program, allowing more time to be spent on actual programming of the internal logic rather than learning XAML and debugging two different languages. This is coupled with previous experience with using WinForms and debugging programs within that framework.

The first milestone reached is the completion of a basic Windows application to translate eye movement into a signal sent to an external device.

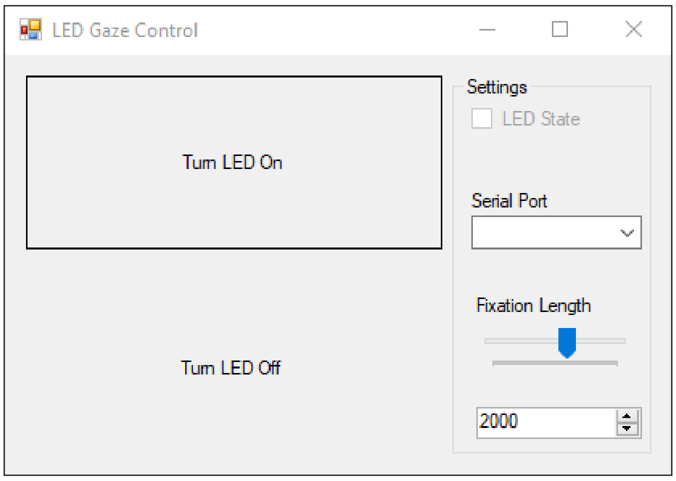


Figure 3.9 - Application to control LED by Eye Gaze.

This app demonstrates the ability to simultaneously open a COM port on the host PC and accept eye tracking data from the Eye Tracker. The two buttons on the left-hand side of the app (Figure 3.9), labelled "Turn LED On" and "Turn LED Off" are both `Gaze Aware` elements that react to the user's gaze applied to them. These buttons have to be added to the Behaviour Map, which itself is connected to the `EyeXHost` when the form is initialised, shown in Figure 3.10.

```
Program.EyeXHost.Connect(behaviorMap1);
behaviorMap1.Add(btnON, new GazeAwareBehavior(OnGaze));
behaviorMap1.Add(btnOFF, new GazeAwareBehavior(OnGaze));
```

Figure 3.10 - Adding Eye Gaze interaction behaviour

This code links the two buttons to a `GazeAwareBehavior` function called `OnGaze`, which will be called then the gaze is on either of the buttons, shown in Figure 3.11.

```
01 private void OnGaze(object sender, GazeAwareEventArgs e)
02 {
03  DisposeTimer();
04  TimerSetup();
05  var button = sender as Button;
06  if (button != null)
07  {
08      if (e.HasGaze)
09      {
10          if (!btnOnSelected || !btnOffSelected)
11          {
12              button.BackColor = Color.Green;
13              selectedButton = button;
14              timer.Enabled = true;
15          }
16      }
17      else
18      {
19          button.BackColor = Color.Transparent;
20          timer.Enabled = false;
21      }
22  }
23 }
```

Figure 3.11 - OnGaze method to select buttons.

This method has been simplified to apply to any button on the form, as long as it has the `OnGaze` Behaviour. On line 8, when the `GazeAwareEvent` has the user's gaze, the buttons change their background colour to green to make the user aware that they are selecting that button and in the background a timer will begin to count down. If the timer reaches zero whilst the user's gaze is on the same button the entire time, the program considers that button to be "clicked" and will carry out a method as if it had been clicked normally. If at any time, before the timer reaches zero, the user removes their gaze from the button the timer will reset (line 20) and not start again until another button is looked at.

In this app, a slider and numerical input box are present to allow us to change the timer length in milliseconds, though a default of 2000ms is set. On the other end of the application is a connection to an Arduino, connected to one of the PCs COM ports. This can be selected in the "Serial Port" drop down box and will establish a connection with the parameters in Table 3.1.

Parameter	**Value**
Port Name	User's choice
Baud Rate	9600
Data Bits	8
Handshake	None
Read Timeout	500ms
Write Timeout	500ms

Table 3.1 - Connection Parameters

As this is an example application it didn’t feel necessary to require these settings to be changed. The Arduino itself has been programmed to accept a single bit (1 or 0) to turn on or off its internal LED. This provides a visual indicator that the app has successfully translated the eye movements of a user into data that can be understood by a microcontroller.

This is a very useful experimental application as it overcomes a major hurdle that was found when trying to use the WPF examples; that it was very difficult to implement a “soft click” by holding the users gaze. In attempts to find out how to do this a question was submitted to the Tobii Developer Forums. It was through this that the extra EyeX.Framework examples were provided and modified with a timer to achieve the desired functionality.

3.4 Xbee Setup and Testing

The first major milestone is to be able to control a robotic vehicle using eye tracking. As this vehicle cannot be wired to the PC when in use, a wireless interface must be used to transfer data between the two. This is where the Xbee modules must be implemented.

For the setup and initial testing of the functionality of the Xbee 802.15.4 radios, the instructions found in the documentation provided by Digi [37] will be followed. The first goal is to transmit and receive real-time text messages over the air. Whilst it would be preferable to have two computers talk to each other, there will simply be a monitor on each COM port that the Xbee’s are connected to, to demonstrate the conversation.
The Xbee’s are connected to the Grove Development board, and by USB to the host PC (Figure 3.12).

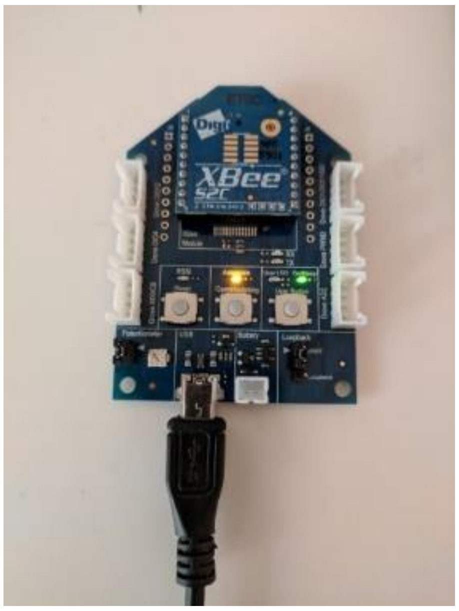
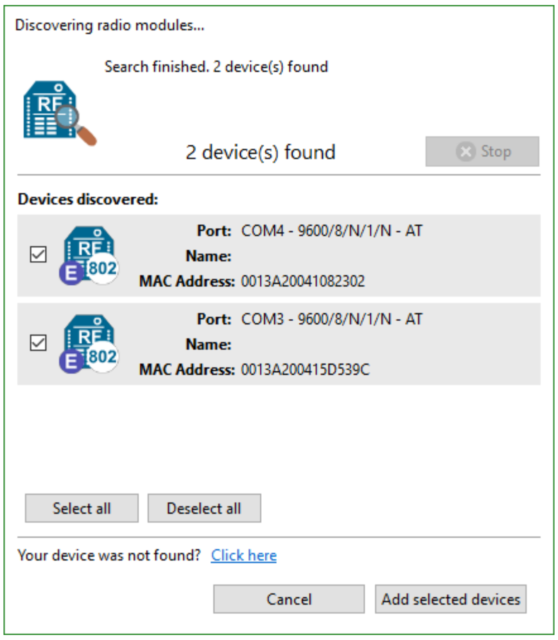


Figure 3.12 - Xbee Module mounted on Grove Development Board, powered and connected to PC.

On the host PC, the XCTU configuration tool is installed and opened. In this tool we can find the radio modules (Figure 3.13 and Figure 3.14) and begin to configure them.


Discovering radio modules...

Search finished. 2 device(s) found

2 device(s) found

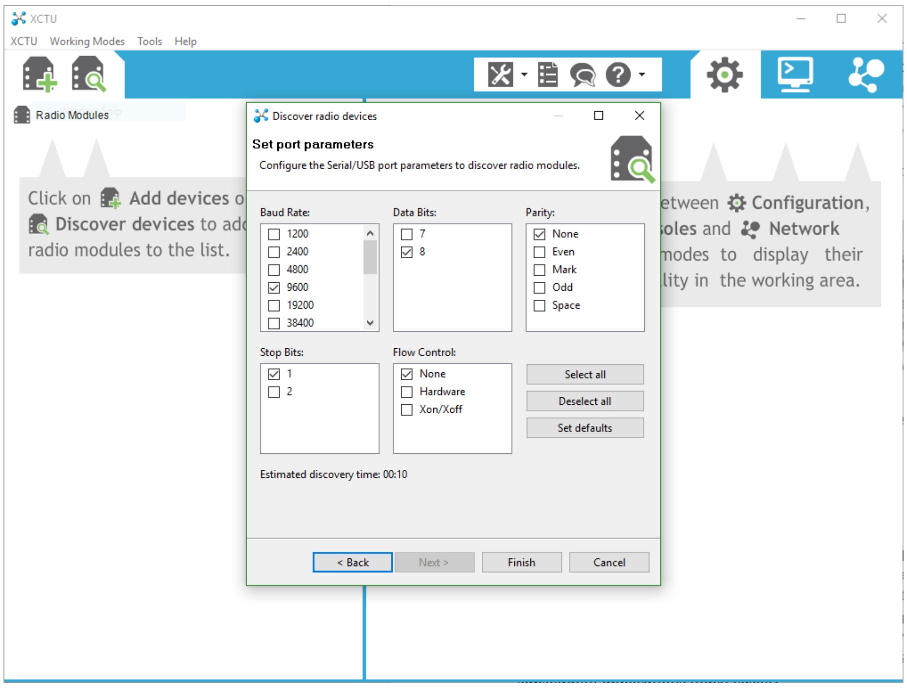
Devices discovered:

☑ **Port:** COM4 - 9600/8/N/1/N - AT
Name:
MAC Address: 0013A20041082302

☑ **Port:** COM3 - 9600/8/N/1/N - AT
Name:
MAC Address: 0013A200415D539C

Your device was not found? Click here

Figure 3.13 - Both Xbee Modules found in XCTU.



Set port parameters

Configure the Serial/USB port parameters to discover radio modules.

Baud Rate: ☐ 1200 ☐ 2400 ☐ 4800 ☑ 9600 ☐ 19200 ☐ 38400

Data Bits: ☐ 7 ☑ 8

Parity: ☑ None ☐ Even ☐ Mark ☐ Odd ☐ Space

Stop Bits: ☑ 1 ☐ 2

Flow Control: ☑ None ☐ Hardware ☐ Xon/Xoff

Estimated discovery time: 00:10

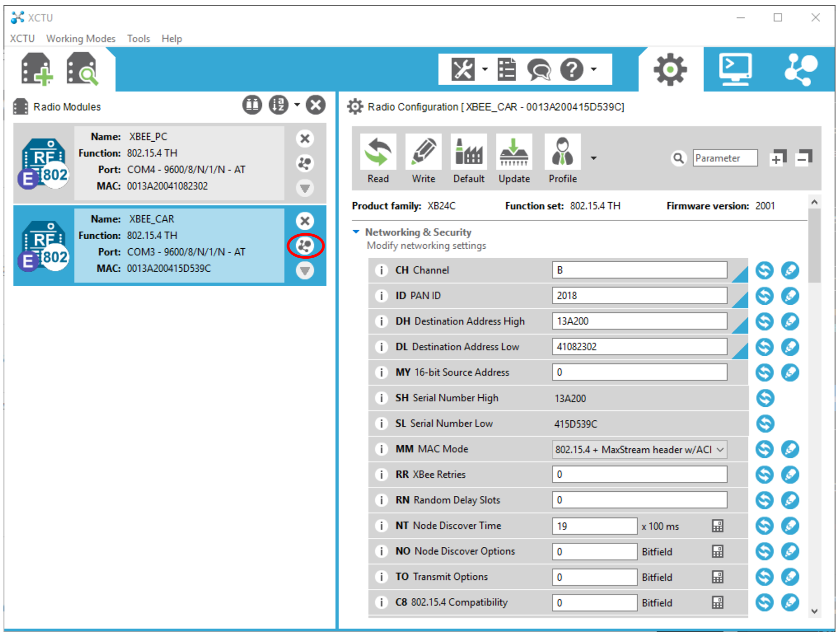
Figure 3.14 - XCTU Radio Discovery Options

To configure these two modules to talk to each other, the destination bytes (DH and DL) must be set to be those of the other modules serial number (SH and SL), which form the modules MAC address. As well as this, the Channel (CH) of each must be the same, as must the ID. Additionally a unique name (NI) will be given to each module to identify them. Table 3.2 shows how these two modules will be setup.

Parameter	**Xbee 1**	**Xbee 2**
CH	B	B
ID	2018	2018
DH	13A200	13A200
DL	415D539C	41082302
NI	XBEE_PC	XBEE_CAR

Table 3.2 - Xbee Configuration options.

The configured radios can be seen in the XCTU window shown in Figure 3.15.



Name: XBEE_PC
Function: 802.15.4 TH
Port: COM4 - 9600/8/N/1/N - AT
MAC: 0013A20041082302

Name: XBEE_CAR
Function: 802.15.4 TH
Port: COM3 - 9600/8/N/1/N - AT
MAC: 0013A200415D539C

Radio Configuration [XBEE_CAR - 0013A200415D539C]

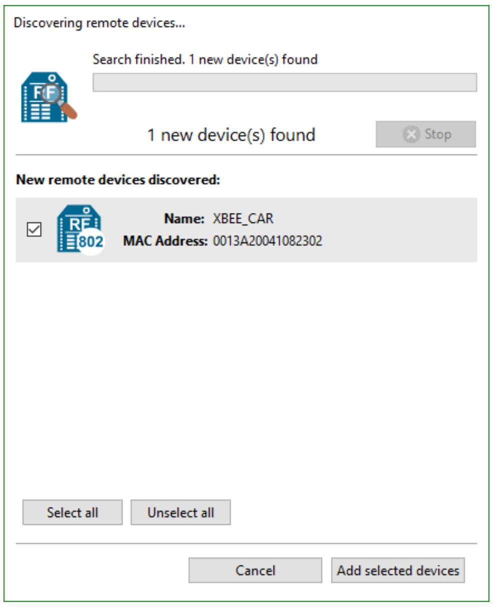
Product family: XB24C Function set: 802.15.4 TH Firmware version: 2001

Networking & Security
Modify networking settings

CH Channel	B	
ID PAN ID	2018	
DH Destination Address High	13A200	
DL Destination Address Low	41082302	
MY 16-bit Source Address	0	
SH Serial Number High	13A200	
SL Serial Number Low	415D539C	
MM MAC Mode	802.15.4 + MaxStream header w/ACI	
RR XBee Retries	0	
RN Random Delay Slots	0	
NT Node Discover Time	19	x 100 ms
NO Node Discover Options	0	Bitfield
TO Transmit Options	0	Bitfield
C8 802.15.4 Compatibility	0	Bitfield

Figure 3.15 - Xbee Modules configured in XCTU.

Pressing the "Discover Radio Nodes button" (highlighted in Figure 3.15) allows us to identify all the present nodes on a single PAN (Personal Area Network).



Discovering remote devices...

Search finished. 1 new device(s) found

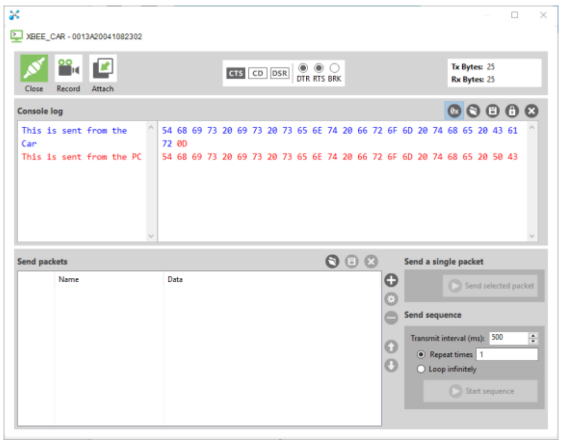
1 new device(s) found

New remote devices discovered:

☑ Name: XBEE_CAR
MAC Address: 0013A20041082302

Figure 3.16 - Car Xbee discovered by the PC antenna.

Figure 3.16 shows how the PC antenna could discover and add the Car antenna as a slave. Now that we can connect the modules together we can test their ability to talk between them. Opening the “Consoles” mode in XCTU allows text and other packets to be sent wirelessly between the devices. In this mode the connections on both modules are opened and text is typed in on one, to be received on the other, demonstrated in Figure 3.17.

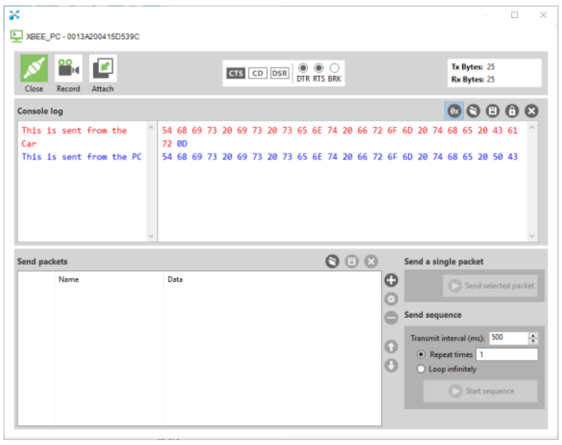


XBEE_CAR - 0013A20041082302

Tx Bytes: 25
Rx Bytes: 25

```
This is sent from the    54 68 69 73 20 69 73 20 73 65 6E 74 20 66 72 6F 6D 20 74 68 65 20 43 61
Car                      72 0D
This is sent from the PC 54 68 69 73 20 69 73 20 73 65 6E 74 20 66 72 6F 6D 20 74 68 65 20 50 43
```

Transmit interval (ms): 500
Repeat times: 1



XBEE_PC - 0013A200415D539C

Tx Bytes: 25
Rx Bytes: 25

```
This is sent from the    54 68 69 73 20 69 73 20 73 65 6E 74 20 66 72 6F 6D 20 74 68 65 20 43 61
Car                      72 0D
This is sent from the PC 54 68 69 73 20 69 73 20 73 65 6E 74 20 66 72 6F 6D 20 74 68 65 20 50 43
```

Transmit interval (ms): 500
Repeat times: 1

Figure 3.17 - Xbee Communication between PC and Car.

The blue text shows data that has been sent from that module, and red text shows data that has been received.

There are two different modes to operate Xbee’s in; Transparent mode and API mode. In transparent mode all data received by the Xbee is transferred to the Serial Port, and all data sent to it from the Serial Port is sent wirelessly. In API mode the data is sent in discrete packets or frames at regular time intervals to collect and send out data. For the purposes of this project Transparent mode will be used, as the Xbee’s are only required to mimic a serial connection over the air.

4 Application Development

4.1 System Design

The application will be designed to work within the top-level system diagram as shown in Figure 4.1. Two wireless transceivers connect the PC to the vehicle, allowing them to communicate. The key PC applications are the one being developed in this project and the calibration and profile management app provided by Tobii. The Tobii Core SDK provides access to this software, not to the root of the device attached by USB to the PC. Whilst this does protect the hardware from accidental internal damage by sending unauthorised commands, it does put a bottleneck on the speed of the software. This means that the speed at which eye movement can be read is only as fast as the tracking interface software allows.

It should be noted that the construction of the robotic vehicle has been used solely as tangible evidence that the application can be used for robotic control. Further details on the construction are omitted for the sake of conciseness. Furthermore, the robot has not been developed with efficiency in mind and whilst there are things to be improved upon, they will not be during the timeframe of this project.

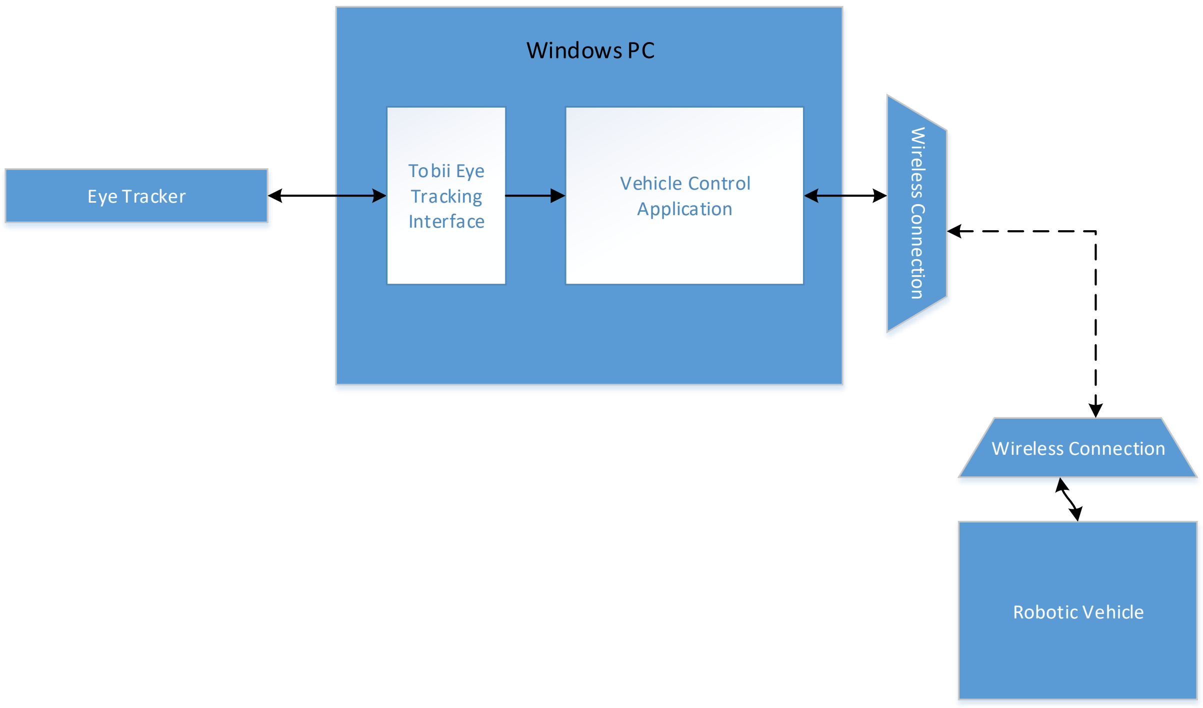


Figure 4.1 - System diagram illustrating the connections and data flow between hardware.

4.2 Application Design

Windows applications designed in Visual Studio are written in C#, a high level, object oriented language, for which Microsoft has developed a vast number of libraries. As this application will use proprietary means of communication and message structure, the classes that handle these will be developed from scratch. The `Communicator` class will use the `SerialPort` object included in the `System.IO.Ports` package as this is a ready-to-use interface for accessing COM ports. Even though the Xbee's will be communicating wirelessly, data will be supplied to the PC Transmitter via a USB-microUSB cable attached to one of the COM ports.

To develop a robust and, more importantly, scalable application design the Model-View-Controller architecture pattern will be implemented to enable parallel development on multiple parts of the software, without having to worry about code becoming unmanageable. In the main application being designed – one that controls a wireless vehicle– having these sections of the program modularised into C# classes makes development of subsequent applications much easier.

Another class to be used will be a `Message` class which forms the message to be sent to the Arduino at the other end of the connection (which itself is handled by the `Communicator` class). The `Message` class can be expanded or otherwise changed as required by another developer to accommodate many other commands. If the handles and return types used by methods remain identical, the internal logic can change as required. An action that may previously have required changes in multiple files now only requires a few lines of code to change and the program will still be stable.
An example of this is:

```
public string PackageMessage (string command)
{
        return command;
}
```

In this method, the same command provided to it is the one that is returned. At first this seems like a lot of unnecessary instructions not to change anything, however, if that function was to change the way it constructed the return string given the command string, all that would need to be changed is the code between the braces and theoretically the rest of the code can go unaltered without compromising the working code.

Figure 4.2 gives the full MVC diagram that applies to both applications. The number of views, communication protocol and interfaces are not specified in the diagram, but this does not affect the core structure of the program, only individual functionalities of the classes. The Model is the core computational centre of the project. Flags raised in the Views are not processed on the UI thread, but are passed by reference to the Model thread. This then coordinates the Controllers to deal with data. Note that the Message and Communications classes never directly interact with anything other than the Vehicle Control class as this ensures the messages sent and received are properly handled, as are the states of the COM ports being used.

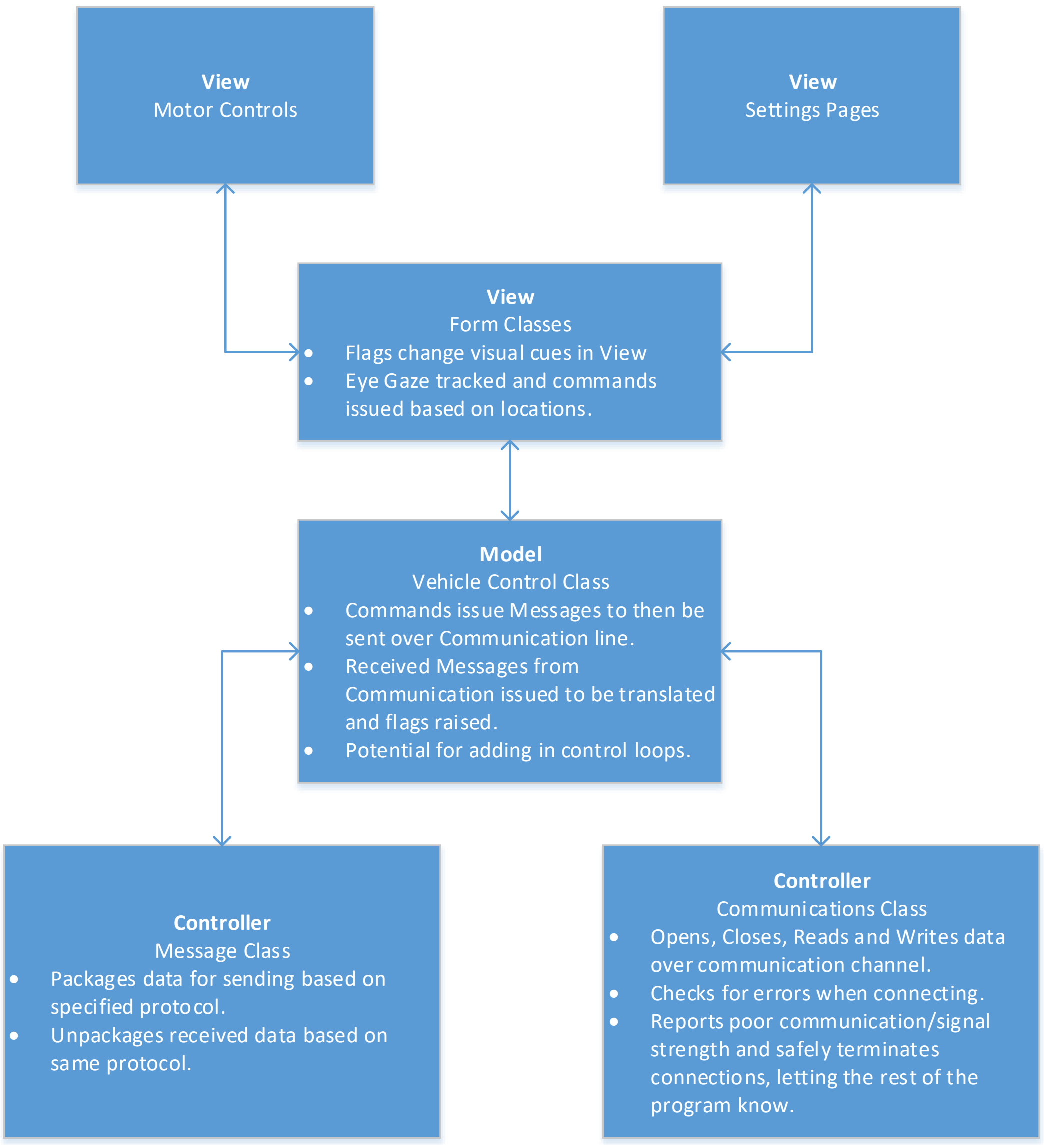

View
Motor Controls
View
Settings Pages
View
Form Classes
Flags change visual cues in View
Eye Gaze tracked and commands issued based on locations.
Model
Vehicle Control Class
Commands issue Messages to then be sent over Communication line.
Received Messages from Communication issued to be translated and flags raised.
Potential for adding in control loops.
Controller
Message Class
Packages data for sending based on specified protocol.
Unpackages received data based on same protocol.
Controller
Communications Class
Opens, Closes, Reads and Writes data over communication channel.
Checks for errors when connecting.
Reports poor communication/signal strength and safely terminates connections, letting the rest of the program know.

Figure 4.2 - Model-View-Controller Diagram

Another function the Model can handle is control loops. Where precise and fast collection and analysis of data from sensors attached to the PC is required, the best place to control it is from the thread that is already coordinating others. This has the added benefit of removing CPU load from the microcontroller handling the movement of motors, as these may have very limited capacity whereas modern CPUs in PCs rarely have issues with moderately intense mathematical algorithms.

4.3 User Interface and Experience

One of the main features, and reasons to use Visual Studio is its "drag and drop" UI development capacity. A designer is able to graphically place components where they need to be and Visual Studio will generate the required code in an XML file. This tool was used to develop the UI before any backend code needed to be written.
The application has been split it several different screens:

4.3.1 Welcome Screen

This opens at the start of every instance of the program. It runs checks to assess which hardware the user has installed on their PC. Specifically, it checks the version of the Tobii EyeX engine (this controls the eye tracking) and instantiates the API controller. There is also the option to connect an external device to a COM port, via a drop-down box. It has been decided that this is to be an optional requirement to enter the main screen in the software as a "demo mode" for the main functionality of the software will be implemented. This will allow a demonstration of the functionality within the app without the need for external hardware. The UI is shown in Figure 4.3

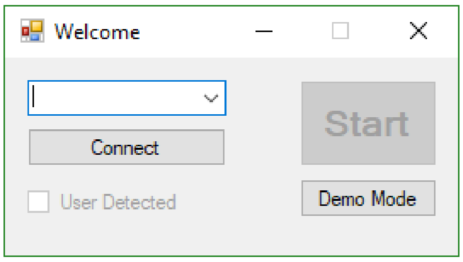
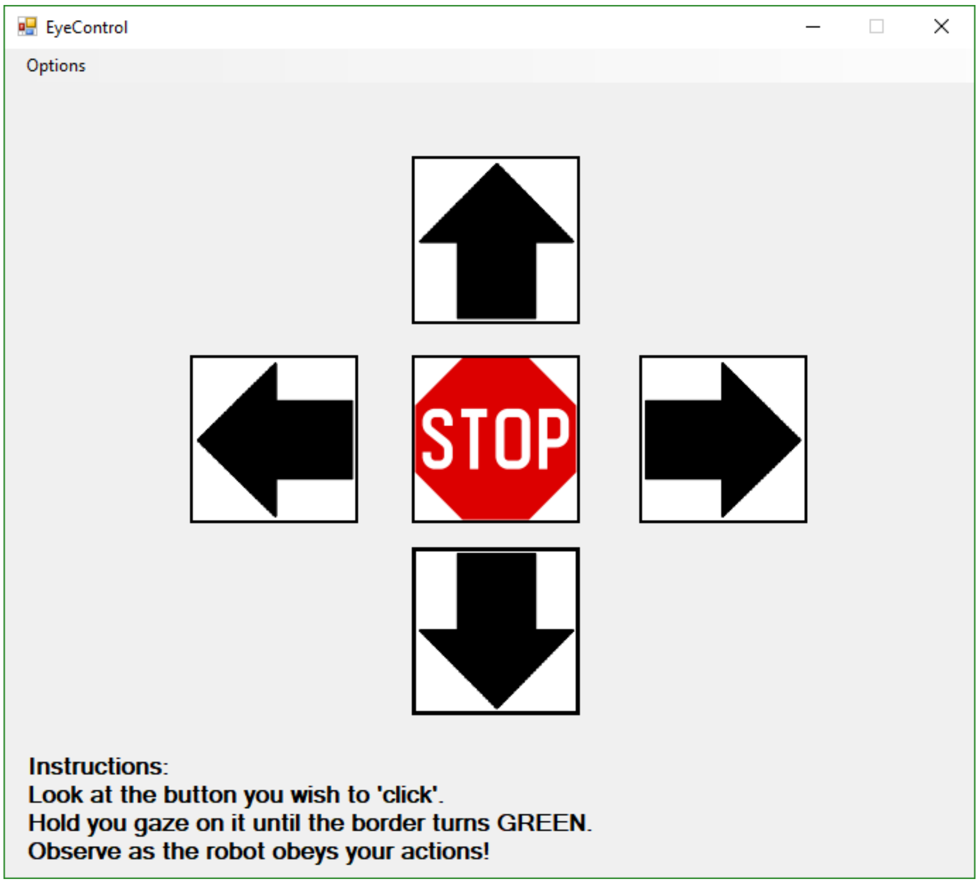


Figure 4.3 - Welcome Screen

4.3.2 Main Screen

This will house the main controls, as well as menu access. For the purposes of the first deliverable of the project (the remote control car) the controls will be; Forward, Reverse, Brake, Turn Left and Turn Right. A short instruction box will tell users what they should do to activate the controls. In normal mode, the user will be able to use their gaze OR the mouse to activate the main controls listed. Staring at a button for the duration of the "dwell time" (set internally to 1500ms) will have the exact same effect as clicking it with the mouse. Each button will change its border colour based on whether the user is looking at it or not or has activated its function. A thick red or green border will make the button stand out to the user and ensure that they know the program has picked up their gaze. The same functionality will occur when the mouse enters, leaves or clicks on the button. Instructions on how to use the application will always be visible to the user in this screen. A menu bar contains access to 3 extra windows, as described below. The UI is shown in Figure 4.4.

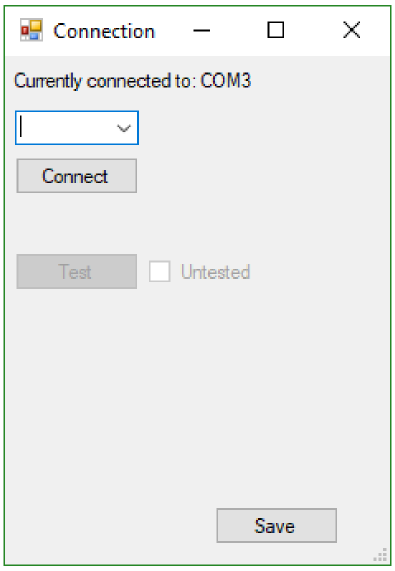


Instructions:
Look at the button you wish to 'click'.
Hold you gaze on it until the border turns GREEN.
Observe as the robot obeys your actions!

Figure 4.4 - Main control window with Eye Tracking

4.3.3 Connection Settings

This setting screen is used to test or change the communication method. The user will be able to change the COM port used and 'ping' the device connected. This makes sure that the connection has been made and that changes can be saved. The UI is shown in Figure 4.5.



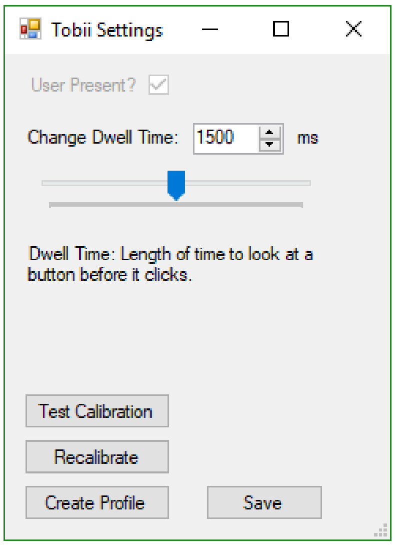
Currently connected to: COM3

☐ Untested

Figure 4.5 - Connection Settings Dialog

4.3.4 Tobii Settings

This is the one of two setting screens that change the characteristics of the main application. The settings here give the user the option to change the Dwell Time, meaning the application can be tailored to the person using it. This display will also show the status of the eye tracker such that the user can tell if it is functioning correctly. This is done by checking if the user is present and representing this in a checkbox. If there is a malfunction with the device, this flag will not be set and the user can be alerted to this. Additionally there are 3 buttons dedicated to external Calibration and User Management. Within the EyeXFramework API there is an `EyeXHost` that handles the connection to the Tobii Windows application. This contains 5 "launch" commands that open various sections of the program, of which the Test Calibration, Recalibrate and Create Profile actions have been chosen for this settings screen. They open the respective programs externally and allow the user to make sure the eye tracking is working as they need it to be. They can also change the profile within Test Calibration if another user wishes to use the application in same session. The UI is shown in Figure 4.6.


Tobii Settings

User Present? ✓

Change Dwell Time: 1500 ms

Dwell Time: Length of time to look at a button before it clicks.

Figure 4.6 - Tobii and Application Settings Dialog

Note: Due to the nature of the way the program has been designed, both this and the Tobii settings screens must have their "save" buttons pressed before settings are changed. Additionally, the changes made will be volatile and not save between application sessions.

4.3.5 Developer Communication

This screen, protected by a password (Figure 4.7), allows a developer access to the communications section of the program, giving them the ability to analyse data sent and received over the selected COM port, as long as the messages have been transferred whilst in this window. During the development process, this window has also been the starting point for the debugging of the communication systems, so will remain present for any future work to take place. The UI is shown in Figure 4.8.

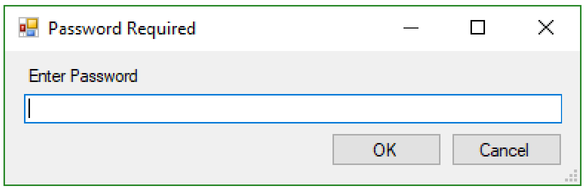
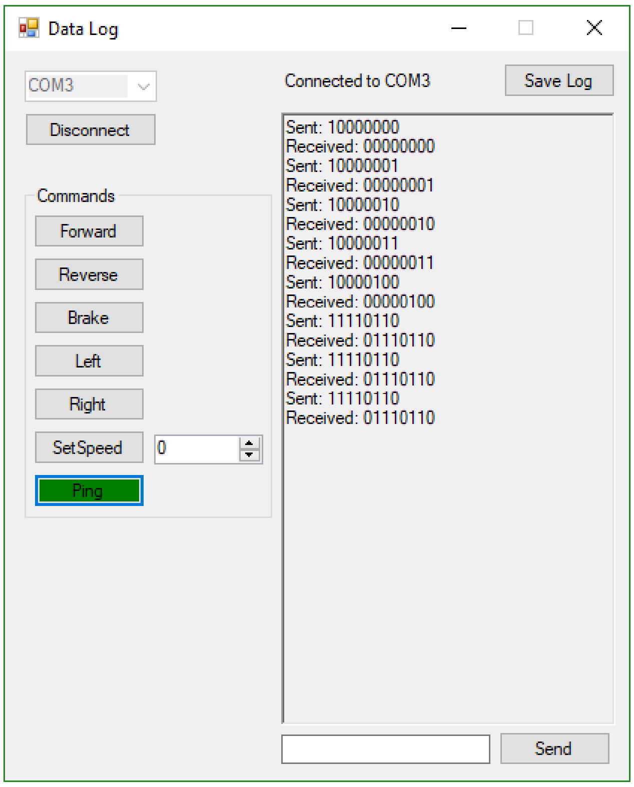


Figure 4.7 - Password Protection Dialog



```
Sent: 10000000
Received: 00000000
Sent: 10000001
Received: 00000001
Sent: 10000010
Received: 00000010
Sent: 10000011
Received: 00000011
Sent: 10000100
Received: 00000100
Sent: 11110110
Received: 01110110
Sent: 11110110
Received: 01110110
Sent: 11110110
Received: 01110110
```

Figure 4.8 - Developer Data Logging Tool

4.4 Communication Protocol and Message Structure

Arduinos support unsigned integers up to 16 bits in length, and unsigned long integers up to 32 bits. This gives a lot of room to send messages that contain both data and meta data. Since the most common communication interfaces are serial interfaces (such as USB and Wi-Fi, which will be used in this project), a shorter message will take less time to reach the unit and be processed. To begin with, the application will utilise 8-bit messages to simplify the app and increase portability. Whilst Arduino microcontrollers are currently being used, a future hardware change may determine that an 8-bit microcontroller would be simpler and cheaper to use with custom hardware. In this case, the message packing and parsing within the Windows application will not need any changes and development time is reduced.
The structure for each message is shown in Table 4.1:

`Direction bit`	`x`	`x`	`x`	`0`	`0`	`0`	`0`
1 = From Windows 0 = From Arduino	Payload			Command			

Table 4.1 - Message Structure between Windows and Arduino

The commands for the robotic vehicle are shown in Table 4.2:

	Command	Payload
Forward	`0000`	n/a (all `0`s)
Backward	`0001`	n/a (all `0`s)
Brake	`0010`	n/a (all `0`s)
Left	`0011`	n/a (all `0`s)
Right	`0100`	n/a (all `0`s)
Set Speed	`0101`	Speed (converted from values 0 – 7)
Ping	`0110`	All `1`s (integer 7)

Table 4.2 - Command List for Robotic Vehicle

During construction of the robotic vehicle (see Hardware Design) it was found that a speed control would not be possible to implement in the timescale allocated to building the car. For this reason, the “Set Speed” command does not perform any action in the car. It has been left in the program as an example of how the message structure allows expansion as hardware is changed, which is important in a scalable solution such as this.

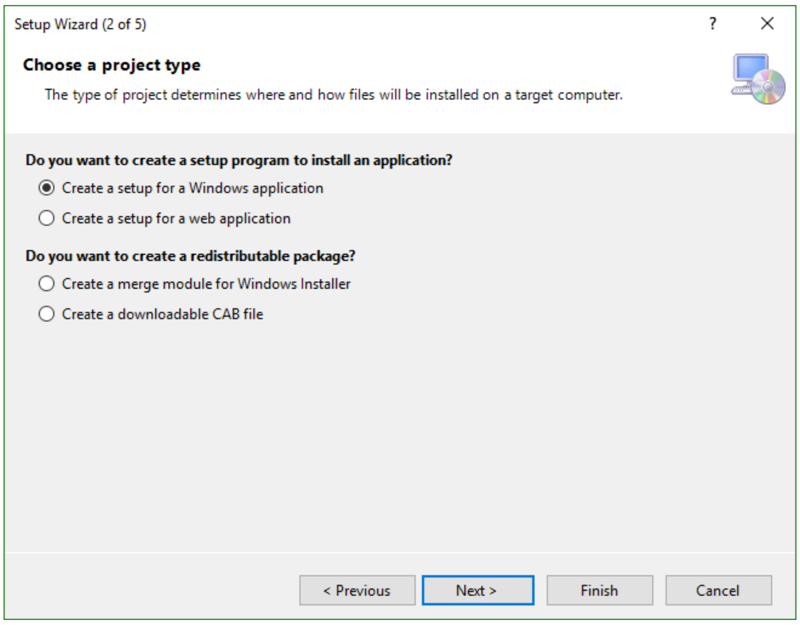
Additionally, the number of implemented commands (7, including “Set Speed”) allows for 9 more commands to be implemented before the 8-bit structure becomes unviable. For a basic vehicle this will not pose a problem, but for more complex devices, or multiple devices connected to the same application, a larger number of messages may be required. At this stage, the message length can be expanded to 16-bit, or even 32-bit if need be, allowing a much larger number of commands to be sent and received as well as the possibility for more meta-data to be transferred.

4.5 Creating a Windows Installer

A core aspect of the deliverables is to make the installation of the application as easy as possible for the user, including all the required drivers and libraries placed into the correct destination folder on the user's PC. In Visual Studio this is made possible through the use of "Setup Wizard" projects included in the same solution file as the application. The projects are not included in the default installation of Visual Studio, so must be installed through a package from the Marketplace, named "Microsoft Visual Studio 2017 Installer Projects". It should also be noted that a different package exists for each version of Visual Studio, and the 2017 version was the right one for the IDE used during development.

To create an installer, the Setup Wizard must be added to the solution. When selected, the wizard walks the developer through the creation of the setup file. In the steps shown on the following pages, the included files and options will be outlined with the reason they have been selected.

The application is designed to work locally on a PC, so the default option on the screen in Figure 4.9 remains selected.


Setup Wizard (2 of 5)

Choose a project type

The type of project determines where and how files will be installed on a target computer.

Do you want to create a setup program to install an application?

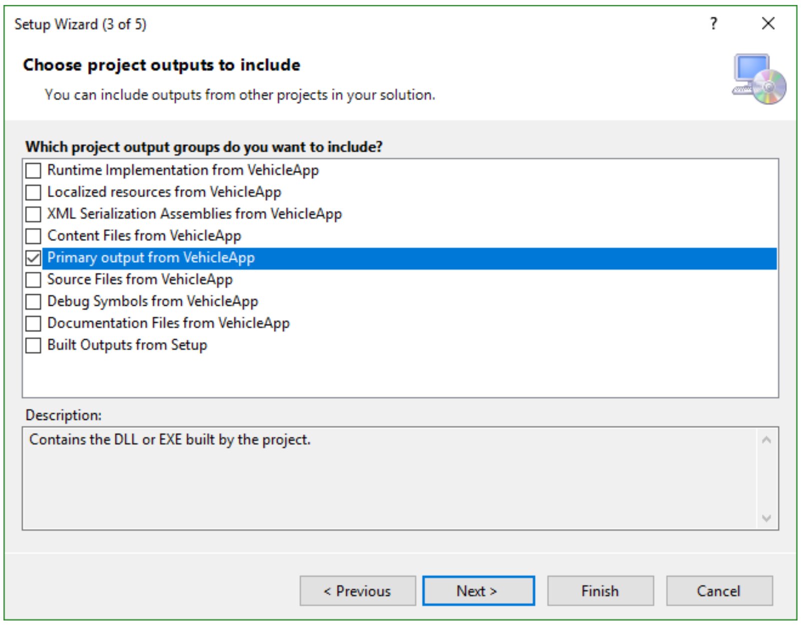
- [✓ Create a setup for a Windows application]
- Create a setup for a web application

Do you want to create a redistributable package?

- Create a merge module for Windows Installer
- Create a downloadable CAB file

Figure 4.9 - Choosing the setup for a locally installed Windows Application.

Whilst there a lot of optional resources to include in Figure 4.10, the only useful item to include in the setup is the .exe output from the VehicleApp project. If any documentation existed in the project, these too would be included by selecting "Documentation Files from VehicleApp".


Setup Wizard (3 of 5)

Choose project outputs to include

You can include outputs from other projects in your solution.

Which project output groups do you want to include?

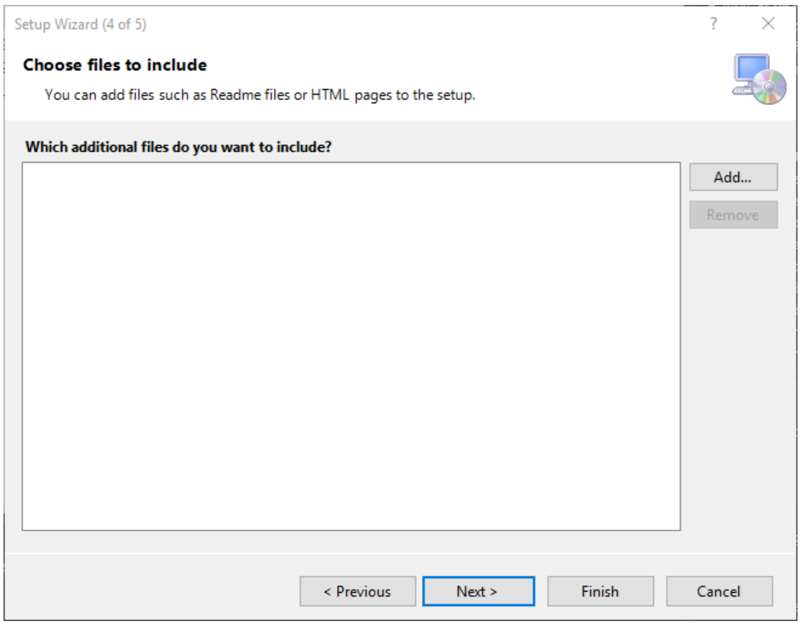
- [ ] Runtime Implementation from VehicleApp
- [ ] Localized resources from VehicleApp
- [ ] XML Serialization Assemblies from VehicleApp
- [ ] Content Files from VehicleApp
- [x] Primary output from VehicleApp
- [ ] Source Files from VehicleApp
- [ ] Debug Symbols from VehicleApp
- [ ] Documentation Files from VehicleApp
- [ ] Built Outputs from Setup

Description:

Contains the DLL or EXE built by the project.

Figure 4.10 - Selected the project outputs to include in the setup files.

Additional instruction and "readme" files can be included in Figure 4.11 if needed to aid the user in installation or usage of the program.


Setup Wizard (4 of 5)

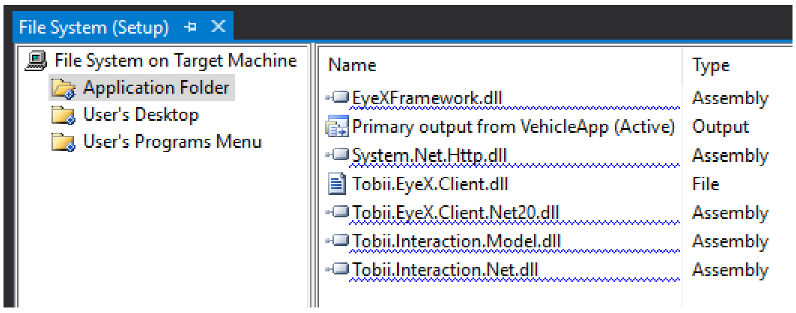
Choose files to include

You can add files such as Readme files or HTML pages to the setup.

Which additional files do you want to include?

Figure 4.11 - Choosing Additional Files to include with the installation.

The final page of the setup wizard provides a summary of the chosen options in case any mistakes were made. When "Finish" is clicked, a page showing the File System of the target machine is shown in Figure 4.12. Three folders, "Application", "User's Desktop" and "User's Program Menu" are available to access and can have the required files and outputs placed in them.


File System (Setup)

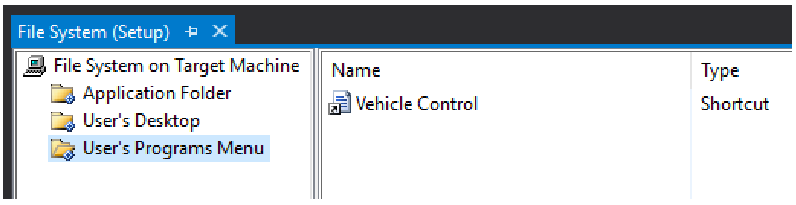
File System on Target Machine
- Application Folder
- User's Desktop
- User's Programs Menu

Name	Type
EyeXFramework.dll	Assembly
Primary output from VehicleApp (Active)	Output
System.Net.Http.dll	Assembly
Tobii.EyeX.Client.dll	File
Tobii.EyeX.Client.Net20.dll	Assembly
Tobii.Interaction.Model.dll	Assembly
Tobii.Interaction.Net.dll	Assembly

Figure 4.12 - Application Folder contents with the File System of the Setup project.

As the Tobii drivers and libraries are not standard on Windows PCs, they must be manually included with every installation. They are copied into the user's application folder at installation to ensure the app runs properly. An issue that came to light was the deprecation of the "Tobii.EyeX.Client.dll" assembly, and this was not included by default in the Tobii NuGet packages. A simple way around this was to copy the file into the Setup project and include it that way. Even though the compiler does not recognise it as an Assembly, it still copies the file in the same way, thus the application works.

The Setup project allows the developer to choose if the user is given a shortcut to the application on the Desktop, Programs Menu, both or neither. In this case, a shortcut has been placed into the user's programs menu (or Start Menu on Windows) which will allow quick access to the program, shown in Figure 4.13.

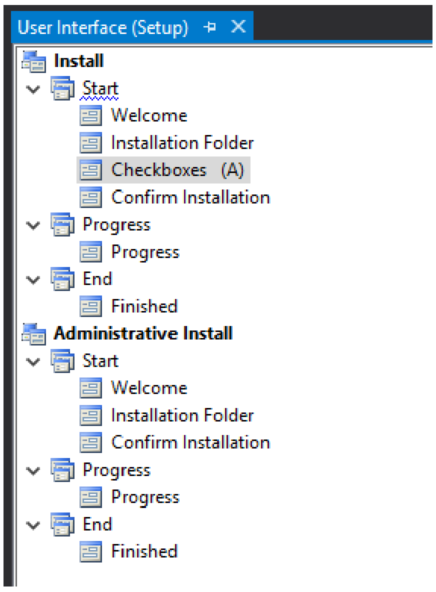

File System (Setup)

File System on Target Machine
- Application Folder
- User's Desktop
- User's Programs Menu

Name	Type
Vehicle Control	Shortcut

Figure 4.13 - User's Programs Menu content with the shortcut to application.

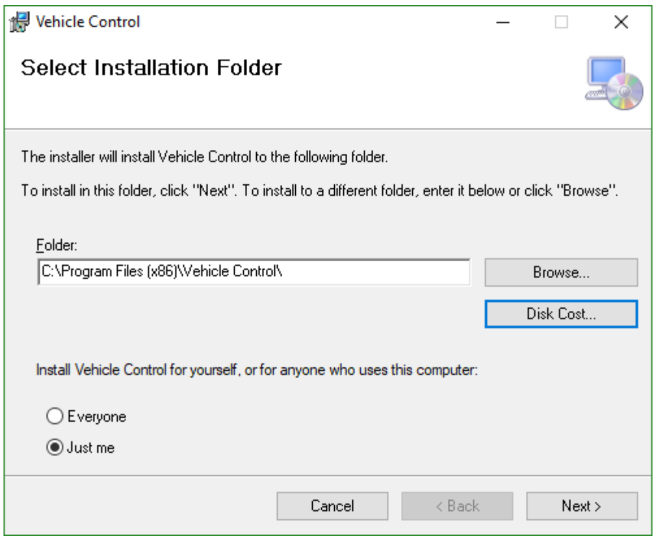
Figure 4.14 shows the path the user will take through the installation wizard to choose settings. By default, a "Welcome", "Installation Folder" and "Confirm Installation" dialog is included, with the "Installation Folder" screen (Figure 4.15) allowing the user to choose where the application is installed. A default path of "C:\Program Files (x86)\Vehicle Control" is entered, though this can be changed.


User Interface (Setup)

- Install
  - Start
    - Welcome
    - Installation Folder
    - Checkboxes (A)
    - Confirm Installation
  - Progress
    - Progress
  - End
    - Finished
- Administrative Install
  - Start
    - Welcome
    - Installation Folder
    - Confirm Installation
  - Progress
    - Progress
  - End
    - Finished

Figure 4.14 - Program flow for the setup executable.

A custom dialog can also be inserted, in this case “Checkboxes (A)” has been added. In this window, the user can choose where the shortcut to the application goes, if one is required at all. The text and names of the checkboxes within the dialog are all set in the Properties box in Figure 4.16. Another property named “Checkbox1Property” allows a variable name to be set for the checkbox, which when ticked sets this variable to 1. In the case shown, two variables “STARTMENU” and “DESKTOP” correspond to creating the shortcut in the Start Menu and on the Desktop respectively.


Vehicle Control

Select Installation Folder

The installer will install Vehicle Control to the following folder.

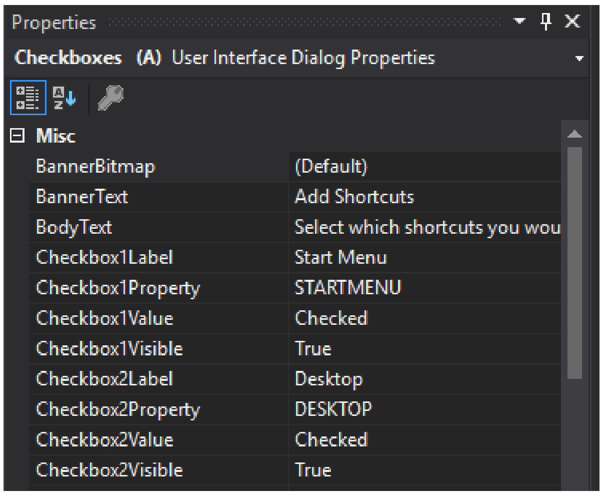
To install in this folder, click "Next". To install to a different folder, enter it below or click "Browse".

Folder:
C:\Program Files (x86)\Vehicle Control\

Install Vehicle Control for yourself, or for anyone who uses this computer:

○ Everyone
● Just me

Figure 4.15 - "Installation Folder" screen in completed installer.



Properties

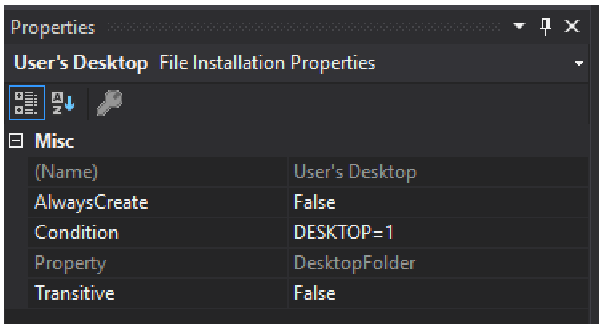
Checkboxes **(A)** User Interface Dialog Properties

Misc

BannerBitmap	(Default)
BannerText	Add Shortcuts
BodyText	Select which shortcuts you wou
Checkbox1Label	Start Menu
Checkbox1Property	STARTMENU
Checkbox1Value	Checked
Checkbox1Visible	True
Checkbox2Label	Desktop
Checkbox2Property	DESKTOP
Checkbox2Value	Checked
Checkbox2Visible	True

Figure 4.16 - Properties box for Checkbox dialog.

To link these variables to the folders, the “Condition” property of the Folder within the File System is set to “STARTMENU=1” or “DESKTOP=1”. This is demonstrated in Figure 4.17.



Properties

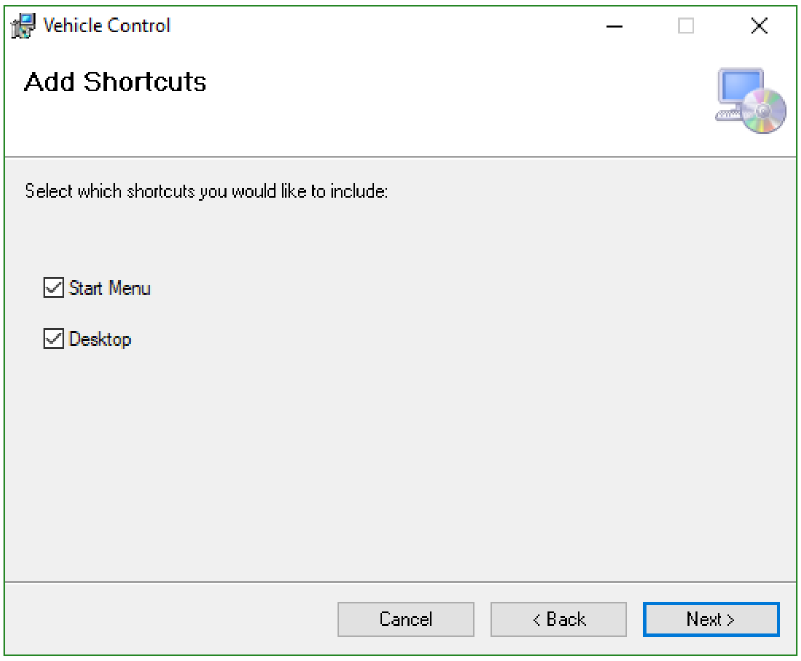
User's Desktop File Installation Properties

Misc

(Name)	User's Desktop
AlwaysCreate	False
Condition	DESKTOP=1
Property	DesktopFolder
Transitive	False

Figure 4.17 - Desktop folder condition set to create only when the DESKTOP variable is set to 1.

The visual representation of this dialog is shown in Figure 4.18.



Add Shortcuts

Select which shortcuts you would like to include:

☑ Start Menu

☑ Desktop

Figure 4.18 - "Add Shortcuts" dialog in finished installer, representing "Checkboxes (A)".

4.6 Arduino Controller

A sketch for the Arduino was developed to read in data from the serial port, either from the wired USB port or the Arduino Wireless shield. The pins on the board where the data is read from and sent to are common to both methods of communication, and a small switch (shown in red) on the shield allows selection between the Xbee data going to the microcontroller or straight through the USB connection. The layout of the hardware used is shown in Figure 4.19.

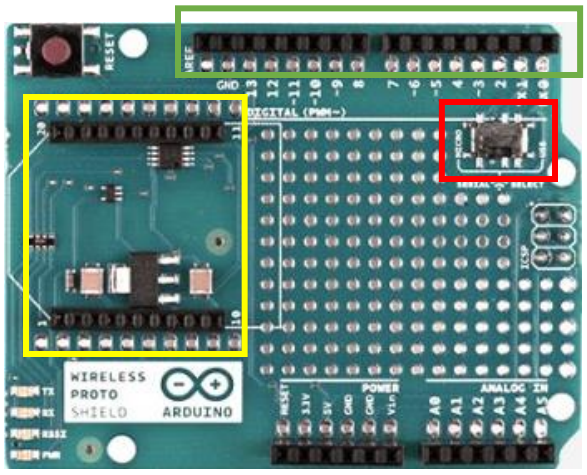


Figure 4.19 - Arduino Wireless Proto Shield [44]

Other areas of interest are the Xbee mounting pins (in yellow) where the Xbee transceiver is connected. The board connects the power and data pins to those on the rest of the board such that it acts as a wired serial connection. Additionally, the pins in green pass through to the main Arduino Uno underneath and allow connection of normal digital input or outputs from pins 2-13. The flow diagram for the Arduino sketch is shown in Figure 4.20.

The main loop of the program checks the structure of the message in the proprietary format described previously. It then parses this data when it knows it has received the full command. From the data parsed it can then determine which functions to carry out. In the case of controlling a vehicle this is a case of turning relays on or off through digital outputs on the board.

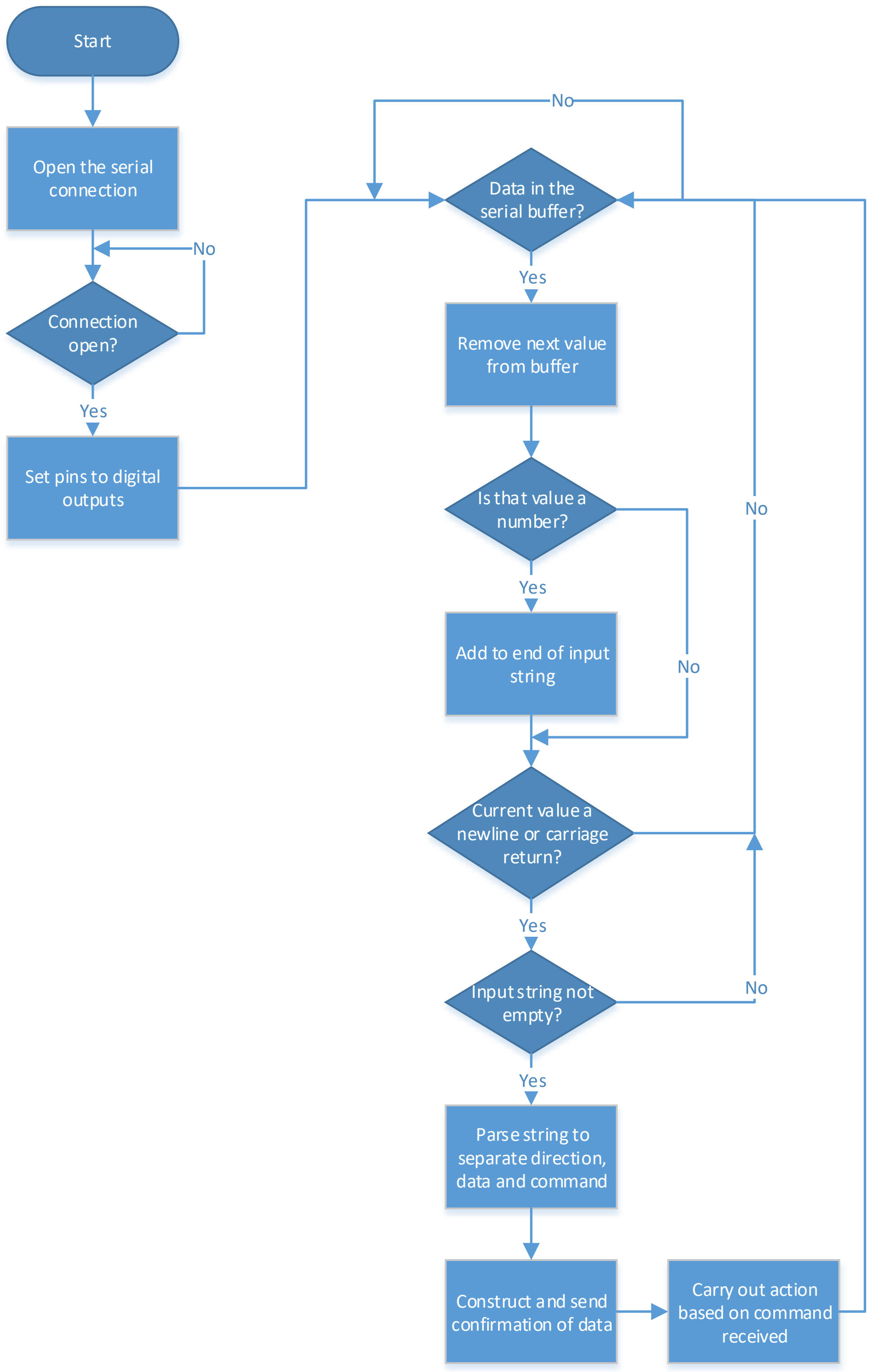

Start
Open the serial connection
No
Connection open?
Yes
Set pins to digital outputs
No
Data in the serial buffer?
Yes
Remove next value from buffer
Is that value a number?
Yes
Add to end of input string
No
No
Current value a newline or carriage return?
Yes
Input string not empty?
No
Yes
Parse string to separate direction, data and command
Construct and send confirmation of data
Carry out action based on command received

Figure 4.20 - Arduino Main Program Flow.

4.7 Testing

As is good practice in all software development projects, a comprehensive list of tests should be written and carried out to assess the actions that a typical user may perform. Whilst it is possible to have Visual Studio carry out these tests automatically with CodedUI, this suite of tools has been known to not perform correctly in all cases based on previous experience. Therefore, testing will be carried out manually as defined in these tests and assertions.

4.7.1 Test Cases

Test cases are designed to either give a 'Pass' or 'Fail' outcome. Each screen in the application will have its own set of tests defined here.

Welcome Screen

Attach the Xbee Transceiver to the PC and power on the Arduino.

1	Turn on the eye tracker. Is the "user detected" box checked?
2	Move away from the eye tracker's range. Is the "user detected" box unchecked?
3	Click on the dropdown box. Does it fill with the current COM ports connected to the PC? Double check this in Device Manager > COM Ports.
4	Select a port that is not the Xbee transceiver and press connect. Does the "Start" button stay unclickable?
5	Select the port that is the Xbee transceiver and press connect. Does the "Start" button become clickable?
6	Click "Start". Does the Eye Control window open and Welcome close?
7	Restart the app. Click "Demo Mode". Does the Welcome window close and Eye Control window open with "Demo Mode Active" shown in the window?

Eye Control

Start in normal (non-demo) mode. Eye tracker must be on.

1	Look at each arrow button and the stop button. Does the border turn red at first glance?
2	Stare at each arrow button and the stop button. Does the border turn red then green after about 1.5s?
3	Stare at each arrow button and the stop button. Does the vehicle perform the relevant action?
4	Move the mouse into each arrow and the stop button. Does the border turn red?
5	Click the mouse on each arrow and the stop button. Does the border turn green?
6	Click the mouse on each arrow and the stop button. Does the vehicle perform the relevant action?

Start in demo mode. Eye tracker must be on.
Carry out Eye Control tests 1, 2, 4, 5.

1	Click "Exit Demo Mode". Does the Welcome screen open again?
2	Stare at each arrow button and the stop button. The vehicle should not perform the relevant action.
3	Click the mouse on each arrow and the stop button. The vehicle should not perform the relevant action.

Developer
Start from the non-demo Eye Control screen.

1	Click Options > Developer. Does a password screen open?
2	Enter “wrong password” and press OK. Does a message pop up saying “Invalid Password”?
3	Press Cancel. Does the password window close? If so this passes. Open the developer window again.
4	Enter “uondev” and press OK. Does the “Data Log” screen open?
5	Click each of the command buttons. Does the vehicle perform the action?
6	When clicking each command, does the log window populate with the binary representations of each sent and received message?
7	Click “Save Log”. Does a file save dialog appear?
8	Save the file and open it, Does the contents match what is in the log?
9	Type a message into the text box and press “Send”. Does it appear in the log?
10	Enter a known command (e.g. 128) and press “Send”. Does the vehicle perform the appropriate action?
11	Press Disconnect. Do the buttons become inactive?
12	Select a new COM port and click Connect. Do the buttons become active?

Connection Settings
Start from the non-demo Eye Control screen.

1	Click Options > Connection Settings. Does the connection screen open?
2	Choose a new connection from the drop down menu and click “Connect”. Does the “Currently connected” label change to the new COM port selected?
3	Click the “Test” button. The checkbox should not tick.
4	Connect back to the transceiver COM port. Click the “Test” button. The checkbox should now tick.
5	Click “Save”. Does the window close and return control to the Eye Control window?

Application Settings
Start from any Eye Control screen.

1	Click Options >Application Settings. Does the Tobii settings screen open?
2	Move in and out of the eye tracker’s field of view. Does the “user present” box tick and untick respectively?
3	Type a new value into the dwell time box. Does the slider change position to match?
4	Slide the slider to a new value. Does the dwell time value change accordingly?
5	Type 5000 into the dwell time box. Does the slider move to the right end of the scale?
6	Slide the slider around. Is the highest value it can reach 5000?
7	Click “Test Calibration”. Are you taken to the Tobii Eyetracking test screen?
8	Click “Recalibrate”. Are you taken to the Tobii Eyetracking recalibration screen?
9	Click “Create”. Are you taken to the Tobii Eyetracking calibration creation screen?
10	Click “Save”. Are you returned to the Eye Control screen?
11	Hold your gaze on a button. Does it take around 5s to “click”?

If all tests pass, the application works as intended.

4.8 Hardware Design

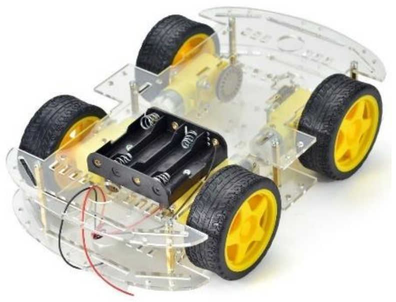


Figure 4.21 - Vehicle Chassis [46]

A basic 4-wheel vehicle will be used as the car chassis with a motor on each individual wheel, as demonstrated in Figure 4.21. The left and right-side motors will be controlled by one of two 5V relay modules, each of which is powered by an external battery pack and controlled by the Arduino. The schematic of the control circuitry is shown in Figure 4.22, though the entire circuit up to the motor is available prebuilt on a single circuit board, duplicated to allow for forward and reverse motor control.

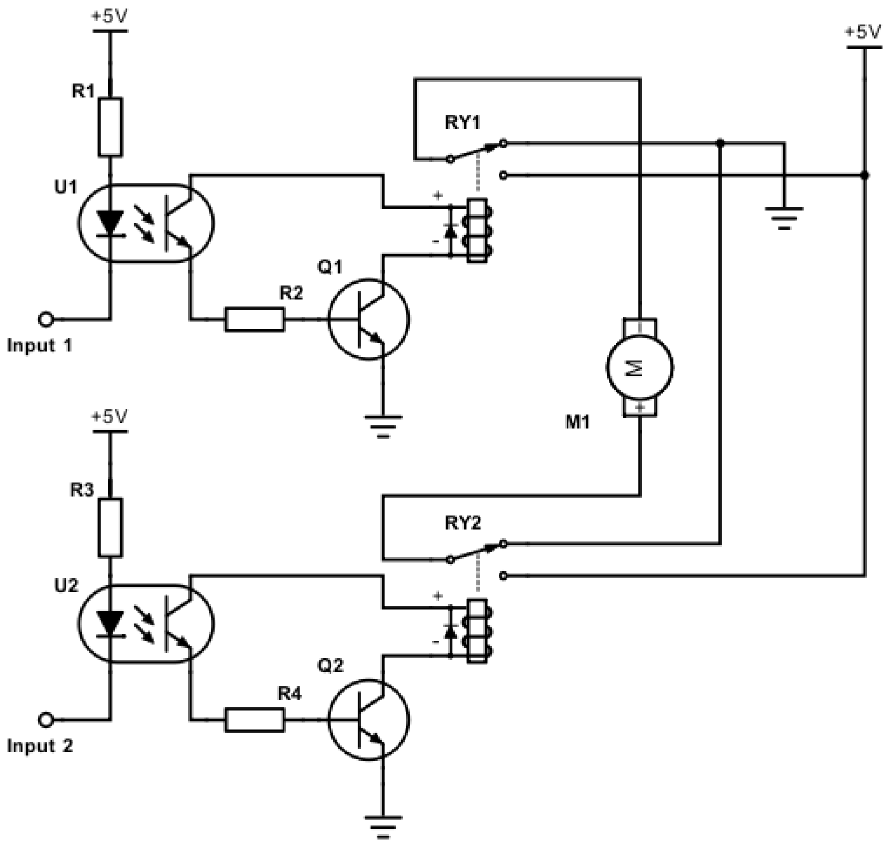


Figure 4.22 - Relay control circuitry for each motor.

The optoisolators U1 and U2 provide a safe transmission of the signal from the Arduino without directly exposing it to the potential high voltages present in the Relay.
The input port connects to an Arduino pin and either goes high or low to turn the relay for that input on or off. This means that only 4 GPIO pins are required for the motor control and the others can be used for communications.

The power supply for the Motors is a 5V supply from the same battery pack powering the Arduino. In lab tests, driving 4 motors in the same direction, carrying the weight of the rest of the package, drew 1A total from the power supply. As the battery pack being used – an Anker 13,000mAh battery with 2 USB ports – can supply up to 3A total over both USB ports, this is deemed suitable for use with this circuit.

The direction of the motors can be controlled by switching both relays either side of two motors in parallel. Switching U1 high and U2 low will make both motors spin in one direction, and inverting the logic makes them spin in the opposite direction.
As shown in Figure 4.23 the Normally Open (NO) port is connected to 5V, and the Normally Closed (NC) port is connected to 0V. When R1 is switched, NO connects to COM, providing a route from 5V, through both motors to Ground. When R2 is switched (and R1 switched back), the current flows the opposite way through the motors, spinning them in the opposite direction.

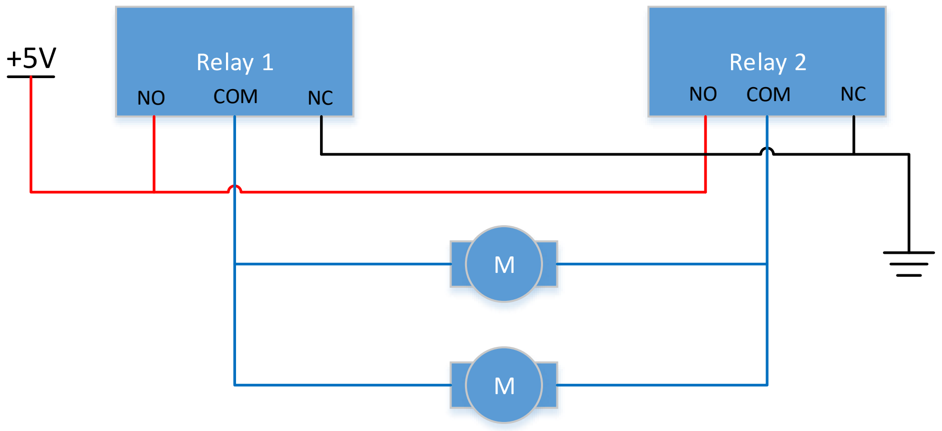


Figure 4.23 - Motor connections between Common lines of relays.

The relay modules, each consisting of two relays mounted onto a logic control board, will be able to control two motors each, in the forward or reverse directions together. This means that turning will be achieved through "tank steering", where to turn left, the left wheels will reverse and the right wheels will go forward, turning the vehicle on the spot. This is demonstrated in Figure 4.24.

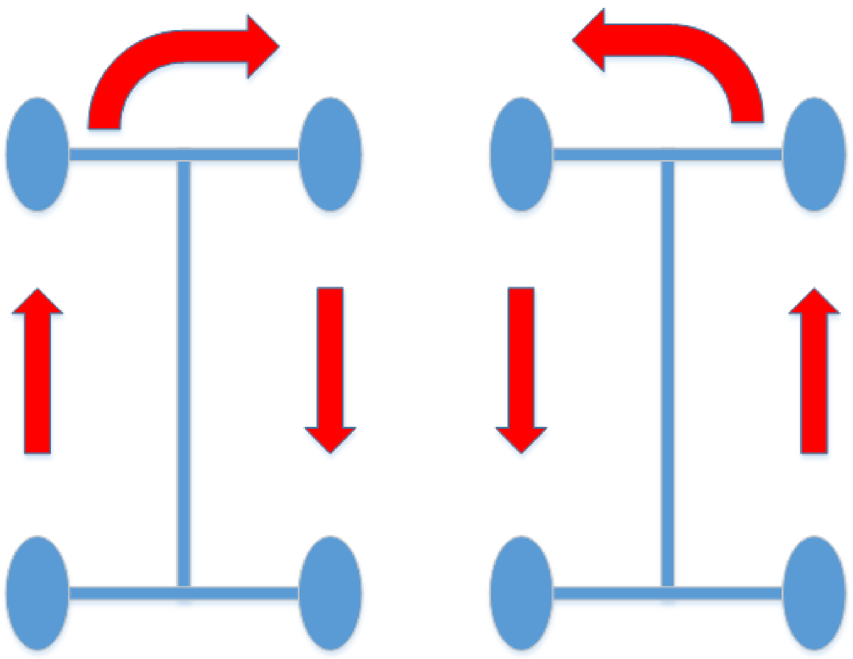


Figure 4.24 - Demonstration of how tank steering works.

To enable free-roaming of the car, a Xbee shield will be mounted on top of the Arduino, and an Xbee 802.15.4 module will be connected to that. The Xbee's will be setup prior to integration with the vehicle, and one device will be connected to the controlling PC as an antenna.

Documents downloaded from Digi's website [37] have been used to learn how the Xbee software and hardware works, and how to integrate the right communication settings into a C# app. Within the scope of this project, the application will only require connection to a single Xbee device, so the settings (discussed in Xbee Setup and Testing) will be hardcoded into both Rx/Tx module.

5 Achievements

At the end of the practical element of the project, the first 3 milestones were reached along with the first deliverable. The final application is able to connect to and communicate with an Arduino over a wireless connection, and the buttons that control the movement of the vehicle can be activated by resting the users gaze on them for the desired dwell time. The full setup is shown in Figure 5.1.

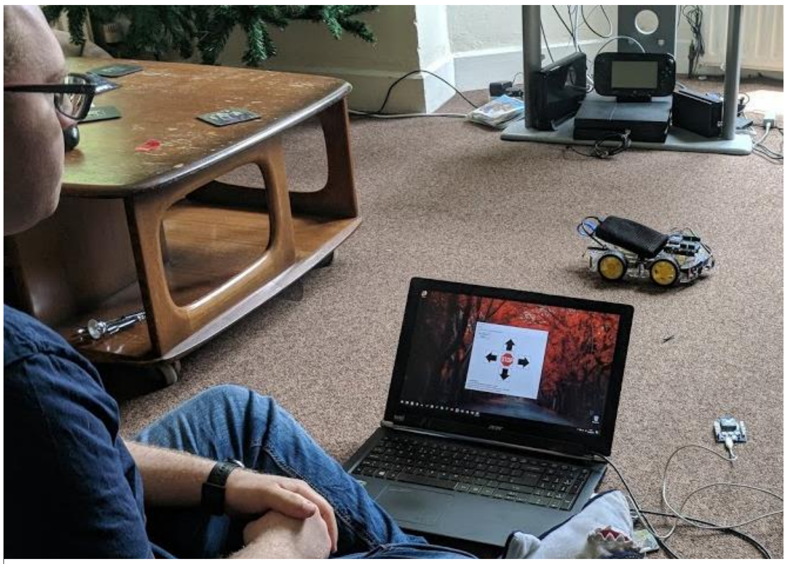


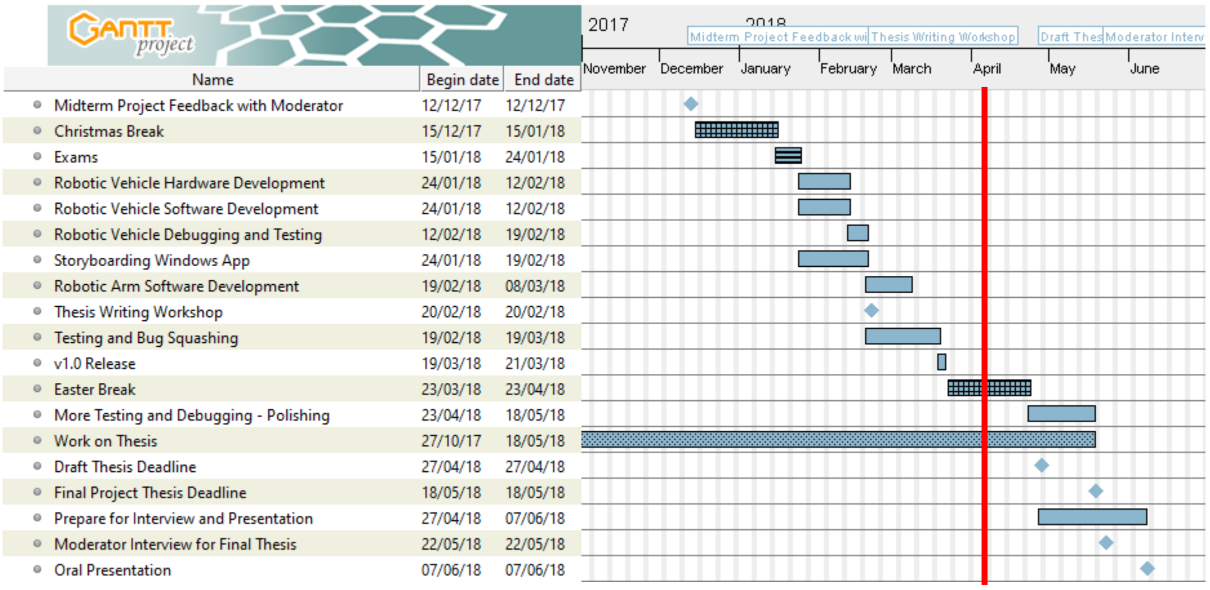
Figure 5.1 - Miles with the fully setup system

Given more time the application could quite easily be adapted to accommodate different hardware such as implementing forward kinematics to control a uArm Metal Robotic Arm. The "Communicator" and "Message" classes were designed as reusable objects that can package and send any message needed. All that is left to design is the commands needed for the arm and the algorithm for calculating the motor movement required. Whilst the algorithm could take a longer time to develop, the theory for control is well established and can be researched.

The motor movement would include extension to reach an object, then grasping or letting go from it. The uArm uses a "sucker" to pick up objects, but the concept would be very similar for a traditional "claw" arm. Whilst most of the Windows application code would stay the same, the control code on the Arduino will be all new. Parsing of the commands onboard the Arduino will stay the same, however.

The revised Gantt chart for the project is shown in Figure 5.2. The red line shows the date at which the practical work has finished. This chart was revised on the 11th December 2017 based on the progress done up to completing the first milestone – the LED Gaze control app. As can be seen, the application was completed later than the v1.0 release was expected, but there is still debugging and testing time left over before the final deadlines.

Overall the project has met its major goals and as a demonstrable exhibit of eye control of robotics it provides a solid starting point for future developments, and has potential to be carried on being developed until a consumer release is viable.



Name	Begin date	End date
Midterm Project Feedback with Moderator	12/12/17	12/12/17
Christmas Break	15/12/17	15/01/18
Exams	15/01/18	24/01/18
Robotic Vehicle Hardware Development	24/01/18	12/02/18
Robotic Vehicle Software Development	24/01/18	12/02/18
Robotic Vehicle Debugging and Testing	12/02/18	19/02/18
Storyboarding Windows App	24/01/18	19/02/18
Robotic Arm Software Development	19/02/18	08/03/18
Thesis Writing Workshop	20/02/18	20/02/18
Testing and Bug Squashing	19/02/18	19/03/18
v1.0 Release	19/03/18	21/03/18
Easter Break	23/03/18	23/04/18
More Testing and Debugging - Polishing	23/04/18	18/05/18
Work on Thesis	27/10/17	18/05/18
Draft Thesis Deadline	27/04/18	27/04/18
Final Project Thesis Deadline	18/05/18	18/05/18
Prepare for Interview and Presentation	27/04/18	07/06/18
Moderator Interview for Final Thesis	22/05/18	22/05/18
Oral Presentation	07/06/18	07/06/18

Figure 5.2 - Revised Gantt Chart (11/12/2017)

Key:

- ♦ Deadline
- ▦ Holiday
- ▩ Document Writing
- ■ Practical Work

6 Conclusions

At the end of the project, a lot had been learned about the practicalities of implementing eye tracking into windows applications. One of the most important things has been giving the user the options to change the dwell time when implementing “clicks”, as this is one of the key factors that changes between people, and even between actions. Likewise, there is still a lot to be learnt about how to create intuitive and visually appealing designs.

This is important as the audience spans a large age range, from young children to those older, most of whom have little to no experience with these kind of applications and may be reluctant to learn complex systems. The tools are present within the Tobii SDKs, however lacking experience in WPF programming created a hurdle that would not have been overcome in the time allocated for this project.

Another key output from this project has been the base that can expanded, reused and developed further to a more commercially viable solution to those in need of robotic assistance. In the same survey as posted to the Tobii Dynavox Facebook group (see Introduction), 35.7% of respondents stated they would expect to pay more than £1000 for such a system, with one stating they would expect to pay between $7000-$10000.

This is unfortunately a standard price point expected by many people targeted by the accessible technology industry. Many devices sold by Tobii Dynavox use the same circuitry as the Tobii 4C Eye Tracker, but charge roughly 10x more because the people purchasing the products are disability charities, who have the money to pay for it.

To sell this software, along with the robotics and eye tracker – both of which are commercially available for less than £600 total – at a price point below £1000 would open up opportunities for those with tighter budgets to live easier lives. £1000 is the minimum someone may have to spend for just the Tobii Dynavox branded eye tracker.

Understandably, there would be some cost in software development and testing, as well as safety approval for any custom hardware, but this can be spread thinly if initial development takes place during personal free time. Working with current developers on the forums and learning from other online courses there is a good chance that a professionally designed and tested application can be developed for the disability community.

7 References

[1] “Prevalence and Incidence of Rett's Syndrome,” [Online]. Available: http://www.rightdiagnosis.com/r/retts_syndrome/prevalence.htm#prevalence_intro. [Accessed 14 February 2018].

[2] S. e. a. Zarei, “A comprehensive review of amyotrophic lateral sclerosis,” *Surg Neurol Int,* vol. 6, no. 171, 2015.

[3] W. C. Davidson-Hoult A, *An Introduction to AAC for People with Rett Syndrome and other Complex Communication Needs,* Rett UK, 2017.

[4] M. Bardon, “Eye Tracking Project Questionnaire,” [Online]. Available: https://docs.google.com/forms/d/e/1FAIpQLSeOADkaCj2gGL9P2C8_UsF6jBxyv-q-eIZ6zLEEL_AJRGIi9A/viewform?usp=sf_link. [Accessed 9 May 2018].

[5] “Tobii Dynavox Community,” [Online]. Available: https://www.facebook.com/groups/tobiidynavoxcommunity/. [Accessed 9 May 2018].

[6] E. B. Huey, The psychology and pedagogy of reading, New York: The Macmillan Company, 1908.

[7] “Eye Tracking Through History,” EyeSee Research, [Online]. Available: http://eyesee-research.com/blog/eye-tracking-history/. [Accessed 10 April 2018].

[8] R. J. K. Jacob and K. S. Karn, “Eye Tracking in Human–Computer Interaction and Usability Research,” Elsevier Science BV, Oxford, 2003.

[9] D. A. Robinson, “A method of measuring eye movement using a scleral search coil in a magnetic field,” *IEEE Transactions on Bio-Medical Electronics,* no. 4, pp. 137-145, 1963.

[10] H. L. Ramkumar, “Electrooculogram,” EyeWiki, 17 October 2015. [Online]. Available: http://eyewiki.aao.org/Electrooculogram. [Accessed 10 April 2018].

[11] “How do Tobii Eye Trackers work?,” [Online]. Available: https://www.tobiipro.com/learn-and-support/learn/eye-tracking-essentials/how-do-tobii-eye-trackers-work/. [Accessed 28 October 2017].

[12] H. Crane and C. Steele, “Generation-V dual-Purkinje-image eyetracker,” *Applied Optics,* vol. 24, no. 4, pp. 527-537, 1985.

[13] D. Witzner Hansen and Q. Ji, “In the Eye of the Beholder: A Survey of Models for Eyes and Gaze,” *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 32, no. 3, pp. 478-500, 2010.

[14] “What is Eye Tracking and How does it work?,” iMotions, 12 January 2016. [Online]. Available: https://imotions.com/blog/eye-tracking-work/. [Accessed 10 April 2018].

[15] “Dark and Bright Pupil Tracking,” Tobii Pro, [Online]. Available: https://www.tobiipro.com/learn-and-support/learn/eye-tracking-essentials/what-is-dark-and-bright-pupil-tracking/. [Accessed 15 October 2017].

[16] “Eye tracking reseach in psychology and neuroscience,” Tobii Pro, [Online]. Available: https://www.tobiipro.com/fields-of-use/psychology-and-neuroscience/. [Accessed 8 April 2018].

[17] “Tobii EyeX Product Specification,” Tobii Gaming, [Online]. Available: https://tobiigaming.com/product/tobii-eyex/. [Accessed 7 October 2017].

[18] “Eye Tracker 4C Product Page,” Tobii, [Online]. Available: https://tobiigaming.com/eye-tracker-4c/. [Accessed 7 October 2017].

[19] “PCEye Mini Product Page,” Tobii Dynavox, [Online]. Available: https://www.tobiidynavox.com/en-GB/devices/Eye-Gaze-Devices/campaign-pceye-mini/`. [Accessed 7 October 2017].

[20] “Arduino Uno Rev3 SMD,” Arduino, [Online]. Available: https://store.arduino.cc/arduino-uno-smd-rev3. [Accessed 29 April 2018].

[21] “Arduino - Compare,” Arduino, [Online]. Available: https://www.arduino.cc/en/Products/Compare. [Accessed 29 April 2018].

[22] “Arduino Mega 2560 Rev3,” Arduino, [Online]. Available: https://store.arduino.cc/arduino-mega-2560-rev3. [Accessed 29 April 2018].

[23] “SparkFun Xbee Shield,” SparkFun, [Online]. Available: https://www.sparkfun.com/products/12847. [Accessed 29 April 2018].

[24] “NRF24L01 2.4GHz Wireless Radio Module,” Hobby Components, [Online]. Available: http://hobbycomponents.com/wired-wireless/156-nrf24l01-24ghz-wireless-radio-transceiver-module. [Accessed 29 April 2018].

[25] 1. Core, “HC-12 Wireless Serial Port Communication Module,” 14 Core, [Online]. Available: https://www.14core.com/wp-content/uploads/2016/03/HC-12-Documentations.pdf. [Accessed 29 April 2018].

[26] “XCSOURCE 433Mhz HC-12,” Amazon, [Online]. Available: https://www.amazon.co.uk/XCSOURCE-Wireless-Replace-Bluetooth-TE296/dp/B0148BLVE0. [Accessed 29 April 2018].

[27] Digi, “Digi Xbee SC2 802.15.4 RF modules datasheet,” [Online]. Available: https://www.digi.com/pdf/ds_xbee-s2c-802-15-4.pdf. [Accessed 29 April 2018].

[28] “XB24CAPIT-001,” Mouser Electronics, [Online]. Available: https://www.mouser.co.uk/ProductDetail/Digi-International/XB24CAPIT-001/?qs=%2fPVulymFwT3%2f4LVtV76gSw==&utm_source=eciaauthorized&utm_medium=aggregator&utm_campaign=XB24CAPIT-001&utm_term=XB24CAPIT-001&utm_content=Digi-International. [Accessed 29 April 2018].

[29] “Tobii Eye tracking components and system designs.,” [Online]. Available: https://www.tobii.com/tech/products/components-system-design/#TobiiEyeSensors. [Accessed 8 April 2018].

[30] “Tobii Eye tracking components and system designs,” [Online]. Available: https://www.tobii.com/tech/products/components-system-design/#TobiiEyeChip. [Accessed 8 April 2018].

[31] “Tobii Tech System Architecture,” [Online]. Available: https://www.tobii.com/tech/technology/system-overview/. [Accessed 8 April 2018].

[32] M. Bardon, “Github Eye Tracking Project,” Github, [Online]. Available: https://github.com/Tohaker/eye-tracking-project. [Accessed 9 May 2018].

[33] M. Bardon, “Github Arduino Firmware,” Github, [Online]. Available: https://github.com/Tohaker/arduino-firmware. [Accessed 9 May 2018].

[34] “Getting Started,” Tobii, [Online]. Available: https://tobii.github.io/CoreSDK/articles/getting_started.html. [Accessed 8 November 2017].

[35] “Tobii Gaming Setup,” Tobii, [Online]. Available: https://tobiigaming.com/getstarted/. [Accessed 8 November 2017].

[36] “Interaction Samples,” Tobii, [Online]. Available: https://github.com/Tobii/interaction-samples. [Accessed 1 April 2018].

[37] "Wireless Connectivity Kit Documentation," Digi, [Online]. Available: https://www.digi.com/resources/documentation/Digidocs/90001456-13/Default.htm#concepts/c_wc_kit.htm%3FTocPath%3D_____1. [Accessed 31 Jan 2018].

[38] iMotions, "Pupil Centre Corneal Reflection".

[39] C. Label, "Diagram of Human Eye without Labels," [Online]. Available: https://madebycreativelabel.com/wp-content/uploads/2018/04/diagram-of-the-human-eye-without-labels-human-eye-diagram-without-labels-41822.jpg.

[40] Lucs-kho, "Reading Fixations Saccades," [Online]. Available: https://commons.wikimedia.org/wiki/File:Reading_Fixations_Saccades.jpg.

[41] NascarEd, "Sleep Stage REM," [Online]. Available: https://commons.wikimedia.org/wiki/File:Sleep_Stage_REM.png.

[42] Parallax, "XBee 802.15.4 Low-Power Module, PCB Antenna (XB24-API-001)," [Online]. Available: https://www.parallax.com/sites/default/files/styles/full-size-product/public/32404_0.png?itok=DL_JiQZK.

[43] E. Research, "Eye Tracking Picture," [Online]. Available: http://eyesee-research.com/wp-content/uploads/2016/09/eye.tracking.picture06.jpg.

[44] Tobii, "IS4 Specs," [Online]. Available: https://www.tobii.com/imagevault/publishedmedia/5s6yjx035dq2ex1ax5en/Web_Images_IS4_specs_3840x1559-2.jpg.

[45] SeeedStudio, "434MHz Transceive Module 1km," [Online]. Available: https://statics3.seeedstudio.com/product/434Mhz%20Transceiver%20Module1km_01.jpg.

[46] M. G. e. al, "Bright pupil by infrared or near infrared illumination," [Online]. Available: https://commons.wikimedia.org/wiki/File:Bright_pupil_by_infrared_or_near_infrared_illumination.jpg.

[47] M. G. e. al, "Dark pupil by infrared or near infrared illumination," [Online]. Available: https://commons.wikimedia.org/wiki/File:Dark_pupil_by_infrared_or_near_infrared_illumination.jpg.

[48] Arduino, "Arduino Wireless Proto Shield," [Online]. Available: https://store-cdn.arduino.cc/uni/catalog/product/cache/1/image/520x330/604a3538c15e081937dbfbd20aa60aad/A/0/A000064_featured_2.jpg.

[49] H. Components, "NRF24I01 Wireless Radio Transceiver Module," [Online]. Available: http://hobbycomponents.com/2136-large_default/nrf24l01-24ghz-wireless-radio-transceiver-module.jpg.

[50] Crazepony, "4 Wheel Robot Smart Car Chassis Kit," [Online]. Available: https://images-na.ssl-images-amazon.com/images/I/61MjDn97kAL._SL1000_.jpg.

[51] Tobii, "EyeX Controller Archives," [Online]. Available: https://developer.tobii.com/wp-content/uploads/2014/04/Tobii-EyeX-Controller.png. [Accessed 9 May 2018].

[52] Tobii, "Tobii Dynavox PCEye Mini," [Online]. Available: https://www.tobiidynavox.com/globalassets/pictures/devices/eyemobile-mini/tobiidynavox-pceye-mini-1920x1080px.jpg. [Accessed 9 May 2018].

Appendix A

Eye Tracking Project Questionnaire

This survey will ask questions related to Miles Bardon's Final Year Engineering project at The University of Nottingham. These questions are designed to assess the public interest in the commercialisation of Eye Tracking control for consumer robotics.

If you have any questions or concerns about the project, or the data collected, please email Miles at eeymb5@nottingham.ac.uk

*Required

1. Do you, or anyone you know, rely on AAC (Augmentative and Alternative Communication) methods? *

AAC are the communication methods used to supplement or replace speech or writing for those with such impairments.

Mark only one option.

- Yes
- No
- Do not wish to say

2. [If you answered "Yes" to the previous question] Which methods are used?

Tick all that apply.

- Sign Language
- Gestures
- Communication Books/Boards
- Talking Keyboards
- Computer applications that use Eye Tracking
- Other:

3. Would you benefit from simple assistance from a robot controlled by eye movement? An example could be a robotic arm that assists in grasping nearby objects. *

Mark only one option.

- Yes
- No

4. If you answered "Yes" to the previous question, what function would you see the most benefit from?

5. How much would you EXPECT to pay for the above described system? This would include an eye tracker, robotics and Windows application. *

Mark only one option.

- £0 - £100
- £100 - £200
- £200 - £500
- £500 - £1000
- £1000+
- Other: