Educational robots for teaching programming to youths

Adam Bowes

Date

# Abstract

 **What you set out to do and why**

 **How you did it**

 **What you found**

[!]

# Table of Contents

[Abstract 2](#_Toc416130963)

[Table of Contents 3](#_Toc416130964)

[1 Introduction 5](#_Toc416130965)

[2 Literature Review 6](#_Toc416130966)

[2.1 Introduction 6](#_Toc416130967)

[2.2 Education 6](#_Toc416130968)

[2.3 Programming 6](#_Toc416130969)

[2.4 Robotics 7](#_Toc416130970)

[2.5 Conclusion 7](#_Toc416130971)

[3 Methodology 9](#_Toc416130972)

[3.1 Project management 9](#_Toc416130973)

[3.1.1 Software Development 9](#_Toc416130974)

[3.1.2 Gantt chart 10](#_Toc416130975)

[3.1.3 Risk assessment 10](#_Toc416130976)

[3.2 Toolsets 12](#_Toc416130977)

[3.2.1 Thymio-II 12](#_Toc416130978)

[3.2.2 Aseba 13](#_Toc416130979)

[3.2.3 Scratchpy 13](#_Toc416130980)

[3.2.4 Scratch 1.4 14](#_Toc416130981)

[3.2.5 Python 2.7.9 14](#_Toc416130982)

[3.2.6 Gedit and Idle 15](#_Toc416130983)

[3.2.7 Microsoft Office 16](#_Toc416130984)

[3.2.8 GitHub 16](#_Toc416130985)

[3.3 Research methods 16](#_Toc416130986)

[4 Design 18](#_Toc416130987)

[4.1 Introduction 18](#_Toc416130988)

[4.2 Scratch 1.4 communication 18](#_Toc416130989)

[4.3 Thymio-II communication 19](#_Toc416130990)

[4.4 Interface 19](#_Toc416130991)

[4.4.1 Communication 19](#_Toc416130992)

[4.4.2 Thymio-II Control modes 20](#_Toc416130993)

[5 Development 22](#_Toc416130994)

[5.1 Introduction 22](#_Toc416130995)

[5.2 Python Interface implementation 22](#_Toc416130996)

[5.2.1 Multiple threads 22](#_Toc416130997)

[5.2.2 Connection to the Thymio-II 22](#_Toc416130998)

[5.2.3 Connecting to Scratch 1.4 23](#_Toc416130999)

[5.2.4 Commands 24](#_Toc416131000)

[5.2.5 Receiving variables from Scratch 1.4 24](#_Toc416131001)

[5.2.6 Arc movement 25](#_Toc416131002)

[5.3 Scratch 1.4 Interface implementation 26](#_Toc416131003)

[5.4 Setup 26](#_Toc416131004)

[5.5 Tutorial and workshop tasks 27](#_Toc416131005)

[6 Evaluation 29](#_Toc416131006)

[7 Critical Reflection 30](#_Toc416131007)

[8 Appendices 31](#_Toc416131008)

[8.1 Python interface code 31](#_Toc416131009)

[8.2 Interface Setup Tutorial 38](#_Toc416131010)

[8.3 Workshop tasks 41](#_Toc416131011)

[8.4 Consent form 44](#_Toc416131012)

[8.5 Questionnaire 45](#_Toc416131013)

[8.6 Gantt Chart 46](#_Toc416131014)

[9 References 47](#_Toc416131015)

# 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.

* 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

## 2.1 Introduction

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 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.

## 2.2 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).

## 2.3 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)

## 2.4 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)

## 2.5 Conclusion

Several conclusions can be drawn from this research. Firstly, that robots used in education can lead to increased practical achievement and can motivate students to go beyond the constraints of an assignment. Secondly, that Scratch is a very powerful but simple language with a majority of standard features while still remaining user friendly and accessible to beginner programmers. As a result of this it can be used to introduce a variety of people to programming and to reduce their anxiety and discomfort with programming. Thirdly, that the Thymio II is the second iteration of a robot produced through extensive user feedback and as a result is very capable and adaptable platform.

# 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 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 contingence 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 Abesa. |
| 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:

* Thymio-II robot
* Aseba
* Scratchpy
* Scratch 1.4
* Python 2.7.9
* Gedit
* GitHub
* Microsoft office suite

### 3.2.1 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.

### 3.2.2 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.

### 3.2.3 Scratchpy

Scratchpy is an open source python module which manages connections to Scratch 1.4 over TCP/IP (Pilliq 2014). The interface 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.

### 3.2.4 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.

### 3.2.5 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.

### 3.2.6 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. 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 editor 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. 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.

### 3.2.7 Microsoft Office

As part of a study a questionnaire and a consent form will be required. 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 include Microsoft Excel which is a spreadsheet 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 Libre Office, a free office suite similar to Microsoft Office. Libre Office contains both a spreadsheet program and a word processor but lack the formatting options of Word and the interconnectivity that Word and Excel share.

### 3.2.8 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

[!]

In order to evaluate the success of the project the research will need to be carried out to determine the usability and the user friendliness of the interface. To investigate these properties a substantial number of people will need to use the interface along with the accompanying tutorials.

[!]

To determine the success of the project the usability and the user friendliness of the interface produced needs to be evaluated. 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 will use the software to complete a series of tutorials and task, they will then be handed a standardised questionnaire in which they will be asked about their experience with the interface. The results will then be compiled and analysed.

# 4 Design

## 4.1 Introduction

In this section the components of the interface will be broken down and the external communication options will be analysed. After this the design of the Interface will be presented.

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.

## 4.2 Scratch 1.4 communication

Internally Scratch 1.4 uses broadcasts which sends 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.

## 4.3 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.

## 4.4 Interface

### 4.4.1 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 form 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 1.

Interface

Receive messages from Scratch 1.4

Send messages to Scratch 1.4 and the Thymio-II

Receive data from the Thymio-II

Scratch 1.4 Remote sensor connections

Thymio-II

Figure : A representation of the interface structure

### 4.4.2 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

## 5.1 Introduction

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 wireless use of the interface. The last section will cover the tutorial and workshop tasks. The final code for the interface can be seen in appendices 8.1.

## 5.2 Python Interface implementation

### 5.2.1 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.

### 5.2.2 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 2)

Figure : 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.



Figure : 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 3). Finally receiving a variable requires an error handler which will receive any errors that occur.

### 5.2.3 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.

### 5.2.4 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.

### 5.2.5 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.

### 5.2.6 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 cause 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 4).

Figure : The parameters of the arc movement

Thymio-II

Radius

Direction of travel

Inner circle

Outer circle

### 5.3 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 5).

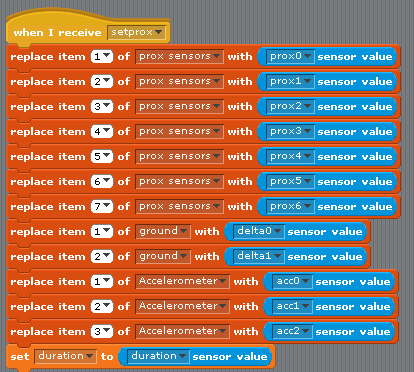


Figure : The function used to sort the sensor data

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 sensor are affected by an action.

### 5.4 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. 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.

### 5.5 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

6.1 Introduction

6.2

[!]

# 7 Critical Reflection

[!]

# 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
9. **from** optparse **import** OptionParser
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 = ""
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:
54. **global** duration
55. **if** sensor['duration']<0:
56. duration = 0
57. **else**:
58. duration = sensor['duration']
59. **print** 'duration set'
60. f.write(" V:duration")
61. **print** duration
62. s.sensorupdate({'duration':duration})
63. **if** 'speed' **in** sensor:
64. **global** wheelSpeed
65. wheelSpeed = sensor['speed']
66. **print** 'speed set'
67. f.write(" V:speed")
68. **print** wheelSpeed
69. **if** 'LeftSpeed' **in** sensor:
70. **global** Lspeed
71. Lspeed = sensor['LeftSpeed']
72. **print** 'Lspeed set'
73. f.write(" V:Lspeed")
74. **print** Lspeed
75. **if** 'RightSpeed' **in** sensor:
76. **global** Rspeed
77. Rspeed = sensor['RightSpeed']
78. **print** 'Rspeed set'
79. f.write(" V:Rspeed")
80. **print** Rspeed
81. **if** 'Radius' **in** sensor:
82. **global** arcVar
83. arcVar[0] = sensor['Radius']
84. **if** arcVar[0] > 0 **and** arcVar[0] < 1:
85. arcVar[0] = 1
86. **if** arcVar[0] < 0 **and** arcVar[0] > -1:
87. arcVar[0] = -1
88. **print** 'radius set'
89. f.write(" V:radius")
90. **print** arcVar[0]
91. **if** 'Length' **in** sensor:
92. arcVar[1] = sensor['Length']
93. **print** 'arc length set'
94. f.write(" V:length")
95. **print** arcVar[1]
96. **for** x **in** range(0,8):
97. **if** ledCircle[x] **in** sensor:
98. circLed[x] = sensor[ledCircle[x]]
99. **for** r **in** res:
100. #set commands
101. **for** c **in** commandList:
102. **if** r == c:
103. **global** command
104. **global** lastsent
105. **if** duration == 0:
106. **if** lastsent == c:
107. **print** 'repeat 0 command'
108. **else**:
109. command = c
110. **else**:
111. command = c
113. lastsent=c
114. f.write(" C:" + c)
115. **if** command == "arc":
116. radius = abs(arcVar[0])
117. arcCenter = abs(arcVar[1])
118. theta = float(arcCenter)/float(radius)
119. 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'
131. **def** get\_variables\_reply\_prox(r):
132. **global** proxSensorsVal
133. proxSensorsVal=r
135. **def** get\_variables\_reply\_delta(r):
136. **global** groundDeltaVal
137. groundDeltaVal=r
139. **def** get\_variables\_reply\_acc(r):
140. **global** accVal
141. accVal=r
143. **def** get\_variables\_error(e):
144. **print** 'error:'
145. **print** str(e)
146. loop.quit()
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\_variables\_error)
152. s.sensorupdate({'prox0': proxSensorsVal[0], 'prox1': proxSensorsVal[1],'prox2': proxSensorsVal[2],'prox3': proxSensorsVal[3],'prox4': proxSensorsVal[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=get\_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\_error)
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
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
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?")
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'})
232. parser = OptionParser()
233. parser.add\_option("-s", "--system", action="store\_true", dest="system", default=False,help="use the system bus instead of the session bus")
234. (options, args) = parser.parse\_args()
236. dbus.mainloop.glib.DBusGMainLoop(set\_as\_default=True)
238. **if** options.system:
239. bus = dbus.SystemBus()
240. **else**:
241. bus = dbus.SessionBus()
243. network = dbus.Interface(bus.get\_object('ch.epfl.mobots.Aseba', '/'), dbus\_interface='ch.epfl.mobots.AsebaNetwork')
245. loop = gobject.MainLoop()
247. handle = gobject.timeout\_add(100,thymioControl)
249. thread.start\_new\_thread(scratchReceiver,())
250. loop.run()

## 8.2 Interface Setup Tutorial





## 8.3 Workshop tasks







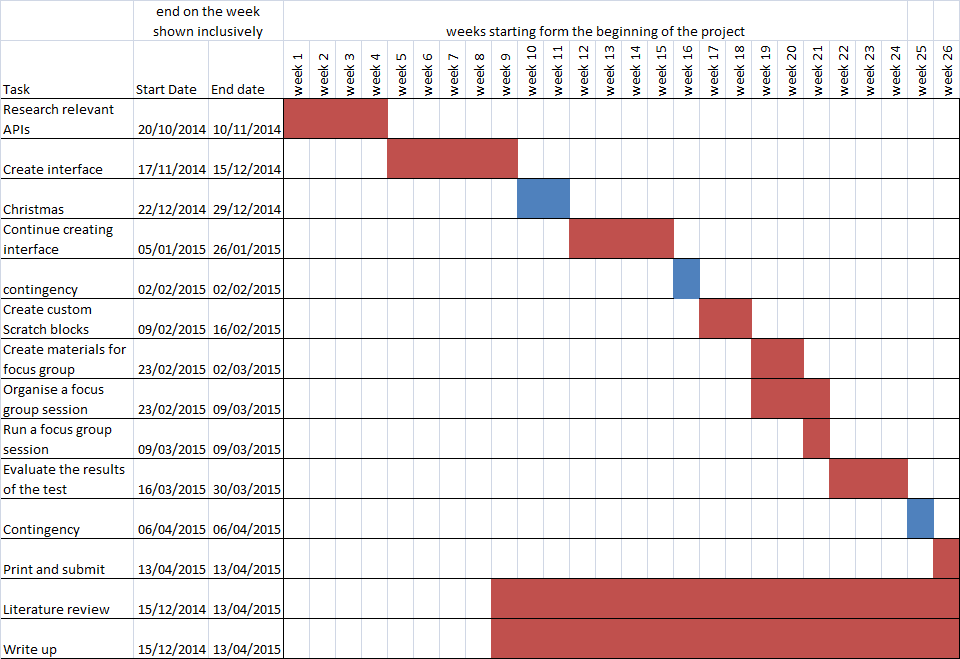
## 8.4 Consent form



## 8.5 Questionnaire



## 8.6 Gantt Chart



## 9 Reference List

Riano, L. and McGinnity, T. (2010) Design and validation of a robotic system to interactively teach geometry. AT&P Journal (2) 91-96

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].

Malan, J. and Leitner, H. (2007) Scratch for Budding Computer Scientists. ACM SIGCSE Bulletin 39(1) 223-227.

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.

Harvey, B. and Mönig, J. (2010). Bringing “No ceiling” to Scratch: can one language serve kids and computer scientists. Proc. Constructionism.

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.

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.

Maloney, J., Resnick, M., Rusk, N.,Silverman, B. and Eastmond, E. (2010) The scratch programming language and environment, ACM Transactions on Computing Education 10 (4) 16.

Shin, J. and Magnenat, S. (2014). Visual programming language for Thymio II robot. ETH-Zürich.

Pilliq (2014) *Scratchpy* [software] GitHub. Available from https://github.com/pilliq/scratchpy [Accessed 03/04/2015].

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.

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.

Ben-Ari, M. (1998) "Constructivism in computer science education." *Acm sigcse bulletin*, vol. 30, no. 1, pp. 257-261. ACM.

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.

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.