Enabling and Encouraging Young People to Become Programmers with Educational Robots

16th April 2015

UNIVERSITY OF
LINCOLN

Adam Bowes

BOW10237560

BSc. (Hons) Games Computing

School of Computer Science

The University of Lincoln

Supervisor: **Dr Grzegorz Cielniak**

Enabling and Encouraging Young People to Become Programmers with Educational Robots

Adam Bowes

Acknowledgements

I am very grateful to my supervisor Dr. Grzegorz Cielniak whose advice and encouragement push the quality of this project beyond its initial scope and scale.

I would also like to thank Alex Saye for setting the bar so high for me to reach, without your example my own efforts would lack direction and motivation.

Abstract

Programming is a very useful and desirable skill that is becoming more important as technology advances, this project explore common issues that face those learning to program and attempts to solve them by introducing robots. Students can find programming overly complex and difficult to grasp as they start to learn. During this project an interface has been created for the programming language Scratch 1.4 so that it could be used to program the Thymio-II robot, which are both already commonly used in education. The interface was created in Python 2.7.9 and was built to run on a Linux computer such that a Raspberry Pi computer could be used. The interface can also be used remotely with a battery pack and a Raspberry Pi meaning a windows machine can be used as well. A tutorial for setup and an instructional workshop has been created. A workshop session was ran with applicants for a computer science course and they used the interface with the tutorial, workshop tasks and guidance. They were asked to fill out a short questionnaire about the interface and data about the commands sent was gathered. From an analysis of the collected data it appears that the interface allows people to engage in programming robots without any former knowledge or experience. As well as this it creates an environment where students become enthusiastic towards programming due to its intuitive usage and features though the methods students use are heavily influenced by examples and guidance.

Table of Contents

Acknowledgements 3
Abstract 4
Table of Contents 5
Figures and Tables 7
1 Introduction 8
2 Literature Review 9
2.1 Education 9
2.2 Programming Language 10
2.3 Robotics 10
2.4 Conclusion 11
3 Methodology 12
3.1 Project Management 12
3.1.1 Software Development 12
3.1.2 Gantt chart 13
3.1.3 Risk assessment 13
3.2 Toolsets 15
3.2.1 Programming 15
3.2.2 Editing and management 18
3.3 Research methods 19
4 Design 21
4.1 Scratch 1.4 communication 22
4.2 Thymio-II communication 22
4.3 Interface 23
4.3.1 Communication 23
4.3.2 Thymio-II Control modes 24
5 Development 25
5.1 Python Interface implementation 25
5.1.1 Multiple threads 25
5.1.2 Connection to the Thymio-II 25
5.1.3 Connecting to Scratch 1.4 26
5.1.4 Commands 27
5.1.5 Receiving variables from Scratch 1.4 27
5.1.6 Arc movement 28
5.2 Scratch 1.4 Interface implementation 29

5.3 Setup 29
5.4 Tutorial and workshop tasks 30
6 Evaluation 32
6.1 Workshop session 32
6.2 Questionnaire results 33
6.3 Command data results 34
6.4 Conclusion 35
7 Critical Reflection 37
7.1 Project reflection 37
7.2 Personal reflection 38
8 Appendices 39
8.1 Python interface code 39
8.2 Interface Setup Tutorial 44
8.3 Workshop tasks 47
8.4 Consent form 50
8.5 Questionnaire 51
8.6 Gantt Chart 52
8.7 Workshop demonstration code 53
9 Reference List 54

Figures and Tables

Figure 1: Overview of the interface 21
Figure 2: A representation of the interface thread and communication structure 23
Figure 3: The D-Bus network being created to Asebamedulla 25
Figure 4: The variable handling for receiving a variable 26
Figure 5: The parameters of the arc movement 28
Figure 6: The function used to sort the sensor data 29
Figure 7: Thymio-II wireless setup 30
Figure 8: Percentage of variables changed 34
Figure 9: Percentage of commands sent 35

Table 1: Questionnaire results 33

1 Introduction

Robots have been used to further education and to increase engagement in a range of topics. For example, in secondary school mathematics robots have been used to demonstrate geometric transformations. Programming is a skill with which many have encountered difficulties understanding and persisting with, one cause is a lack of motivation and enthusiasm towards the topic. Another cause is that the interaction with the computer is limited to the computers screen preventing any physical or real world feedback. In this research an interface to allow the use of Scratch, a user friendly programming language to control a Thymio-II robot was created to allow users to have their programs affect the real world. This interface should also enable users to program robots in a familiar and comfortable environment allowing them to learn how to program more effectively.

- The first objective of this project is to understand the Scratch and Thymio-II software in such a way that an interface can be built from that knowledge. By understanding how to communicate between the software components fully before planning the interface the design process can be optimised.
- Secondly, an objective is to create an interface between the programming language Scratch and the Thymio-II robot. This would be the main artefact from this project and would allow users to program in Scratch to control the Thymio-II robot.
- For this project the Scratch 1.4 would be adapted to conform to the interface to improve the ease of use. This would make use of Scratch 1.4 code to manage communication with the interface as well as to store and update variables from sensors.
- Another objective is to run a focus group session. Materials and tutorials appropriate for the attendants of the session will need to be created and the event will need to be organised and advertised.
- Finally the results of the focus group will need to be evaluated by compiling questionnaire results and analysing the answers to see if the Scratch and Thymio interface is usable.

2 Literature Review

Programming and computer skills are becoming increasingly important as the influence of the internet and the power of computers grows, programming has even been called the 'second literacy'. Despite this there are still significant barriers to education in this field and few attempts to integrate it in to other computer related topics. This project will use the language Scratch 1.4 and the Thymio-II as they have both been shown to be good at introducing people to the topic of programming and robots as well as maintaining interest and creating enthusiasm. The first section explores education and why robots aren't already being used and what benefits they might have. The second evaluates different languages and how they might be viewed by first time programmers. The final section looks at robots and the barriers that used to exist preventing them from being used and examples of their successful use in education.

2.1 Education

Amongst the reasons why teachers don't accept technology in to the class as readily as they do other tool are stress and fear of failure. When a teacher tries to teach using methods in which they have little experience they often find that it can be daunting and cause stress. Introducing people to Scratch has both caused people to be more likely to include programming in lessons and to worry about failing less when they consider programming courses or sessions. A study of students learning to become preschool teachers were given an introductory lesson in computer programming and found that interest in using technology in the classroom increased from 80% to 92%. As well as this they found that 65% found Scratch easy to use and 85% found it simple and understandable (Fesakis and Kiriaki, 2009).

Students learning to program often partake in lectures and other non-participatory forms of education. According to the constructivist learning is constructed by the student as a result of creating and building upon knowledge recursively. This means that students would learn by recursively building upon previous experience with new experiences. Constructivism also claims that passive learning attempts to give students the same information despite students having different ideas and preconceived perceptions (Ben-Ari, M. 1998). This method for education has some flaws such as in science where it is difficult to experience some concepts. In computer science students can directly interact with all aspects of a program making it suitable for constructivism. Several programming languages have been built with education and constructivism in mind such as Scratch and Logo (Shin, S. et al, 2013).

2.2 Programming Language

Scratch is a language developed at MIT which has been used in education with broad success. Scratch allows for the use of most programming concept without requiring the user to be aware of syntax through the use of blocks. As well as being easy to use it is also free to use and has a large community with a wide range of users from 4 to 60 year olds meaning support can be found relating to the most basic of tasks to complex ones. One study found that during a Harvard Summer School for Computer Science course that 76% of students felt that using Scratch as an introduction help them when they later moved on to java, students also found it was more rewarding to have visual feedback on what they had programmed than just having a text window (Malan and Leitner, 2007).

Though languages like Scratch tend to be limited and lack features available in languages like C++ and LISP, they do however have strong visual feedback enabling a user to understand the semantics of programming. General purpose languages often have issues with their size and their idiosyncratic language often preventing users from forming an understanding of fundamental principles that are required to program. General purpose languages are also used in professional environments which favour speed and efficiency over intuitiveness and ease of use. (Brusilovsky, P, et al 1997)

2.3 Robotics

One challenge facing robotics in education is the price of the platforms and how easy they are to use. The Thymio II can be bought for around £100 which is cheaper than alternatives such as the LEGO Mindstorm while still having most of the feature. Besides the LEGO robot there are few available robotics platforms that are simple enough that they can be used for an introduction to the topic while also being capable enough that they can perform complex programs. The Thymio II is a powerful system and with Scratch it would mean that it can be easily picked up by beginners while still being able to perform some complicated programs. Scratch include features such as the ability to create object orientate programs as well as use multithreading, and is seen to have one major limitation which is recursion which has been purposely left out so that beginners would not feel threatened by the complexity (Harvey, B. 2010).

Robots have been used before with other aspects of programming to create courses that have proven to create very enthusiastic students. For example, at the University of Lincoln robotics was taught alongside computer vision, this lead to positive results in practical sessions including some students ended up going far beyond the brief of their assignments with some advanced feature that they researched and implemented under their own direction. (Cielniak, G. et al, 2013), this suggests that enthusiasm can be created with practical assignments using robots.

The Thymio II is a programmable robot with a wide variety of sensors and methods for feedback. It has 2 wheels for movement, a speaker for audio output and several lights, some of which are programmable and others which indicate the feedback from the distance sensors. There are 9 distance sensors to prevent it from falling off objects and to detect thing in front or behind it. It also has a 3 axis accelerometer, a microphone and an infrared sensor for remote input. With these features the Thymio II is well suited to education as it can be applied to a lot of situations. The Thymio II is the result of testing amongst children with the Thymio II. After running courses with the Thymio 89.2% of parents thought the session was educational and 78.5% thought that it had increased their child's interest in robotics (Riedo, F. et al 2012)

2.4 Conclusion

Several conclusions can be drawn from this research. Firstly, there are several barriers to effective education in programming and robotics, one is the reluctance of educators to learn and use the available tools. Making use of currently employed programming languages such as Scratch could be one method of circumventing this issue. Another is the format of traditional education where students may not be able to experience concepts from programming and so might struggle to understand them without practical workshop sessions.

Another Conclusion would be that modern professional programming languages are not entirely suitable for beginners as they favour efficiency and effectiveness over transparency and ease of understanding. Languages such as Scratch however provide clear visual feedback on what is currently happening and make use of simple terms which can be easily understood allowing students to engage more and be less likely to be discouraged by complex and difficult to comprehend code.

Finally we can conclude that robots provide programmers with physical responses to their programming allowing them to better understand what their code does. As well as this, the use of robots has been seen to increase student’s enthusiasm towards practical aspects of programming robots. In addition to this this Robots are beginning to become cheaper and more widely available making them a suitable and practical educational aide.

3 Methodology

3.1 Project Management

3.1.1 Software Development

In order to carry out this project an interface needed to be developed. The interface needed to work with Scratch 1.4 and the Thymio-II robot. This piece of software is largely unique in its purpose and so it needed to be developed with a method which could accommodate the flexibility that would be necessary to complete it. A variation on the waterfall method was chosen called incremental. Incremental is a method which employs small waterfall stages allowing for reflection and adjustment after each (Cms, 2005).

Developing the interface required overcoming unique challenges which were specific to this type of software, as a result some segments of the code needed to be rewritten or upgraded as new knowledge was gained. Incremental development allows for the software to be reviewed after each task is completed so that improvements to the code can be added as tasks at suitable points in the development. Consistently improved code was important for this project as this meant the scope of the interface could expand without causing conflicts with obsolete code. Scratch 1.4 and the Thymio-II robot both needed to communicate with the interface and as neither components were built with compatibilities for the other the development process had to account for impassable obstacles which could result in fundamental aspects of the interface needing to be changed such as the programming language or the methods of communication. Incremental development suited this as its review process gives an opportunity to evaluate the current state of the program in addition allowing newly acquired knowledge to be applied to the situation.

The result of using this method meant that during initial development when experimentation took place there was little commitment to carrying on with a particular plan. Following this unsuccessful ideas could be discarded quickly and so that the core functionality of the interface was more robust. The incremental aspect of the development meant that the software was completed in discrete and complete chunks such that each iteration resulted in a working prototype with individually changing features. This discrete improvement meant that progress could be monitored easily by reviewing which features had been added or changed. In addition to this the scope of the interface was expand where appropriate during the brief review periods to include feature which previously could not be planned for as the relevant knowledge about the system was not known.

One issue with the method was that features were often completed on their own without consideration for future iterations. This caused large amounts of the interface to need to be

rewritten, although this had the advantage of ensuring the program was built with recently acquired knowledge it also caused the progress of the project to slow down. With a more rigid method features could be anticipated and prepared for better. Another issue with the development method was that while it was useful for increasing the scope of the software and monitoring progress it wasn't very useful for estimating the tasks remaining and the time they would take. As the software needed to be functional before any testing or studies could take place

3.1.2 Gantt chart

A Gantt chart was created in order to visualise the required tasks and the time frames that were allowed for them (Appendices 8.6). Gantt charts have many advantages such as they are easy to read and clearly show time required for a project and its tasks. The main downside to a Gantt chart is that they don't not usually show dependency and so calculating a critical path isn't usually possible (Wysocki, R. 2012). To counter this flaw time was allowed at several intervals for contingency so that tasks which take longer than intended can still be completed without having to modify the Gantt chart.

3.1.3 Risk assessment

In order to reduce the chance of an unexpected failure at some point in the project a chart has been created which outlines serious risks along with their likelihood. The chart then includes a contingence plan to manage the project after a failure as well as steps that can be taken to mitigate the risks.

Description	Likelihood	Severity	Contingency plan/ Mitigation
The Aseba software receives an update that changes the way it works meaning that previous research or code is no longer valid.	Low	High	The update notes shall be reviewed for changes that would affect the project, if the project can then it will be adapted to use the update within a reasonable amount of time. Otherwise the update will not be installed and the project will make use of the older version of Aseba.
The attendance of the focus group is low or there is no attendance.	Low	Medium	The focus group session will be rearranged and the method of advertising the session will be changed.
Robot specific hardware failure.	Low	Medium	Measures will be taken to ensure the robot is stored in safe and suitable locations when

			not in use. If a fault does occur then the robot will have to be replaced.
Focus group results are lost	Low	High	GitHub shall be used to store the files, meaning that they will be stored online; also if the data is overwritten or deleted then using GitHub the file can be reverted to previous version. If the data is lost then a secondary study will need to be performed.
Thymio II and Scratch are incompatible	Low	High	The language and the robot will need to be re-evaluated and alternatives will need to be considered.
A component or feature of the robot is incompatible with Scratch	Medium	Medium	Time will be spent considering a possible work around and alternate solutions. If none are available then the feature will have to be considered for its value to the operation of the robot. For example, the infrared receiver will not be required for the robot to work.
Data loss or corruption	Low	High	All work should be stored on GitHub with appropriate summaries and descriptions so that it is clear which version of the work is available.

3.2 Toolsets

Several tools were required to complete this project. In this section they shall be listed and their inclusion will be explained with comparisons to alternate software. The tools used in this project were as follows:

Programming

- Thymio-II robot
- Aseba
- Scratchpy
- Scratch 1.4
- Python 2.7.9

Editing and management

- Gedit
- GitHub
- Microsoft office suite

3.2.1 Programming

3.2.1.1 Thymio-II

As the aim of this project is to create the tools to enable education using robots, a suitable robot was needed to work with. When deciding which to use several available options were considered and evaluated before the Thymio-II was chosen. The Lego Mindstorm NXT, the WowWee Rovio and the Aseba Thymio-II were compared and the results are shown in the table below. As the robots were going to be used in an interface some normally prevalent aspects such as ease of use were not important as they would be hidden by the interface.

Robot	Cost	Connectivity	Sensors	Actuators	Battery life
Thymio-II	£84 (€115)	USB	Proximity – 7 Ground sensors - 2 Rotation – 3 axis Sound sensor - 1	Wheels - 2	3.7 Volts 1500 mAh
Mindstorm NXT	£440	USB Bluetooth	Touch sensor – 1 Light sensor – 1 Sound sensor – 1 Ultrasonic - 1	Motors – 3	7.4 Volts 2100 mAh

Rovio	£500	USB Wi-Fi	Camera – 1 Proximity – 1 Microphone – 1 Speaker - 1	Wheels – 3 Camera mount	6 Volts 3000 mAh

Out of all of the robots the Thymio-II was the cheapest at 19% of the NXT, the next cheapest robot. As the aim of the project is to encourage people to learn with robots it is important that the equipment is accessible, the other robots are less accessible due to their prices. The camera and the microphone on the Rovio cannot easily be integrated in to Scratch 1.4 and so would have little functionality remaining. The NXT has a variety of sensors that can be used in Scratch 1.4 as they can be simplified in to raw numbers. The Thymio-II sensors are also varied, on top of this there are lots of them for users to make use of. The camera mount on the Rovio is largely useless without using the camera. However the wheels on the Rovio allow for omni-directional travel due to rollers on the wheels making the Rovio quite mobile. The three motors on the NXT can have wheels attached or they can be used in other configurations such as the power tracks or simple legs. The Thymio-II has two wheels and a small plastic nub that slides over surfaces. The wheels on the Thymio-II have generic construction points that are compatible with Lego, but unlike the NXT the Thymio-IIs wheels are in fixed positions. The length of time a battery charge will last in one of the robots is dependent on the actions the robot is taking. Although the NXT and the Rovio have larger capacity batteries they also use the power faster due to their features such as cameras and ultrasonic sensors. The biggest flaw with the Thymio-II is the connectivity. While the other robots have wireless connection options the Thymio-II does not, this could be augmented with a small ARM computer added to the Thymio-II using the building block mounting points on its top. As well as this there has been research in to integrating a wireless interface without a significant impact on battery life (Retornaz, P. et al, 2013). After considering these three robots the Thymio-II stood out as the most suitable mostly due to its price and quantity of sensors.

3.2.1.2 Aseba

The Aseba software package gives the user access to the Thymio-II programming environment. The programming environment was used to quickly test concepts without having to fully implement them. Programs can also be loaded on to the Thymio-II which can be executed remotely. The Aseba software also includes Asebamedulla which is required to communicate to the Thymio-II from external software using D-Bus.

3.2.1.3 Scratchpy

Scratchpy is an open source python module which manages connections to Scratch 1.4 over TCP/IP (Pilliq 2014). This module was chosen as it can be downloaded and applied entirely from the terminal. This is important as it means that the process can be automated leaving less for the users to have to do. The project is not focused on communicating with Scratch 1.4 and so this module saves a significant amount of time meaning that another communication method would not have to be made.

3.2.1.4 Scratch 1.4

A programming environment is required so that the user has a means of interacting with the interface. This artefact of this project is aimed at young people and people new to programming. As such the environment user's use will need to be easy to use and intuitive. As well as this the environment will also need to allow for remote input.

Scratch makes use of blocks rather than text for coding. This prevents users from encountering syntax errors and means that they are not required to memorise key words or commands. All of the blocks that a user can use are available in a window to the left. The blocks are also sorted by type with categories such as operators, control and variables so they can be easily found. All of the blocks in Scratch can be execute by themselves allowing programmers to test and explore code easily (Maloney, J. et al, 2010).

There are currently two versions of Scratch available, version 1.4 and 2.0. Scratch 2.0 is an update to Scratch 1.4 and has been ported in to Adobe Flash to allow it to run in a web browser. The update includes customizable blocks, a sound editor, time blocks and image editing. Scratch 2.0 can be used offline but it requires Adobe Air to run which means installing Scratch 2.0 on Linux operating systems is difficult requiring use of the command line. Scratch 2.0 has also had its remote connections server removed as it now runs in a web browser. Scratch 1.4 however has a remote connections server as well as being easy to install on all operating systems.

Due to the remote connections as well as a standalone Linux version Scratch 1.4 was used for this project. Another significant advantage of using Scratch 1.4 is that it can be run on a Raspberry Pi, a cheap and accessible arm computer, making the interface more accessible.

3.2.1.5 Python 2.7.9

While the user facing side of the interface needs to be easy to use the actual interface itself needs to have the complexity to manage commands between a robot and a programming environment. With this in mind the language used to program the interface needs to be the most suitable to the two chosen components, in this instance Scratch 1.4 and the Thymio-II.

During research it was discovered that the Thymio-II allows for control over D-Bus. This was built primarily for debugging and allows access to all variables and functions and gives full control over Thymio-II. Scratch is a popular programming environment and already has modules to allow scratch to communicate with some programming languages.

Python is one of the languages with both a D-Bus module. The D-Bus module allows for connections to the Thymio-II such that it can be controlled. There is also a Scratch 1.4 connection module called scratchpy. Scratchpy manages all communication between Scratch 1.4 and Python allowing messages and variables to be sent and received.

Although the interface is not required to be accessible to users it is a desirable feature as users may want to advance from Scratch 1.4 at some point while still using the interface. Python is popular language with lots of support and documentation available online. As well as this it also has a garbage collector meaning the user doesn't need to manage the programs memory. Another attractive feature is the way that variables in python don't need to be declared explicitly but instead are inferred by the interpreter.

Due to the ability for Python to communicate with both the Thymio-II and Scratch 1.4 as well as its support and user friendly features it is the most suitable for this project.

3.2.2 Editing and management

3.2.2.1 Gedit and Idle

As the interface will need to be programmed in a Linux operating system a suitable editor is also required. Python is an interpreted language and so does not need a compiler and is instead read and executed at runtime by a Python interpreter (Sanner, M. F. 1999). As a result creating the interface does not significantly benefit from an integrated development environment (IDE) like C based languages or Java. Instead Python can be programmed using a text editor.

For this project two editors were used. Gedit is the default text editor for GNOME based desktop environments. It supports line counting and simple syntax highlighting for recognised key words (Sebastian Wilmet, 2005). Gedit is Light and quick to use allowing for a program to be amended quickly and then saved and run in the terminal. One advantage of this is that the finished interface was executed from the terminal so while programming the output from the interface was the same as what a user would be expected to encounter.

The other editor was Idle. Idle is in actuality an IDE that has a built in interpreter allowing code to be tested quickly. Idle also has syntax highlighting like Gedit but Idle also has auto completion for object members. Idle is useful for writing large segments of code as it can check for errors and

help with auto completion. Python uses indents to define scope and Idle manages scope automatically again making development easier.

As Gedit is quick and easy to use but lacks some features and Idle is fully featured but not so quick to start and runs code in an environment different to the one the users will use both editors have been used.

3.2.2.2 Microsoft Office

As part of a study a questionnaire and a consent form will be required as well as workshop and tutorial documents. The Microsoft Office suite includes a fully featured word processor with a range of formatting options allowing it to be used for creating a simple consent form and a questionnaire.

The Office suite also includes Microsoft Excel which is a spread sheet program. This program can be used for contacting and presenting data via its ability to tabulate and graph raw data. As it is part of the same suite as Word graphs and tables can be copied directly from one program to the other with no issues with formatting. On top of this changes to a table, a graph or the data in Excel are transferred directly in to word ensuring that word always has up to date data.

An alternative to word would be LibreOffice, a free office suite similar to Microsoft Office. LibreOffice contains both a spread sheet program and a word processor but lack the formatting options of Word and the interconnectivity that Word and Excel share.

3.2.2.3 GitHub

In order to effectively manage work as well as maintain backups and manage changes GitHub was used. Git is a version tracking software which allows a user to build a repository for a project and enter messages that are associated with changes they make; these changes are stored on a database allowing mistakes or errors to be reverted such that the repository returns to a functioning state.

GitHub extends this versioning software so that repositories can be stored online. This means that a project is backed up on all computers it is stored on as well as on the GitHub servers. Being online the code is also public access allowing people to create forks in the project so that they can develop it themselves and download the project for use. This gives users an easy way to download the project as well as a safe way to back up and store the project.

3.3 Research methods

The goal of this project was to create an interface allowing users to program in Scratch 1.4 and have the code run on a Thymio-II in an appropriate way. The interface needs to be user friendly

and intuitive to use as well as allowing users to create complex code. In order to evaluate the success of the project the research will need to be carried out to determine these factors of the interface. To investigate these properties a substantial number of people will need to use the interface along with the accompanying tutorials.

To do this, qualitative research needs to take place in an environment where the users can have access to the hardware and software and their experiences can be recorded. To achieve this, a focus group will be used. A focus group is when multiple individuals gather to perform 'focused' task and influence each other with their ideas. This will be used as it is easy to implement as compared to other methods, has a low cost to create and run and also because it allows for a large sample size (Freitas, H. et al, 1998).

The focus group will be given a tutorial and a series of tasks to complete, after they finish or run out of time they will be given a questionnaire to complete. The questionnaire is standardised and uses the Likert scale in order to get a discrete measure of their opinion towards their experience. The main aim of the questionnaire is to collect information about user's experience with the interface. This is important information to gather as the main qualities of the interface being tested for are subjective to the individual user.

Beside the questionnaire, information about the commands users sent will be gathered, this will provide information about the number of unique commands sent and the variables that users set. This information will give context to the questionnaire as it will be possible to see how they interacted with the interface and so better understand the users experience in terms of usability and intuitiveness.

4 Design

For this project an interface between Scratch 1.4 and the Thymio-II robot was developed. In order to build this the architecture and functionality of each component had to be evaluated and designed for. The Scratch 1.4 programming language contains trigger blocks which react to events allowing for event based programs to be developed. On top of this Scratch 1.4 supports concurrent processes and thread based programming (Maloney, J. et al, 2010). In relation to building an interface this flexibility allows for a range of approaches to be made. The Thymio-II makes use of its own programming language where code is executed when events are triggered. These events include timers, sensor updates and user created functions (Shin, J., and Magnenat, S. 2014). This event based programming caused conflicts with thread based programs in Scratch 1.4. The interface was decided to be built using python which supports multithreading as well as event based programming.

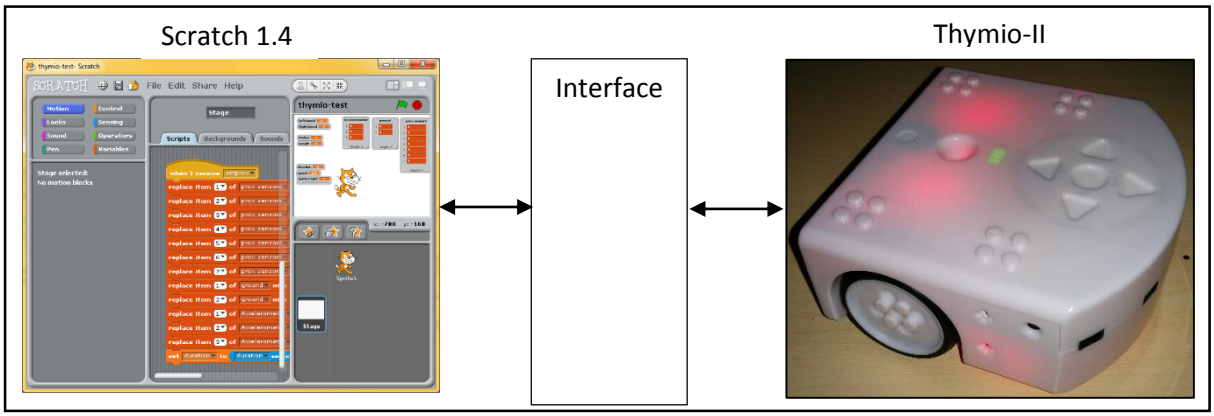


FIGURE 1: OVERVIEW OF THE INTERFACE

4.1 Scratch 1.4 communication

Internally Scratch 1.4 uses broadcasts which send a message to all objects in the project, if an object contains a trigger for that specific message then it will be executed. As well as broadcast the variables in Scratch 1.4 are global allowing objects to communicate values directly.

Scratch 1.4 already has support for external sensors in the form of remote sensor connections (RSC). Enabling the RSC causes scratch to run a server which will transmit all variable changes, variable creation and broadcast to all connected devices. Broadcast are sent as a string containing the value of the message being broadcast. Variable changes and creation are sent with the variable name and the new value. Unlike standard variables in Scratch 1.4 variable sent over the RSC will be received as sensor values which cannot be edited. Broadcasts can also be received by the RSC server which will broadcast the message to all objects in the Scratch project. Variables can set and created by send the RSC server the variable name and its value.

The ability for variables to be created through the RSC server allows for the interface to send any value without Scratch or the interface encountering a fatal error. Sensor values is the main way for data to be communicated in to Scratch 1.4 through the RSC server, as the value can't be edited it also won't be transmitted back to the interface. Due to the inability to easily edit standard variables in Scratch 1.4 from the interface feedback to the user about parameters entered cannot be easily given.

4.2 Thymio-II communication

Besides the custom built software the Thymio-II supports communication over D-Bus, a proprietary Linux protocol for communication. Using D-Bus variables can be requested. Requesting variables requires a handler to manager the different variable type that can be received. Variables on the Thymio-II can also be adjusted. The Thymio-II changes its state based of its local variable such that a variable corresponding to a LED can be altered to change the brightness of that LED.

Receiving messages from the Thymio-II requires a command to be sent first and then a response containing data to be sent back. Requesting large amounts of data often can cause network latency to increase and prevent other messages from being received. Sending commands to the Thymio-II requires variables to be changed, after this the Thymio-II will then have to recognise these changes and then relay the changes to the actuators. This method of sending commands is simple and handles timing and optimisation automatically, however as the Thymio-II might not react instantly some commands might be delayed.

4.3 Interface

4.3.1 Communication

With the methods of communication available and understood the structure of the interface can be planned. The Thymio-II will only send messages when requested by the interface and so all communication with the Thymio-II can be managed in the main thread and loop a there will be no wait time. Scratch 1.4 is the side of the interface which sends messages based on when the user wants to send them. The method available to receive commands from Scratch 1.4 require the thread to wait which would cause the thread it was running on to stop until a command has been received. As a result Scratch 1.4 messages cannot be received in the main thread without a message being received every loop. To solve this problem the interface will make use of a separate thread to handle all messages sent from Scratch1.4. As this thread would be dedicated to messages from Scratch it can be left to wait for messages. Global variables in python can be accessed by multiple threads so the extra thread doesn't restrict transferring data.

All variables related to the state of the, Thymio-II such as speed, duration and command are global so that the Scratch message handling thread can set their values and the main thread can read their values.

The commands from Scratch 1.4 are received in the second thread and then the message is interpreted. The message is compared to a list of known commands, if it matches then the message is a command and the command is carried out. If the message is a variable then a series of if statements checks to see if it's for a known variable and if it is the value from the message is set in the interface.

A diagram of this structure can be seen below in figure 2.

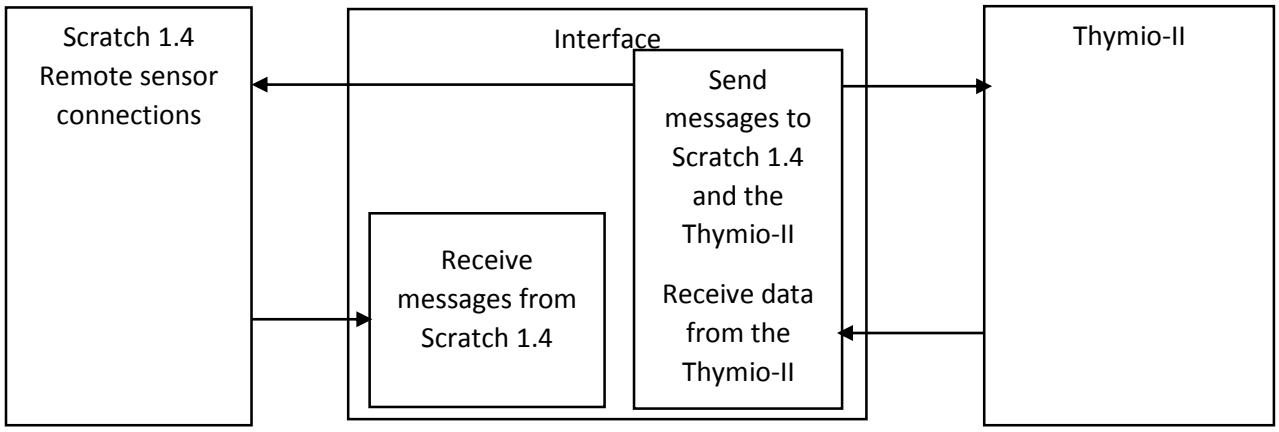


FIGURE 2: A REPRESENTATION OF THE INTERFACE THREAD AND COMMUNICATION STRUCTURE

4.3.2 Thymio-II Control modes

The interface needs to be accessible to inexperienced programmers but have the capability to handle complicated tasks. Accordingly the interface can be used to control the Thymio-II in 3 different ways.

The First and easiest method of controlling the Thymio-II is with commands and a duration. In this mode a user can send commands such as “forward” or “left” and then the robot will react accordingly. This mode limits the Thymio-II to moving in straight lines and turning on the spot or moving in a fixed arc. The user can set the radius and the length of the arc and then send the “arc” command to cause the Thymio-II to move in an arc. This mode is useful for manual control of the Thymio-II allowing users to become familiar with the interface. As commands queue in the interface this mode is not useful for loops but can be used for simple linear programs.

The Second method for controlling the Thymio-II is similar to the first but requires the duration of commands to be set to zero. When the duration is zero the commands (except arc) will continue to execute until another command is sent. For example, if the forward command is sent then in will continue forward until another command is sent. Repeat commands are ignored preventing commands from building up in the queue too much. As the Thymio-II will never stop following a command until another is sent there is a special stop command. The stop command is called “null” and is interpreted as a stop by the interface but it actually represents an empty command which causes the Thymio-II to stop until another command is sent. The null command has no practical purpose outside of this mode. This mode is very useful for programs with loops and conditions such as a program which reacts to input from the sensors.

The final mode for controlling the Thymio-II is with direct input. This mode is entered by sending the command “direct” and can be left by sending the command “command”. In the direct control mode the individual wheels can be controlled. Each wheel has a variable which controls the speed of that wheel. With these variables the users can control the Thymio to move in any way possible. To stop the Thymio-II the user must set both of the wheel speed variables to zero. This mode allows for complete control and is difficult to use manually and is better utilised by programs in Scratch 1.4. This mode is useful for creating behaviours similar to the Braitenberg vehicles.

5 Development

This section will explain the development process of the project. The first section will elaborate on the steps taken to create the interface in Python. In addition any issues encountered will be described along with their solutions. The Second section will explain the Scratch 1.4 side of the interface. The Third section will explain the setup and the wireless use of the interface. The fourth section will cover the tutorial and workshop tasks followed by the setup. The final code for the interface can be seen in appendices 8.1.

5.1 Python Interface implementation

5.1.1 Multiple threads

The interface requires multiple threads in order to operate effectively. The main thread handles communication with the Thymio-II and translating commands from the user in to actions. As there is potentially a lot of data that needs to be communicated between the interface and the Thymio-II the main loop is limited to repeating every 100 milliseconds. This helps prevent lag building up over the connection to the Thymio-II but this has the drawback of causing some delay between user input and action. The second thread receives commands from Scratch 1.4. This loop runs as often as it can. Due to the nature of the function that receives commands the thread will wait for a command to be received and then execute without any delay.

5.1.2 Connection to the Thymio-II

In order for the interface to operate it required a connection to the Thymio-II robot. As previously stated D-Bus can be used to communicate with the Thymio-II. One of Pythons default modules is the “dbus” module, with this module Python can connect to Asebamedulla and through that interact with the Thymio. To connect to the Thymio a D-Bus interface is needed and this interface uses the network named “ch.epfl.mobots.AsebaNetwork” which is created by Asebamedulla (figure 3)

```
dbus.mainloop.glib.DBusGMainLoop(set_as_default=True)

if options.system:
    bus = dbus.SystemBus()
else:
    bus = dbus.SessionBus()

network = dbus.Interface(bus.get_object('ch.epfl.mobots.Aseba', '/'), dbus_interface='ch.epfl.mobots.AsebaNetw
```

FIGURE 3: THE D-BUS NETWORK BEING CREATED TO ASEBAMEDULLA

This network can be used to communicate with the Thymio-II robot. The extent of the communication is limited to setting the value of variables, receiving the value of variables and

executing pre-programmed functions. Functions work by creating a custom function in the Aseba programming environment and then loading this code on to the Thymio-II and running it. Then using the network variable an event can be called with the event id (the position of the event in a list of custom functions) and the corresponding parameters.

Setting variables is the simplest form of communication with the Thymio-II over D-Bus. As with custom functions the network variable is required to set the variables. To set a variable the first parameter required is the name of the device, in this case the name is “thymio-II”. The next parameter is the variable name. The variable names are the same as can be found in the Aseba language including the parent structure, for example setting the target speed on the left motor requires the variable name “motor.left.target”. The final variable required is the value being sent. When the variable and values are sent the Thymio-II will receive them and set them internally. In this example, setting the target speed will cause the wheel speed to increase or decrease to match it automatically.

```
def get_variables_reply_prox(r):
    global proxSensorsVal
    proxSensorsVal=r

def get_variables_reply_delta(r):
    global groundDeltaVal
    groundDeltaVal=r

def get_variables_reply_acc(r):
    global accVal
    accVal=r

def get_variables_error(e):
    print 'error:'
    print str(e)
    loop.quit()
```

FIGURE 4: THE VARIABLE HANDLING FOR RECEIVING A VARIABLE

Getting a variable from the Thymio-II requires more steps than the others. As with getting a variable the device and variable name are required. As well as these functions are required to handle the data. These functions need to be tailored to the variable being received as the accelerometer returns three values and the proximity sensors returns seven values. The main difference between these functions is the variable the results are assigned to (figure 4). Finally receiving a variable requires an error handler which will receive any errors that occur.

5.1.3 Connecting to Scratch 1.4

Connecting to Scratch 1.4 requires for scratch to have its remote connections enabled. This can be done by opening the sensing tab in Scratch 1.4 and right clicking on the sensor value block and clicking on the “enable remote sensor connections” button. With this done Scratch 1.4 will transmit all variable changes and broadcast as well as receive broadcasts and variables in the form of sensor values.

Communicating with Scratch 1.4 in this way requires messages to be sent in a specific format with a header. As handling this communication is not the purpose of this project a Python module called “scratchpy” is used instead. This module can be used to handle sending data as well as receiving it from Scratch 1.4 (Pilliq 2014). As with D-Bus a scratch object is required to handle all

communication. To create this object the IP address of the computer running Scratch 1.4 is required.

Variables and broadcasts can be sent to Scratch 1.4, variable show up as sensor values with the variable name and value. Broadcasts are strings which can be received by certain control blocks to start executing code. To send a variable the name and the value are required, multiple values can be sent at the same time though each variable can only hold a single value. Broadcasting a message requires the message to be broadcast.

5.1.4 Commands

The user can send commands to the Interface using a broadcast in Scratch 1.4. To manage commands the interface stores a list of commands hard coded in to it. This list contains the following: forward, backward, left, right, null, direct, command, arc. When a broadcast is received the interface checks if the broadcast is the same as one of the elements in the list, if it is then the broadcast is set to a global variable so that the broadcast message can be accessed by other threads. The thread is then set to sleep for the duration that the user has set in Scratch 1.4. If the arc command is sent then the thread will sleep for as long as it takes to complete the arc at the current speed.

Once the command has been verified the main thread has access to it. In the main thread there are several if statements which set variable on the Thymio-II. For example, the command forward set the left and right target motor values to the wheels speed. With this system the commands can be received and relayed to the main thread without any interruption or miscellaneous commands.

5.1.5 Receiving variables from Scratch 1.4

When receiving variables from Scratch 1.4 the interface needs to put the values in to the correct variable in the interface so that they can be used to control the Thymio-II. When Scratch 1.4 sends variables it sends them in a dictionary format with the variable name and then the value after it. Using this the interface makes use of pre-set names. This allows an if statement to check if the current variable is an expected variable and then set the value of the corresponding variable in the interface.

When receiving a variable from Scratch 1.4 the first component of a message will contain the string “sensor-update”, with this the interface can distinguish variables from broadcasts. The second part of the message is a dictionary of the variable name and the value that was changed.

Using the key word ‘global’ the thread can access the global variable and assign the new value received from Scratch 1.4 to it.

5.1.6 Arc movement

One of the unique functions the interface can be used for is movement in an arc. The function requires a user to specify the radius of the circle, with a positive radius to the left and a negative radius to the right. As well as the radius the user needs to define the length of the arc that the Thymio-II will travel. The function then causes the centre of the Thymio-II (the pen hole) to travel the specified distance around a circle with the specified radius at the current speed.

To do this the wheels need to move at different speeds to create the arc. Each wheel is approximately 5cm from the centre of the Thymio-II. With these variables the distance that each wheel will need to travel can be calculated. The first step is to calculate the angle (theta) which can be found with the length of the arc divided by the radius. With this the speed of each wheel needs to be calculated. Using the theta from earlier to multiply the radius plus and minus 5cm (distance to the wheels) the length of each arc is found. By dividing the inner arc by the outer arc the percentage difference is found. The outer wheel then moves at the current wheel speed and the inner wheel movers at the wheel speed multiplied by the percentage difference between the arcs. This cause the inner wheel to move slower than the outer one causing the Thymio-II to travel in the appropriate arc (figure 5).

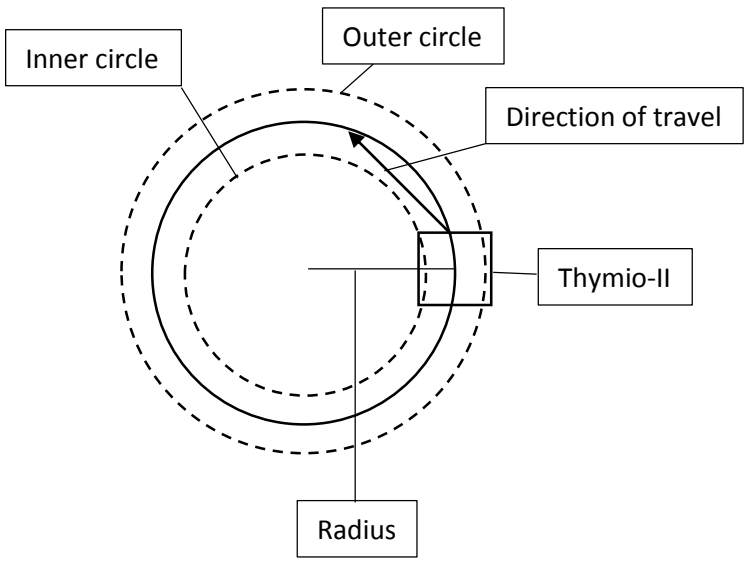


FIGURE 5: THE PARAMETERS OF THE ARC MOVEMENT

5.2 Scratch 1.4 Interface implementation

The interface needs to relay information to Scratch 1.4 and the information needs to be in a usable and accessible format. As potentially twelve data points can be returned every tenth of a second the values need to use the Scratch 1.4 lists rather than the default sensor value format. To allow this to happen a custom scratch program has been made. The program makes use of the control block “when I receive” which executes the code below it when a certain broadcast is received. Below the control block are several blacks which assign the values in to lists (figure 6).

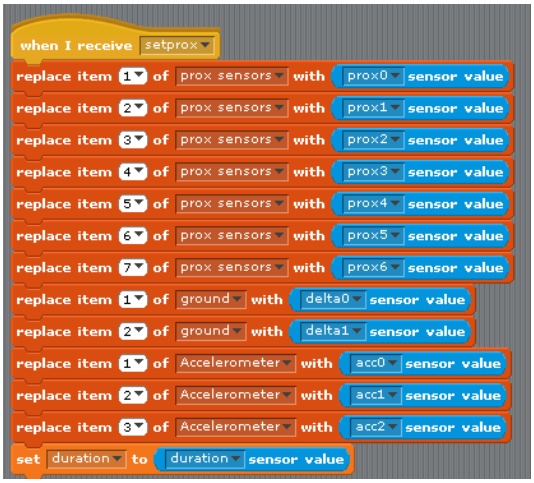


FIGURE 6: THE FUNCTION USED TO SORT THE SENSOR DATA

This function allows the interface to tell Scratch 1.4 to update its values whenever the interface requires saving processing time on the machine running Scratch 1.4.

With the sensor data in lists it can be accessed and manipulated more easily with functions such as searching through lists. List data can also be shown on the window in the top left of Scratch 1.4 allowing users to see what sensors are affected by an action.

5.3 Setup

To use the interface a user will need to follow a few steps. Initially installing the interface required the user to manually enter all of the commands in to a Linux terminal. Now there is a bash script that users can run and should install all components of the interface on any system using the Advance Packaging Tool. The user needs to run one command to allow the script to be run and then they can run the script which will install the following, a python module manager, the python module scratchpy, Scratch 1.4 and Aseba. This script reduces what the number of lines that the user has to run from six lines to two simple ones.

With the software installed the user then needs to ensure the Thymio-II is connected to the computer and run the command “sudo Asebamedulla “ser:name=Thymio-II”. This starts the program which will allow the interface to communicate with the Thymio-II over D-Bus. Next the user needs to run Scratch 1.4 and load the custom program mentioned in section 5.3. Finally the user needs to run the interface which will connect to both the Thymio-II and Scratch.

The Thymio can be controlled remotely though this requires more setup and more components. The user will need a battery pack, a Raspberry Pi and a Wi-Fi adapter (figure 7). The Raspberry Pi is powered by the battery pack and the Wi-Fi adapter is connected to the Raspberry Pi. As mentioned in section 5.2.4 scratchpy requires the IP address of the computer running scratch meaning that the interface can connect with Scratch over a network. Besides the extra components the other main issue is getting the interface to run on the Raspberry Pi without a screen or keyboard attached. This can be overcome with Virtual Network Computing (VNC), a graphical desktop sharing system which can be used across multiple platforms using TCP/IP (Kaplinsky, K. V. 2001). This however requires VNC to be running on the Raspberry Pi which could be done using a secure shell but requires some knowledge of the program.

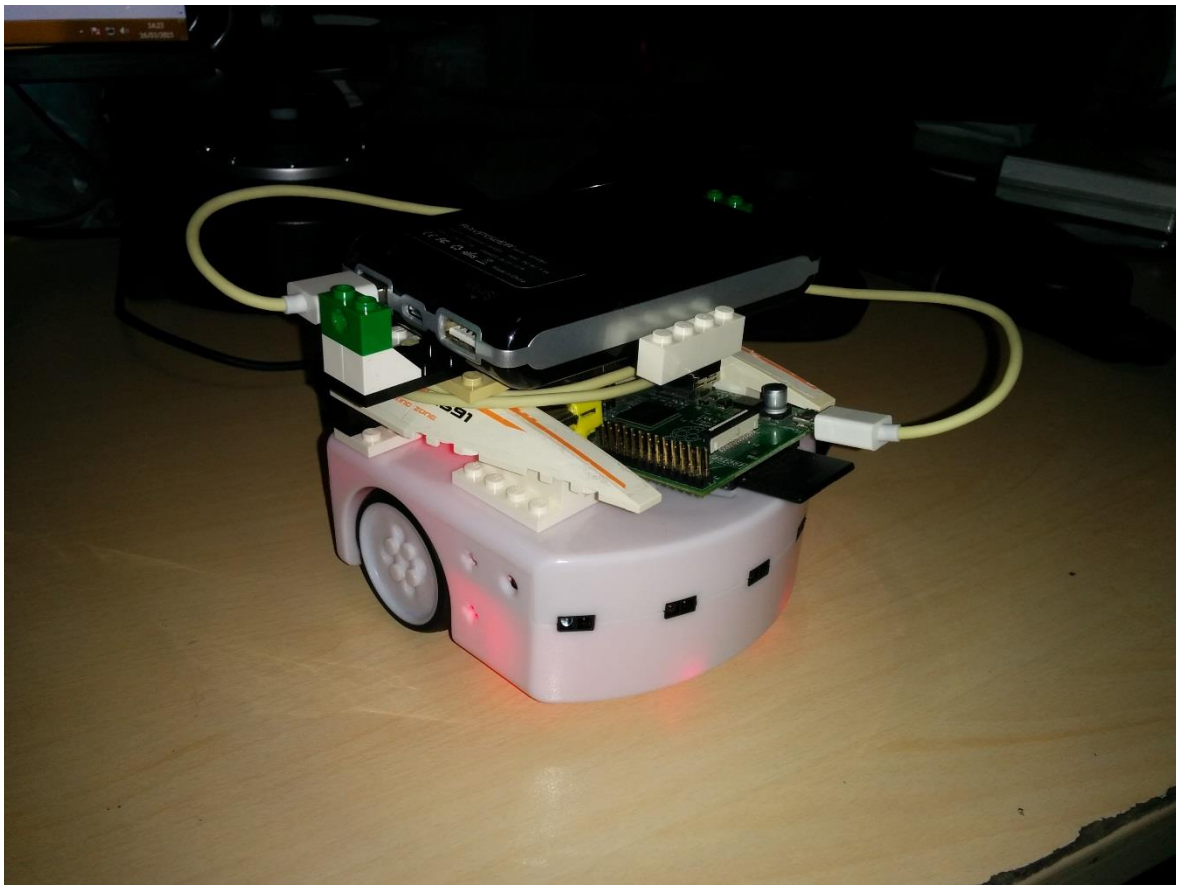


FIGURE 7: THYMIO-II WIRELESS SETUP

5.4 Tutorial and workshop tasks

To give users guidance in using the interface a tutorial for setup and a set of workshop tasks were made. The tutorial can be seen in the appendices section 8.2. The brief introduction states the platforms supported and the basic use of the interface. There is then a section briefly explaining some of the commands available in the Linux terminal. The average user would not be expected to understand the Linux terminal and the tutorial provides commands which can be simply copied

and pasted. Some users may need to deviate from the tutorial to deal with different operating systems or package managers; the notes are intended to give the user the tools they would require to navigate in the terminal and a basic understanding of what the commands in the tutorial are doing.

The fast setup section details the use of the bash script which installs all required components. The decision to create the bash script was after discovering it would take about 20 minutes to read and complete the standard set up. The new set up takes approximately 5 minutes to complete

The next step details the order in which the different components need to be set up and contains some help on how to handle errors.

The workshop tasks are intended to teach users the different methods of using the interface. The first things stated are all of the default commands in a bold list to ensure that they are accessible to users at all times.

Initially the tasks go in to detail and explain exactly what the user needs to do. As the tasks go on they contain less and less detail in how to complete the tasks and instead give the users a goal and asks them to achieve this. The tasks also build off of each other so that users can create more complex programs while still thinking in largely simple terms. The workshop tasks can be seen in appendices 8.3.

6 Evaluation

The evaluation will describe and explain what took place and any issues in the workshop session. The second will look at the questionnaire results and infer their meaning to the project. The next section will examine at the data gathered by the interface as the participants used it and the final section will draw conclusions for the project from these results.

6.1 Workshop session

The workshop too place the 18th of February 2015 as part of an open day held by the University Of Lincoln School Of Computer Science. Five attendees to the open day were asked to volunteer without revealing the task. After the volunteers were introduced to the workshop they were all given the option to withdraw or to continue with the workshop but to not have any data gathered. All of the participants Agreed to take part.

The participants each sat at a computer with a Thymio-II connected and Scratch 1.4 open with the interface already running and the workshop document open (appendices 8.3). Loaded in to Scratch 1.4 was a short demonstration program which causes the robot to move towards an object its sensors detect (appendices 8.7). The interface was briefly explained to them along with the features of the Thymio-II. After this they were asked to initially control the robot with keyboard controls and then introduced to the workshop tasks.

Assistance was given throughout to the participants along with suggestions of extensions to programs the participants had made. Unfortunately during the session a bug was found in the interface which caused the interface to lag considerably. The bug was a result of the interface receiving commands faster than it could handle and was fixed by preventing a command from being executed twice in a row for the zero duration mode.

After approximately 30 minutes the participants had begun to complete the tasks and were asked to fill in the questionnaire. After this they returned to the open day tasks and 6 others who finished the open day tasks came over to participate. As there were only 5 computer and robots one had to watch. They were given the same introduction and tasks. After another 30 minutes they had finished most tasks and were dismissed after filling in the questionnaire. The questionnaires were collected and the data of commands sent were gathered.

6.2 Questionnaire results

The questionnaire (appendices 8.5) was created with consideration from a computer system usability questionnaire outlined and evaluated by IBM (Lewis, J. R. 1995). On the questionnaire there were 5 questions, here are the question with their results and some brief analysis (table 1).

TABLE 1: QUESTIONNAIRE RESULTS

question	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
1. The interface is intuitive	1	9	0	0	0
2. Learning to use the interface was easy	3	5	2	0	0
3. The tutorial tasks were difficult	0	1	3	5	1
4. The tutorial and guidance were clear and useful	5	3	2	0	0
5. Using the interface was enjoyable	5	5	0	0	0

1. **The interface is intuitive**

All participants agreed to some extent that the interface was intuitive. Most agreed with this statement suggesting that while they found the interface intuitive it could be more so while one participant felt that the interface was very intuitive. The lack of any neutral or negative feedback suggests that the interface is to a degree intuitive.

2. **Learning to use the interface was easy**

This question intends to find out about the participants experience in using the interface with no prior knowledge and only a short amount of time to learn it. 8 participants found it easy to learn though 2 were neutral. This indicates that the participants were able to learn to use the interface effectively in most cases. The 2 neutral cases however suggest that the tutorial and interaction during the workshop could be improved upon.

3. **The tutorial tasks were difficult**

This question was included to gauge workshop tasks to see if they were suitable for the participants. An ideal response would be distributed between neutral and disagree, this is almost what has happened. Neutral and disagree implies that the tasks were either suitable or a slightly easy such that the participants could complete them. The Strongly disagree indicates that they found the tasks too easy, this could be because they were applicants for a computer science course and may have had pre-existing experience in programming. The single agree suggests that user found the tasks difficult, this could be due to a lack of assistance or due to varying abilities of the group.

4. **The tutorial and guidance were clear and useful**

Question four relates to the interaction in the session and the help they were given. Again the majority agree with the question with only 2 giving a neutral response. These results indicate that the tutorials and guidance were largely useful and not hindrances to the participants.

5. **Using the interface was enjoyable**

This question directly asks about how enjoyable the interface was. The results from this question are quite conclusive with every participant giving a positive response. The results are half agree and half strongly agree suggesting there is some divide in opinion but only a minor one.

6.3 Command data results

Besides the questionnaire data was collected from the interface. The interface recorded every time a variable changed and every time a command was sent that was different to the previous command. The values for the ninth participant became corrupt and so are displayed as 0. The other results are shown as a percentage showing what percentage of the messages sent where what.

Due to a previously mentioned bug the interface would encounter lag. As a temporary fix the interface was restarted, this lead to a lot of the data becoming lost as the interface overwrote the previous data.

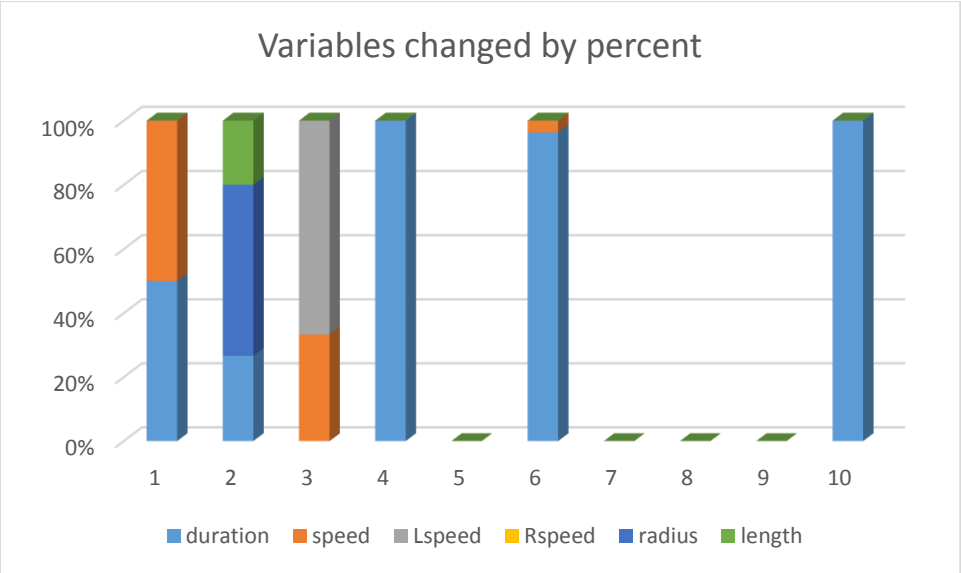


FIGURE 8: PERCENTAGE OF VARIABLES CHANGED

These results though quite varying show a favour towards the duration command and the speed command. These two variables are important for the basic mode for using the interface, with a

time for the action and a speed for the action set. Due to the nature of the results not much else can be reliably ascertained.

The data from the commands sent were also affected by the bug mention previously but these were sent more frequently than the variable changes and so are easier to interpret.

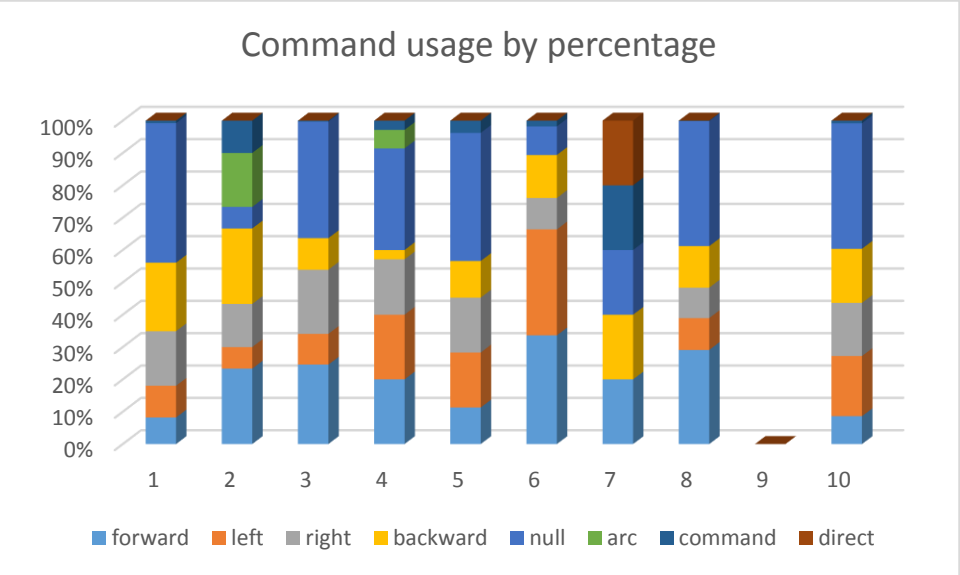


FIGURE 9: PERCENTAGE OF COMMANDS SENT

These show a wide usage of the four directional commands which indicates usage of the first mode and the second mode (4.4.2). Two users made use of the Arc command and one user sent the direct command suggesting they might have tried the direct when control method of control. Most users sent the "null" command (the command to stop the Thymio-II) suggesting that the second mode was used, this is understandable as this was the mode employed by the example program the participants were given.

6.4 Conclusion

The goal of this project was to create a user friendly and intuitive interface between Scratch 1.4 and the Thymio-II robot. The participant questionnaire shows that the interface was intuitive and met this goal though there is some room for improvement. Despite this the ease of learning and guidance could both be improved, possibly by having a move visual tutorial or possibly even a video walkthrough. The workshop tasks gained a range of responses suggesting that they were suitable to the majority of participants though could be made easier to tailor to the less capable users.

From the data from the Thymio-II it can be seen that the demonstration code shaped users experience of the interface and lead them to use the interface in a particular way. It also causes

them to mostly avoid more abstract or difficult methods of controlling the robot such as the direct wheel control which wasn't used by any of the participants to any significant degree. This suggests that the users experience is heavily affected by the examples they are given and so multiple examples could be given to give users an understanding of how to use different aspects of the interface.

For future work the interface would need to be used in an extended study to test its educational merit. Testing how useful it is and how much it helps students to learn. The interface could also be improved to work with the Wi-Fi module (Retornaz, P. et al, 2013).

7 Critical Reflection

7.1 Project reflection

The project as a whole was a reasonable success with all of the objectives of the project met within reason and with consideration to the scope and time scale of the project. The Interface was started early in the project and was functional after two weeks. The workshop session experience some technical issues but still yielded some results which gave a reasonably clear response.

Initially there was some difficulty in understanding how to communicate between the Thymio-II and Scratch 1.4. After a week of research and testing however a method of communicating with the Thymio-II was found.

After research it became clear that this project would be easiest to create in Linux due to D-Bus. Luckily this wasn't an issue as I was familiar with Ubuntu, Debian, Raspbian and Arch distributions of Linux. Despite this I felt it was necessary to allow the interface to be used with windows. Although this was eventually done using a Raspberry Pi it made the setup even more complex as it introduced VNC and SSH.

Although a working version of the interface was completed early it was built naively and meant that most additions needed large sections of the code to be re-written. Although this cause some development to take longer than it could have it did mean that new knowledge could be applied and the interface improved.

Running the workshop went well and more participants took part than were expected. The feedback was generally positive though I was a bit concerned that my presence might have affected their choices. The initial volunteers had all agreed to take part without knowing what was going on and so were likely outgoing and so were easier to interact with and more willing to try things and take advice. The second group were those who had finished the main task, suggesting that they were reasonably capable but at the same time seemed less out going and less willing to interact making the session harder to supervise.

The Gantt chart quickly became obsolete as development quickly reached its goal and time could be spent on additional features as well as workshop tasks. The incremental development method was used and was very useful allowing for revisions and improvements though it wasn't adhered to and instead came about naturally as development continued.

7.2 Personal reflection

Through this project I have learnt new skills and gain experience in managing a large project. I encounter python for the first time during this project before and so this posed a slight challenge as adjusting to the syntax and learning how to run the code took place. These challenges were relatively easy to overcome due to experience with languages like C++ with which I learnt core programing concepts and Action script 2 which shared some features like implicit variable types. All of these factors made me reasonable confident in creating a multi-threaded program.

As the interface was functional early on I learnt about project planning, mainly not to let it restrict progress on the project or its scope. The Gantt provided the majority of time to development, this time was instead put to use with improved features and meant that other tasks could be started sooner and situation like the open day could be taken advantage of.

Developing the interface was largely thanks to being flexible in languages and approaches. Initial the interface was planned to be created in C++ using Windows com ports to handle traffic. After research in to the Thymio-II development I discovered that Python was used to debug it, with knowledge from the research about how the Thymio-II worked I realised that it could be used to control the Thymio-II as well. This experience taught me that Starting from scratch isn't always necessary especially when considering the point of the project and the ultimate goal.

I found that my work ethic throughout was reasonably stable and successful with usually small but constant step taken for both development and writing this report. This lead to a largely stress free project with largely relaxed self-imposed deadlines. This idea initially sound lazy and poorly motivated but in practice it allows for time to reflect and to consider how to approach the next task.

8 Appendices

8.1 Python interface code

```
1.  import dbus
2.  import dbus.mainloop.glib
3.  import gobject
4.  import sys
5.  import time
6.  import scratch
7.  import thread
8.
9.  from optparse import OptionParser
10.
11. #variables
12. #length of operation
13. duration = 0.5
14. #speed of both wheels
15. wheelSpeed = 500
16. #individual left speed
17. Lspeed = 0
18. #individual right speed
19. Rspeed = 0
20. #is the thymio reponding to wheel speeds or commands
21. dirCont = False
22. #what is the currently recieved broadcast
23. command = 'null'
24. #moving in an arc [radius,arc_length]
25. arcVar = [0,0]
26. #what sensors are we using
27. accBool = False
28. groundBool = False
29. proxBool = False
30. #arrays for recieving data
31. proxSensorsVal=[0,0,0,0,0,0,0]
32. groundDeltaVal=[0,0]
33. accVal=[0,0,0]
34. circLed = [0,0,0,0,0,0,0,0]
35. #list of recognised commands
36. commandList = ["forward","backward","left","right","null","direct","command","arc
    "]
37. ledCircle =['circ0','circ1','circ2','circ3','circ4','circ5','circ6','circ7']
38. #strign to store commands sent
39. f = open('data.txt','w')
40. f.write("Data\n")
41. lastsent = ""
42.
43. #listen for scratch
44. def scratchReceiver():
45.     global f
46.     while True:
47.         res = s.receive()
48.         #print res
49.         #receive variables
50.         if res[0] == 'sensor-update':
51.             print 'sensor update'
52.             sensor = res[1]
53.             if 'duration' in sensor:
```

```
                global duration
        if sensor['duration']<0:
            duration = 0
        else:
                    duration = sensor['duration']
                print 'duration set'
        f.write(" V:duration")
                print duration
        s.sensorupdate({'duration':duration})
            if 'speed' in sensor:
                global wheelSpeed
                wheelSpeed = sensor['speed']
                print 'speed set'
        f.write(" V:speed")
                print wheelSpeed
        if 'LeftSpeed' in sensor:
                global Lspeed
                Lspeed = sensor['LeftSpeed']
                print 'Lspeed set'
        f.write(" V:Lspeed")
                print Lspeed
        if 'RightSpeed' in sensor:
                global Rspeed
                Rspeed = sensor['RightSpeed']
                print 'Rspeed set'
        f.write(" V:Rspeed")
                print Rspeed
        if 'Radius' in sensor:
                global arcVar
                arcVar[0] = sensor['Radius']
        if arcVar[0] > 0 and arcVar[0] < 1:
            arcVar[0] = 1
            if arcVar[0] < 0 and arcVar[0] > -1:
            arcVar[0] = -1
                print 'radius set'
        f.write(" V:radius")
                print arcVar[0]
        if 'Length' in sensor:
                arcVar[1] = sensor['Length']
                print 'arc length set'
        f.write(" V:length")
                print arcVar[1]
            for x in range(0,8):
                if ledCircle[x] in sensor:
                    circLed[x] = sensor[ledCircle[x]]
        for r in res:
                    #set commands
                    for c in commandList:
                        if r == c:
                            global command
                    global lastsent
                    if duration == 0:
                    if lastsent == c:
                        print 'repeat 0 command'
                    else:
                        command = c
                    else:
                                command = c

                    lastsent=c
                    f.write(" C:" + c)
                    if command == "arc":
                        radius = abs(arcVar[0])
                        arcCenter = abs(arcVar[1])
                        theta = float(arcCenter)/float(radius)
                        arcOuter = theta*(radius+5)
```

```
120.                         arcInner = theta*(radius)
121.                         percent = float(arcInner)/float(arcOuter)
122.                         vel = float(wheelSpeed)*percent / float(100.0/3.0)
123.                         temp = vel/float(abs(arcVar[1]))
124.                         print (1.0/temp)
125.                                 time.sleep(abs(1.0/temp))
126.                         command = 'null'
127.                             elif duration != 0 :
128.                                 time.sleep(abs(duration))
129.                                 command = 'null'
130.
131.         def get_variables_reply_prox(r):
132.             global proxSensorsVal
133.             proxSensorsVal=r
134.
135.         def get_variables_reply_delta(r):
136.             global groundDeltaVal
137.             groundDeltaVal=r
138.
139.         def get_variables_reply_acc(r):
140.             global accVal
141.             accVal=r
142.
143.         def get_variables_error(e):
144.             print 'error:'
145.             print str(e)
146.             loop.quit()
147.
148.         def thymioControl():
149.             network.SendEvent(0,circLed)
150.             if proxBool:
151.                 network.GetVariable("thymio-
    II", "prox.horizontal",reply_handler=get_variables_reply_prox,error_handler=get_v
    ariables_error)
152.                 s.sensorupdate({'prox0': proxSensorsVal[0], 'prox1': proxSensorsVa
    l[1],'prox2': proxSensorsVal[2],'prox3': proxSensorsVal[3],'prox4': proxSensorsVa
    l[4],'prox5': proxSensorsVal[5],'prox6': proxSensorsVal[6]})
153.             if groundBool:
154.                 network.GetVariable("thymio-
    II", "prox.ground.delta",reply_handler=get_variables_reply_delta,error_handler=ge
    t_variables_error)
155.                 s.sensorupdate({'delta0':groundDeltaVal[0],'delta1':groundDeltaVal
    [1]})
156.             if accBool:
157.                 network.GetVariable("thymio-
    II", "acc",reply_handler=get_variables_reply_acc,error_handler=get_variables_erro
    r)
158.                 s.sensorupdate({'acc0':accVal[0],'acc1':accVal[1],'acc2':accVal[2]
    })
159.             s.broadcast("setprox")
160.             global dirCont
161.             if command == "direct":
162.             dirCont = False
163.             if command == "command":
164.             dirCont = True
165.
166.             if dirCont == True:
167.                 if command == "forward":
168.                 network.SetVariable("thymio-
    II", "motor.left.target", [wheelSpeed])
169.                     network.SetVariable("thymio-
    II", "motor.right.target", [wheelSpeed])
170.                 elif command == "left":
171.                 network.SetVariable("thymio-II", "motor.left.target", [-
    wheelSpeed])
```

```
172.                    network.SetVariable("thymio-
    II", "motor.right.target", [wheelSpeed])
173.                elif command == "right":
174.                network.SetVariable("thymio-
    II", "motor.left.target", [wheelSpeed])
175.                    network.SetVariable("thymio-II", "motor.right.target", [-
    wheelSpeed])
176.                elif command == "backward":
177.                network.SetVariable("thymio-II", "motor.left.target", [-
    wheelSpeed])
178.                    network.SetVariable("thymio-II", "motor.right.target", [-
    wheelSpeed])
179.                elif command == "null":
180.                    network.SendEvent(1,[])
181.                network.SetVariable("thymio-II", "motor.left.target", [0])
182.                    network.SetVariable("thymio-II", "motor.right.target", [0])
183.                elif command == "arc":
184.                #calculate the percentage difference between wheels
185.                if arcVar[0] > 0:
186.                radius = arcVar[0]
187.                arcCenter = arcVar[1]
188.                theta = float(arcCenter)/float(radius)
189.                arcOuter = theta*(radius+5)
190.                arcInner = theta*(radius-5)
191.                percent = float(arcInner)/float(arcOuter)
192.                        network.SetVariable("thymio-
    II", "motor.left.target", [wheelSpeed*percent])
193.                        network.SetVariable("thymio-
    II", "motor.right.target", [wheelSpeed])
194.                else:
195.                radius = abs(arcVar[0])
196.                arcCenter = arcVar[1]
197.                theta = float(arcCenter)/float(radius)
198.                arcOuter = theta*(radius+5)
199.                arcInner = theta*(radius-5)
200.                percent = float(arcInner)/float(arcOuter)
201.                        network.SetVariable("thymio-
    II", "motor.left.target", [wheelSpeed])
202.                        network.SetVariable("thymio-
    II", "motor.right.target", [wheelSpeed*percent])
203.                else:
204.                    #set the motors to 0 if there is no command
205.                    network.SetVariable("thymio-II", "motor.left.target", [0])
206.                    network.SetVariable("thymio-II", "motor.right.target", [0])
207.            else:
208.                network.SetVariable("thymio-II", "motor.left.target", [Lspeed])
209.                network.SetVariable("thymio-II", "motor.right.target", [Rspeed])
210.            return True
211.
212.        if __name__ == '__main__':
213.            print 'answer with \'y\' or \'n\''
214.            print 'use proximity sensors: '
215.            if sys.stdin.read(1) == 'y':
216.                proxBool = True
217.            sys.stdin.read(1)
218.            print 'use ground sensors: '
219.            if sys.stdin.read(1) == 'y':
220.                groundBool = True
221.            sys.stdin.read(1)
222.            print 'use acceleration sensors: '
223.            if sys.stdin.read(1) == 'y':
224.                accBool = True
225.            sys.stdin.read(1)
226.            #ask the user for the address of scratch
227.            host = raw_input("What is the IP address of scratch?")
228.
```

```
229.            s = scratch.Scratch(host = host)
230.            s.sensorupdate({'prox0': '0', 'prox1': '0','prox2': '0','prox3': '0','
    prox4': '0','prox5': '0','prox6':'0','delta0':'0','delta1':'0','acc0':'0','acc1':
    '0','acc2':'0'})
231.
232.            parser = OptionParser()
233.            parser.add_option("-s", "--
    system", action="store_true", dest="system", default=False,help="use the system b
    us instead of the session bus")
234.            (options, args) = parser.parse_args()
235.
236.            dbus.mainloop.glib.DBusGMainLoop(set_as_default=True)
237.
238.            if options.system:
239.                bus = dbus.SystemBus()
240.            else:
241.                bus = dbus.SessionBus()
242.
243.            network = dbus.Interface(bus.get_object('ch.epfl.mobots.Aseba', '/'),
    dbus_interface='ch.epfl.mobots.AsebaNetwork')
244.
245.            loop = gobject.MainLoop()
246.
247.            handle = gobject.timeout_add(100,thymioControl)
248.
249.            thread.start_new_thread(scratchReceiver,())
250.            loop.run()
```

8.2 Interface Setup Tutorial

Thymio Scratch interface tutorial

1 Introduction

The Thymio Scratch interface allows scratch to communicate with the Thymio robot so that commands and data can be exchanged. The interface can only be run on a Linux machine, so far the interface has been tested on Ubuntu 14.0.4 and Raspbian on the Raspberry Pi. The interface itself and Scratch do not need to be run on the same machine as long as both computers are on the same network.
This tutorial will cover setting up the interface and how to use Scratch to control the Thymio.

2 Linux notes

If you have never used the Linux terminal before then here are a list of common commands as well as useful commands. Please note that the terminal is case sensitive.
sudo: This commands runs the command as the root user. It is similar to the "run as administrator" option in windows. It is required for most commands which can make some modification to the computer and may require you to enter your password.
apt-get: This is the software manager for Ubuntu and Debian (Raspbian) distributions of Linux. It deals with installing, removing and managing software.
cd: This is used to change the directory that the terminal will use and is the same as in the windows console. This command is used with an address after it.
ls –a/ dir: These commands will list the current contents of the directory. Ls –a will include hidden files as well.

3 Fast Setup

There is a bash script which will install everything for you. If you want to learn about the UNIX terminal then you might want to follow the slow setup, otherwise this will do everything for you.
Firstly you need to download the actual program which will do the work. Go to the following link and click "download zip".
https://github.com/lazerduck/Thymio_python_interface
Now extract the files to somewhere convenient, for this example I will be using the desktop

Open the terminal with "ctrl + alt + T" and in the window you will need to navigate to the folder where you extracted the interface. In this example I use the desktop, refer to the Linux notes section for the commands for navigating using the terminal.

```
cd /home/<your_username>/Desktop/Thymio_python_interface-master
```

Now we need to ensure the file has the permissions to run, enter the following.

```
sudo chmod +x Install.sh
```

With that done you need to run the script, enter the following .

```
./Install.sh
```

```
And now everything has been install.
```

4 Running the Interface

You are now ready to set up the interface. This will describe how to setup the interface tethered to the computer. For remote connections you will need to connect the interface to Scratch on the other computer, section 4.x will cover this in greater detail.

4.1 Run the Required Programs

For the interface to communicate with Scratch and the Thymio they will need to be running. **Connect the Thymio via USB to the computer** and enter the following command to run Asebamedulla and make it connect to the Thymio.

```
sudo asebamedulla "ser:name=Thymio-II"
```

If this doesn't work, try it without the speech marks.

Now the terminal should say that it has connected to the Thymio. If this is not the case, press "ctrl + C" to stop Asebamedulla from running and try again.

Open Scratch, it should appear in the start menu or on the desktop. Use scratch to open the Scratch file in the previously extracted files as mentioned in 3.3. It should be called **"Thymio-test.sb".** The Scratch code should now load and there should be a box saying "remote sensor connections enabled", press okay. There will be code on the cat sprite and on the stage. The code on the cat is example code and the code in the stage is the back end code that performs some simple actions to ensure sensor data is available.

4.2 Run the Interface

You are now ready to run the interface. **Open another terminal window** as the current one is still running Asebamedulla. In the new terminal window you will need to navigate to the folder where you extracted the interface. In this example I use the desktop, refer to the Linux notes section for the commands for navigating using the terminal.

```
cd /home/pi/Desktop/Thymio_python_interface-master
```

In this example, pi is the username and the desktop is where the folder is, **you may need to change this to match your system**. Now that the terminal is looking in the right folder you can run the interface. Enter the following in to the terminal

```
sudo python Thymio_Interface.py
```

It should now ask you what sensors you want to use, just say yes to all 3. Finally it will ask you for the address of the computer running Scratch. As we are running Scratch on the same machine you can just enter “localhost” without the quote marks. Localhost refers to the address of the current machine.

If you come across an error saying
“dbus.exceptions.DBusException: org.freedesktop.DBus.Error.ServiceUnknown: The name ch.epfl.mobots.Aseba was not provided by any .service files”
Then the interface couldn’t find Asebamedulla. Switch to the terminal running Asebamedulla and stop the program and start it running again.
If you see a line saying “sensor update” then everything is working and you are ready to move on to the tasks.

4.3 Remote connections

To use another machine to run Scratch you will need to run Scratch on that machine and open the scratch file just like before. When running the interface instead of entering localhost as the address of scratch enter the local IP of the computer running Scratch.

A remote connection is intended so that a Raspberry Pi with a battery pack can be put on the Thymio and use Wi-Fi to allow you to control it from a distance and with no trailing wires.

8.3 Workshop tasks

Tasks

Here are 6 tasks to try with the Thymio Scratch interface.

For reference, here is a list of all of the broadcasts that the interface can use: **forward, backward, left, right, null, direct, command and arc**. If a command is **not available from the dropdown list then click on new and add it again.**

Press "C" or broadcast command to enable the interface to use the commands for the next few tasks.

Key events have already been included. The arrow keys broadcast in their directions and the space bar broadcasts null. Try using the arrow keys to make the Thymio move. The scroll wheel will act as either up or down in Scratch and commands will queue so be careful not to accidentally queue lots of actions.

1 basic movement

Inside sprite one, use the broadcast tile to make the robot do the following

- Move forward
- Turn right
- Move forward
- Move backwards
- Turn left
- Move backwards

You will need to use the tile as pictured below

Note how all of commands are sent almost straight away and that they the stack up, being performed one after the other.

2 Movement conditions

Repeat the same task as before, this time however, us the set variable to change the duration and the speed. Duration must be positive and duration is in seconds and the speed must be between 500 and -500. After entering the value in the box, make sure to click on the box, a white outline should appear and will indicate it has been run.

Now set the duration to 0 and use the wait function to time broadcasts and the null broadcast to stop the Thymio.

Try to make the Thymio do the following:

- Move forward at speed 500 for 1 seconds
- Rotate left as speed 200 for 0.5 seconds
- Stop for 1 second
- Move backwards at speed -200 for 0.5 seconds
- Stop

Notice how moving backwards at a negative speed causes the Thymio to move forwards.

3 Arc Movement

The Thymio can also be programed to move in an arc. The arc command works differently to the others as it doesn't use the duration. Instead it uses a **radius and length** variable, both in centimetres to define the size of the radius of the arc and to define the distance that should be travelled. **Set the radius to 10 and the length to 10** as well and then broadcast arc. If arc is not in the dropdown list on the broadcast tile then you will have to add it as a new message.

Notice how the wheels move at different speeds in order to make the Thymio move in an arc. The radius is from the centre of the pen hole and the distance to the wheels is 5cm, try setting the radius to 4 and -3.

The arc cannot have a radius of 0, if it did then the centre would never travel and so it would try to move forever.

4 Direct wheel control

You may have noticed the direct and command broadcast. So far we have been using the commands but you can control the wheels individually. Instead of broadcasting, set the **LeftSpeed and RightSpeed variable to control the wheels**. To enable this mode you will need to **press "D" or send the command "direct".**

Using these variables make the Thymio do the following.

- Left speed set to 0, right speed set to 400 for 0.7 seconds
- Left speed set to 500, right speed set to -500 for 1 second
- Left speed set to 100, right speed set to 100 for 2 seconds
- Left speed set to -500, right speed set to 0 for 1 second
- Stop

This mode has no built in stop command so you will have to stop it manually by setting the speeds to 0.

5 Sensors

When you started the interface you were asked what sensors you wanted to use, that was because they are broadcast to Scratch every 10th of a second. In the top right window in Scratch there should be 3 lists. The “prox sensor” list stores all the values from the infra-red sensors along the front and back of the Thymio. The ground list stores the values of the 2 sensor one the underside of the Thymio. These don’t record the distance however but instead use ambient light and reflected light to get a more reliable reading. The larger the value the more likely it is that it has gone off the end. The final list is the accelerometer list. This stores the values of the 3 axis for the accelerometer.

Program the Thymio to move forward when there is an object close behind it.

Program the Thymio to turn when the sensors to the side detect an object.

If you are finding this hard, remember that Scratch is very good at parallel processes. One loop could look for objects and then broadcast a message that would start another process. You can broadcast any message and the “when I receive …” tile to react to that broadcast

6 Avoid objects

Build on the previous task to make the robot move around an area while avoiding objects and determining the best direction to move in when it does encounter an obstacle.

8.4 Consent form

Consent Form

By signing this you give permission for data to be anonymously gathered from your use of the robot interface. The data gathered will be your answers to the questionnaire and a log of commands sent through the interface. The data gathered will be used as research for part of a dissertation. No personal or identifying data will be collected and you are free to withdraw your data at any time.

Contact email: bowes372@gmail.com

Signature _____

Date _____

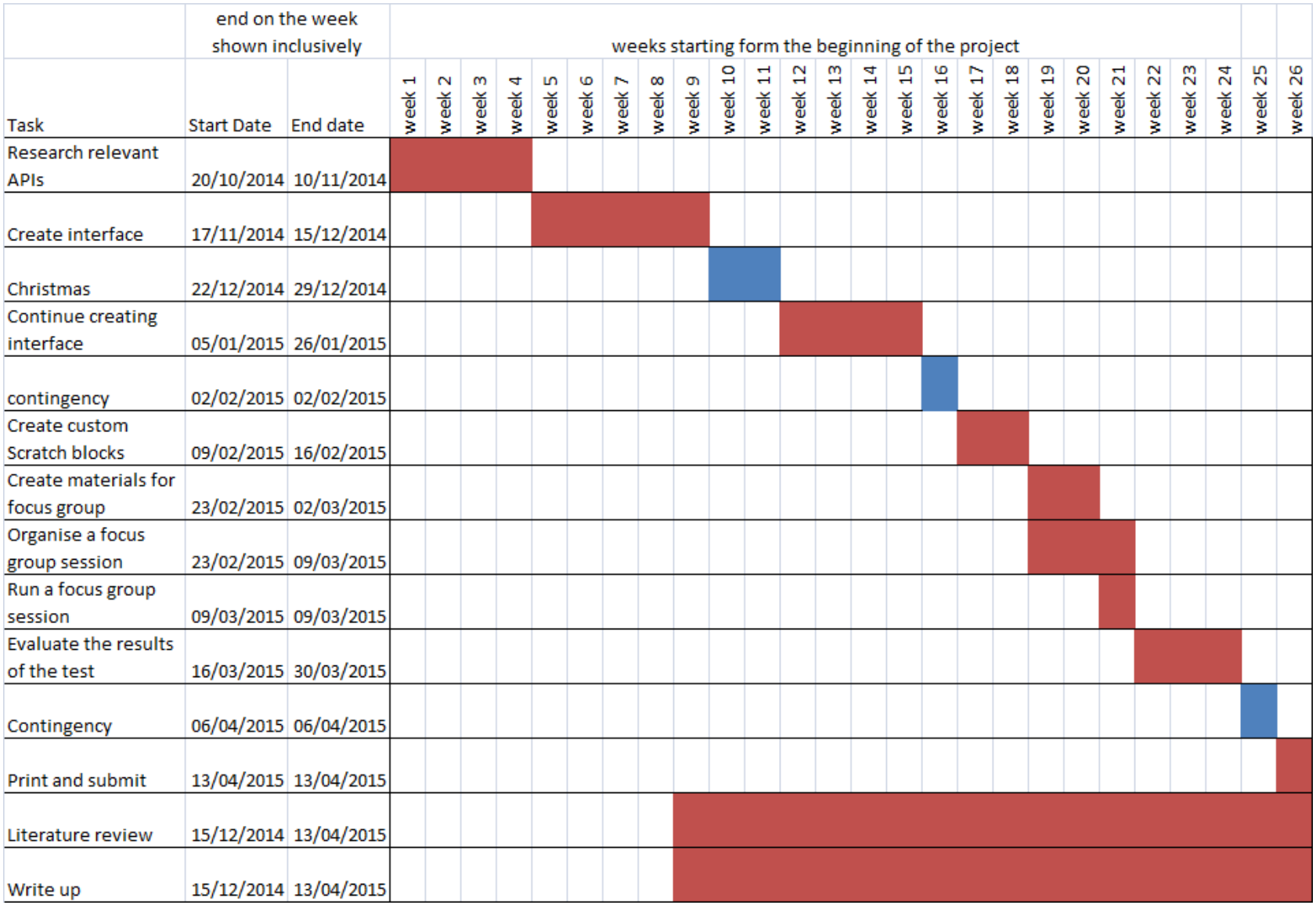
8.5 Questionnaire

Open Day – Robotics Questionnaire

	Strongly agree	Agree	Neutral	Disagree	Strongly disagree
1. The interface is Intuitive	☐	☐	☐	☐	☐
2. Learning to use the interface was easy	☐	☐	☐	☐	☐
3. The tasks tutorial were difficult	☐	☐	☐	☐	☐
4. The tutorial and guidance were clear and useful	☐	☐	☐	☐	☐
5. Using the interface was enjoyable	☐	☐	☐	☐	☐

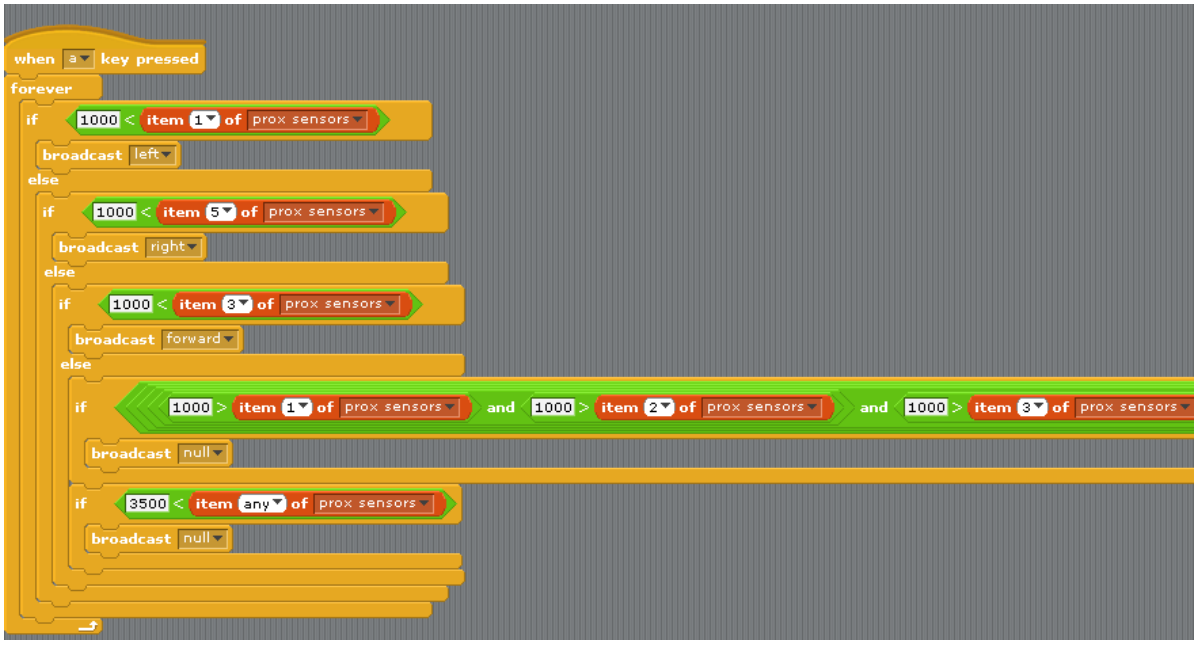
Additional comments

8.6 Gantt Chart



Task	Start Date	End date
Research relevant APIs	20/10/2014	10/11/2014
Create interface	17/11/2014	15/12/2014
Christmas	22/12/2014	29/12/2014
Continue creating interface	05/01/2015	26/01/2015
contingency	02/02/2015	02/02/2015
Create custom Scratch blocks	09/02/2015	16/02/2015
Create materials for focus group	23/02/2015	02/03/2015
Organise a focus group session	23/02/2015	09/03/2015
Run a focus group session	09/03/2015	09/03/2015
Evaluate the results of the test	16/03/2015	30/03/2015
Contingency	06/04/2015	06/04/2015
Print and submit	13/04/2015	13/04/2015
Literature review	15/12/2014	13/04/2015
Write up	15/12/2014	13/04/2015

8.7 Workshop demonstration code



9 Reference List

1. Ben-Ari, M. (1998) "Constructivism in computer science education." *Acm sigcse bulletin*, vol. 30, no. 1, pp. 257-261. ACM.
2. Brusilovsky, P., Calabrese, E., Hvorecky, J., Kouchnirenko, A., and Miller, P. (1997) Mini-languages: A Way to Learn Programming Principles. *Education and Information Technologies* **2** (1), pp. 65-83.
3. Cielniak, G., Bellotto, N. and Duckett, T. (2013) Integrating mobile robotics and vision with undergraduate computer science. Education, IEEE Transactions on 56(1) 48-53.
4. Cms, (2005) Selecting Developing Approach. [Online] Available from: www.cms.gov/Research-Statistics-Data-and-Systems/CMS-Information-Technology/XLC/Downloads/SelectingDevelopmentApproach.pdf [Accessed 18 October 2014].
5. Fesakis, G. and Kiriaki, S. (2009) Influence of the familiarization with Scratch on future teachers' opinions and attitudes about programming and ICT in education. ACM SIGCSE Bulletin, 41 (3) 258-262.
6. Harvey, B. and Mönig, J. (2010). Bringing “No ceiling” to Scratch: can one language serve kids and computer scientists. Proc. Constructionism.
7. Kaplinsky, K. V. (2001) VNC tight encoder-data compression for VNC. In*Modern Techniques and Technology, 2001. MTT 2001. Proceedings of the 7th International Scientific and Practical Conference of Students, Post-graduates and Young Scientists* 155-157.
8. Lewis, J. R. (1995). IBM computer usability satisfaction questionnaires: psychometric evaluation and instructions for use. *International Journal of Human-Computer Interaction*, *7*(1), 57-78.
9. Malan, J. and Leitner, H. (2007) Scratch for Budding Computer Scientists. ACM SIGCSE Bulletin 39(1) 223-227.
10. Maloney, J., Resnick, M., Rusk, N.,Silverman, B. and Eastmond, E. (2010) The scratch programming, ACM Transactions on Computing Education 10 (4) 16.
11. Pilliq (2014) *Scratchpy* [software] GitHub. Available from https://github.com/pilliq/scratchpy [Accessed 03/04/2015].
12. Retornaz, P., Riedo, F., Magnenat, S., Vaussard, F., Bonani, M. and Mondada, F. (2013) Seamless Multi-Robot Programming for the People: ASEBA and the Wireless Thymio II Robot. Information and Automation, No. EPFL-CONF-188402 337-343.

13. Riano, L. and McGinnity, T. (2010) Design and validation of a robotic system to interactively teach geometry. AT&P Journal (2) 91-96
14. Sanner, M. F. (1999). Python: a programming language for software integration and development. *J Mol Graph Model*, *17*(1), 57-61.
15. Sebastian Wilmet, (2005) Gedit. [Online] https://wiki.gnome.org/Apps/Gedit [Accessed 12 April 2015].
16. Shin, J. and Magnenat, S. (2014). Visual programming language for Thymio II robot. ETH-Zürich.
17. Shin, S., Park, P., & Bae, Y. (2013). The Effects of an Information-Technology Gifted Program on Friendship Using Scratch Programming Language and Clutter. International Journal of Computer and Communication Engineering, Vol. 2, No. 3.
18. Wysocki, RK. (2012). [Ebook] :Effective Project Management : Traditional, Agile, Extreme / Robert K. Wysocki, n.p.: Hoboken, N.J. : Wiley, c2012., University of Lincoln Library Catalogue, EBSCOhost, viewed 26 October 2013.