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**

Do this last

# Table of Contents

[Abstract 2](#_Toc413512945)

[Table of Contents 3](#_Toc413512946)

[Aims and Objectives 4](#_Toc413512947)

[Literature Review 5](#_Toc413512948)

[Methodology 6](#_Toc413512949)

[Evaluation 7](#_Toc413512950)

[Critical Reflection 8](#_Toc413512951)

[Appendices 9](#_Toc413512952)

# Aims and Objectives

Note - Not sure if this is needed or the format.

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 I have created an interface to allow the use of Scratch, a user friendly programming language to control a Thymio-II robot to allow users to have their programs affect the real world.

In order to achieve this I have completed several objectives.

# Literature Review

# Methodology

## Project management

Agile/learn/extreme

## Software Development

-software development methodology  
-characteristics of the software and the computer environments available.  
-simplistic review of strengths and weaknesses

**Needs references?**

This project required a piece of software to be developed. This software took form as an interface allowing communications between the programming language Scratch 1.4 and the Thymio-II robot. To create this interface the incremental software development method was applied. Incremental development is a variation on the waterfall method which consists of multiple waterfalls with reviews between each allowing waterfall to be used on flexible projects.

This method was chosen after considering the characteristics of the software being built and the environment in which it was constructed. Developing the interface required overcoming unique challenges which were largely unique to this type of software, as a result some segments of the code needed to be rewritten and 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. 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

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].

# Implementation

## Introduction

This section will explain the development process of the project and will go over the steps taken to create and use the interface. In addition any issues encountered will be described along with their solutions. The interface will be explained first followed by the theory behind connecting the interface to the robot.

## Interface implementation

The first task that had to be undertaken was enabling Python to communicate to the Thymio. After some research it was clear that Asebamedulla could be used to allow a D-Bus connection to communicate with the Thymio-II. Using Pythons D-Bus module the code could connect with the Thymio-II via Asebamedulla using a D-Bus Interface and the Asebamedulla D-Bus name (figure [X]).

Figure 1: The D-Bus network being created

Now there was a connection to Asebamedulla which would then send our data through to the Thymio-II robot which would apply them as appropriate. The Python code now needs to retrieve and store the current values from the Thymio. To do this a loop was created that would send a command every 100 milliseconds telling the Thymio to send the current values from its proximity sensors. The sensor data could be accessed using the previously declared network variable along with the GetVariable command. The GetVariable command required several parameters to function. Firstly it needs the device to be specified which in this case is the Thymio-II. Next the variable that is meant to be returned is set, for the proximity sensor the string “prox.horizontal” is sent. The next parameter is a function to handle the data and finally there needs to be a function to handle any errors. The variable handling functions make use of variables declared with appropriate types meaning that for each sensor type the data can be stored directly in to a variable. The error handling receives an error message in the form of a string and cause the current code to quit (figure [X]).

Figure 2: The variable handling for receiving a variable

Due to the nature of Python variables can be declared with a type and then referred to outside of their initial scope by using the key word “global”. The proxSensorVal holds an array of 7 floating point variables where each is the current distance the corresponding proximity sensor. The groundDeltaVal stores 2 values for the downward facing proximity sensors which are used to tell if there is a surface below the front of the robot. The accVal stores 3 floating point variables, each of which representing an axis from the Thymio-II’s accelerometer.

Before the interface and send commands the code first needs a way of dealing with user input. An array of strings is used where each string is the text for a command (e.g. “forward”). This array is used to set a single string to a variable called command which stores the current command. As the loop progresses a series of if statements check for the different commands and runs the appropriate command.

The Thymio-II deals with movement by setting a target speed for each wheel and then automatically tries to achieve this. The interface can take advantage of this by sending a command to set the target speed variables to the required speed instead of trying to access functions on the Thymio-II. The command to set the variables makes use of the network object and is called SetVariable. This function is similar to the GetVariable function in that it requires the device and the variable name but then you send the value you want to set the motor to (figure [X]).

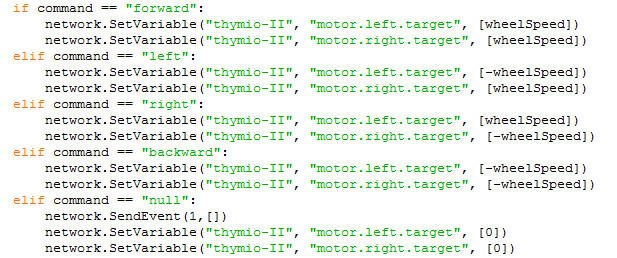


Figure 3: Setting wheel target speed depending on command

Now that data has been received and commands can be sent the interface now needs to communicate with Scratch 1.4. Scratch 1.4 has a built in server for robot communication which allows values to be read in. On top of this, with the server enabled Scratch 1.4 will send all broadcasts and variable changes over the network. To connect to this the Python code will make use of a python module called scratch. This module connects to Scratch 1.4 and handles all data traveling between the interface and Scratch 1.4. To initialise the connection to Scratch 1.4 a Scratch object is made, during this the host IP address is given (the address to the machine running scratch) and the module establishes the link. At this point any broadcast made in Scratch 1.4 will be sent and added to the back of the received commands queue.

As the main loop needs to keep running to update sensor data

1 connect to Thymio -  
2 receive data -  
3 send commands -  
4 connect to scratch -  
5 Receive commands from scratch  
6 transfer scratch commands into Thymio commands  
7 transfer thymio data to scratch  
8

# Evaluation

# Critical Reflection

# Appendices