|  |  |  |
| --- | --- | --- |
| ju | | **Description** |
| Author(s)  Jokitulppo, Matti | Type of publication  Bachelor’s thesis | Date dd.mm.yyyy |
| Language of publication:  English |
| Number of pages | Permission for web publication: x |
| Title of publication  **Arduino-controlled robot** | | |
| Degree programme  Software engineering | | |
| Tutor(s)  Manninen, Pasi  Kotkansalo, Jouko | | |
| Assigned by  JAMK University of Applied Sciences, Manninen, Pasi | | |
| Abstract  The aim of this thesis was to create a simple Bluetooth controlled robot for JAMK University of Applied Sciences for marketing purposes. The robot is controlled via a Bluetooth LE link, from any Android phone or tablet using a custom application. The robot itself is controlled by an Arduino microcontroller, which in turn is controlled by the mobile application via a custom communications protocol.  This thesis contains a throughout description of the build process involved in the robot project, along with details and descriptions about the technological choices made on the assignment. Implementation of the project began first by researching possible technologies and sketching out the features the final product might have. After getting a clear picture of the desired end product, necessary components were ordered and assembled into a rough prototype. The Android mobile application and the embedded Arduino code were developed in tandem, implementing new features on both platforms as the time arose.  As a result of this thesis, the customer (JAMK) received a fully functional robot with companion app, which could easily be demonstrated in fairs or other similar events. | | |
| Keywords/tags ([subjects](http://www.nelliportaali.fi/V/?institute=JAMK&portal=JAMK&new_lng=eng&force_login=Y&func=find-db-1-category&mode=category&restricted=all&sequence=000013943))  Arduino, Bluetooth, Bluetooth LE, Android, Robot | | |
| Miscellaneous | | |

|  |  |  |
| --- | --- | --- |
|  | | **Kuvailulehti** |
| Tekijä(t)  Jokitulppo, Matti | Julkaisun laji  Opinnäytetyö | Päivämäärä pp.kk.vvvv |
| Sivumäärä | Julkaisun kieli  Englanti |
|  | Verkkojulkaisulupa myönnetty: x |
| Työn nimi  **Arduino-controlled robot** | | |
| Koulutusohjelma  Ohjelmistotekniikan koulutusohjelma | | |
| Työn ohjaaja(t)  Manninen, Pasi  Kotkansalo, Jouko | | |
| Toimeksiantaja(t)  Jyväskylän ammattikorkeakoulu, Manninen, Pasi | | |
| Tiivistelmä  Tämän työn tavoitteena oli luoda yksinkertainen Bluetooth-kommunikaatiostandardin ylitse kontrolloitava robotti Jyväskylän Ammattikorkeakoululle markkinointitarkoituksiin. Robottia voidaan käskyttää millä tahansa Android-yhteensopivilla laitteella mobiilisovellusta hyödyntäen. Robottia itseään ohjaa Arduino-mikropiiri, jota sovellus käskyttää omalla protokollallaan muodostettuaan Bluetooth-linkin.  Tämä opinnäytetyö sisältää kattavan kuvauksen robottiprojektin rakennusprosessista, mukaanlukien kuvauksia projektin aikana tehdyistä teknologisista valinnoista. Toteutus aloitettiin aiemman prototyypin pohjalta tutkimalla uuteen robottiin tarvittavia ominaisuuksia ja komponentteja. Halutun lopputuloksen selkeennyttyä tarvittavat osat tilattiin ja koottiin valmiiksi lopputuotteeksi. Android-applikaatio ja robotin sulautettu sovellus tehtiin samanaikaisesti, toteuttaen uusia ominaisuuksia molemmilla alustoille tarpeen vaatiessa  Tämän työn tuloksena tilaaja (Jyväskylän ammattikorkeakoulu) sai toimivan robotin ja mobiilisovelluksen, jonka avulla robottia voidaan helposti käskyttää ja esitellä messuilla tai muissa vastaavissa tilaisuuksissa. | | |
| Avainsanat ([asiasanat](http://vesa.lib.helsinki.fi/))  Arduino, Bluetooth, Bluetooth LE, Android, Robotti | | |
| Muut tiedot | | |

Contents

[Glossary 4](#_Toc426402140)

[1 Introduction 7](#_Toc426402141)

[1.1 Project background and assignment 7](#_Toc426402142)

[1.2 The previous prototype 8](#_Toc426402143)

[1.3 Objectives for the new project 11](#_Toc426402144)

[1.4 Tools and technologies 12](#_Toc426402145)

[1.4.1 Arduino IDE 12](#_Toc426402146)

[1.4.2 Eclipse 13](#_Toc426402147)

[1.4.3 C++ and the Arduino language 13](#_Toc426402148)

[1.4.4 Java 13](#_Toc426402149)

[2 The Android application 14](#_Toc426402150)

[2.1 Introduction 14](#_Toc426402151)

[2.2 Activities 15](#_Toc426402152)

[2.2.1 Settings activity 15](#_Toc426402153)

[2.2.2 Robot feed activity 17](#_Toc426402154)

[2.3 Application classes 19](#_Toc426402155)

[2.3.1 MjpegInputStream & MjpegView 19](#_Toc426402156)

[2.3.2 JoyStickView 19](#_Toc426402157)

[2.3.3 ApplicationState 19](#_Toc426402158)

[2.3.4 BluetoothStreamManager 20](#_Toc426402159)

[2.3.5 Feed 20](#_Toc426402160)

[2.3.6 Settings 21](#_Toc426402161)

[3 The robot 22](#_Toc426402162)

[3.1 Introduction 22](#_Toc426402163)

[3.2 Overview of parts 23](#_Toc426402164)

[3.2.1 Arduino Mega 2560 23](#_Toc426402165)

[3.2.2 Robot chassis 24](#_Toc426402166)

[3.2.3 Pan + tilt servo 24](#_Toc426402167)

[3.2.4 LiPo battery and UBEC 25](#_Toc426402168)

[3.2.5 Adafruit Motor Shield V2 25](#_Toc426402169)

[3.2.6 NeoPixel shield & matrix 26](#_Toc426402170)

[3.2.7 Emic 2 Text-to-speech module & speaker 26](#_Toc426402171)

[3.2.8 Raspberry Pi 2 with webcam 26](#_Toc426402172)

[3.3 Assembly 27](#_Toc426402173)

[4 The embedded code 28](#_Toc426402174)

[4.1 Introduction 28](#_Toc426402175)

[4.2 Communication between Android application and robot 29](#_Toc426402176)

[5 Results and conclusion 31](#_Toc426402177)

[References 32](#_Toc426402178)

**Figures**

Figure 1. Robot prototype, powered off and on 6

Figure 2. The pin control view 8

Figure 3. Top-down view of wiring 9

Figure 4. Android settings activity 11

Figure 5. Robot feed activity 13

**Tables**

[Table 1. Robot parts & prices 21](#_Toc425785902)

# Glossary

**Arduino**

Arduino is a series of popular open-source microcontroller. Programmed using C or C++, it is intended to be an easy and inexpensive way for students and hobbyists to ease themselves into electronics. There exists a large variety of different add-ons and components to enhance the Arduino’s functionality.

**Bluetooth LE**

**Bluetooth Low Energy, also known as Bluetooth Smart, is a new, less power-intensive version of the existing Bluetooth standard, known today as Bluetooth Classic. Besides saving battery life, Bluetooth LE also promises less latency for most applications.**

**DC Motor**

A DC motor is a type of motor with a magnetic coil inside. When an electric current passes through the coil, the magnetic force generated proceeds to turn the motor. The current is passed through a commutator before entering the coil, which switches the direction of the current at the apex point, so the spinning continues. The speed of the motor can be controlled by limiting the current, and the direction is affected by the direction of the current.

**IDE**

Integrated development environment

**LiPo Battery**

**Lithium polymer batteries are a type of batteries very popular in the world on remote controlled planes. They’re lightweight, hold a lot of capacity and are capable of discharging this capacity very quickly. (Understanding RC LiPo Batteries, 2015)**

**NeoPixel**

NeoPixel is a brand of individually addressable full-color RGB LEDs that can be controlled by a single input of a microcontroller.

**Raspberry Pi**

The Raspberry Pi is a small, cheap single-board computer. Originally meant as a teaching aid, it has gained much popularity in the hobbyist electronics community as being a portable, cheap solution to adding computing power to embedded projects. It runs it’s own modified distribution of the GNU/Linux operating system.

**RSSI**

**RSSI, short for ”Received Signal Strength Indicator” is the strength of a wireless connection, usually measured in dBm from 0 to -120db. The closer the value is to zero, the stronger the signal. (http://www.speedguide.net/faq/how-does-rssi-dbm-relate-to-signal-quality-percent-439)**

**Servo**

A servo is a special type of motor that’s controlled by electronic pulses of varying lengths. The timing of these pulses tells the servo which position it should move to. Generating these pulses is extremely timing dependent, and therefore controlling them through normal computers is not recommended. Since operating systems with many processes taking up different amounts of resources at times, it’s not guaranteed a length of code is always run at the same exact interval. This can cause the servo to jitter, since the timing may be off by a few tenths of a millisecond.

Most servos have a limited range of movement, usually from 0 to 180 degrees. There are many different kinds of servos on the market, from cheap and tiny ones to models costing up to hundreds of dollars, with features such as acceleration and temperature tracking and their own microcontrollers built in.

**UART**

UART is a two-way serial communication protocol, which means there’s no feasible way to communicate between more than two chips at the same time. It has two pins, TX (transmit) and RX (receive). Since there’s no clock line, the data transfer amount per second must be negotiated in advance, before starting communications.

**Raspberry Pi**

The Raspberry Pi is a small, cheap single-board computer. Originally meant as a teaching aid, it has gained much popularity in the hobbyist electronics community as being a portable, cheap solution to adding computing power to embedded projects. It runs its own modified distribution of the GNU/Linux operating system.

**Servo**

A servo is a special type of motor that’s controlled by electronic pulses of varying lengths. The timing of these pulses tells the servo which position it should move to. Generating these pulses is extremely timing dependent, and therefore controlling them through normal computers is not recommended. Since operating systems with many processes taking up different amounts of resources at times, it’s not guaranteed a length of code is always run at the same exact interval. This can cause the servo to jitter, since the timing may be off by a few tenths of a millisecond.

Most servos have a limited range of movement, usually from 0 to 180 degrees. There are many different kinds of servos on the market, from cheap and tiny ones to models costing up to hundreds of dollars, with features such as acceleration and temperature tracking and their own microcontrollers built in.

# Introduction

## Project background and assignment

In the winter of 2014 during an Android programming course held at JAMK University of Applied Sciences, an early prototype of the robot was built as the course’s mandatory final project. After the course senior lecturer Pasi Manninen, on behalf of JAMK, suggested a new version be built specifically for JAMK. This new robot could be then used for marketing purposes in fairs, career days and such.

The new and improved robot would be built mostly with the same basic concepts as the earlier version. It would consist mostly of an Arduino microcontroller on top of a basic two-wheel hobbyist robot chassis. A small computer would provide a live video feed via an on-board camera mounted on a servo platform. The robot could be driven using an Android phone or tablet or other compatible device.

Due to the request of Mr. Manninen acting in the role of the customer, some of the more “technically oriented” features of the earlier robot could be cut on the whim of the builder. The main qualities the new prototype should focus on were agreed to be ease-of-use and visual impressiveness.

## The previous prototype

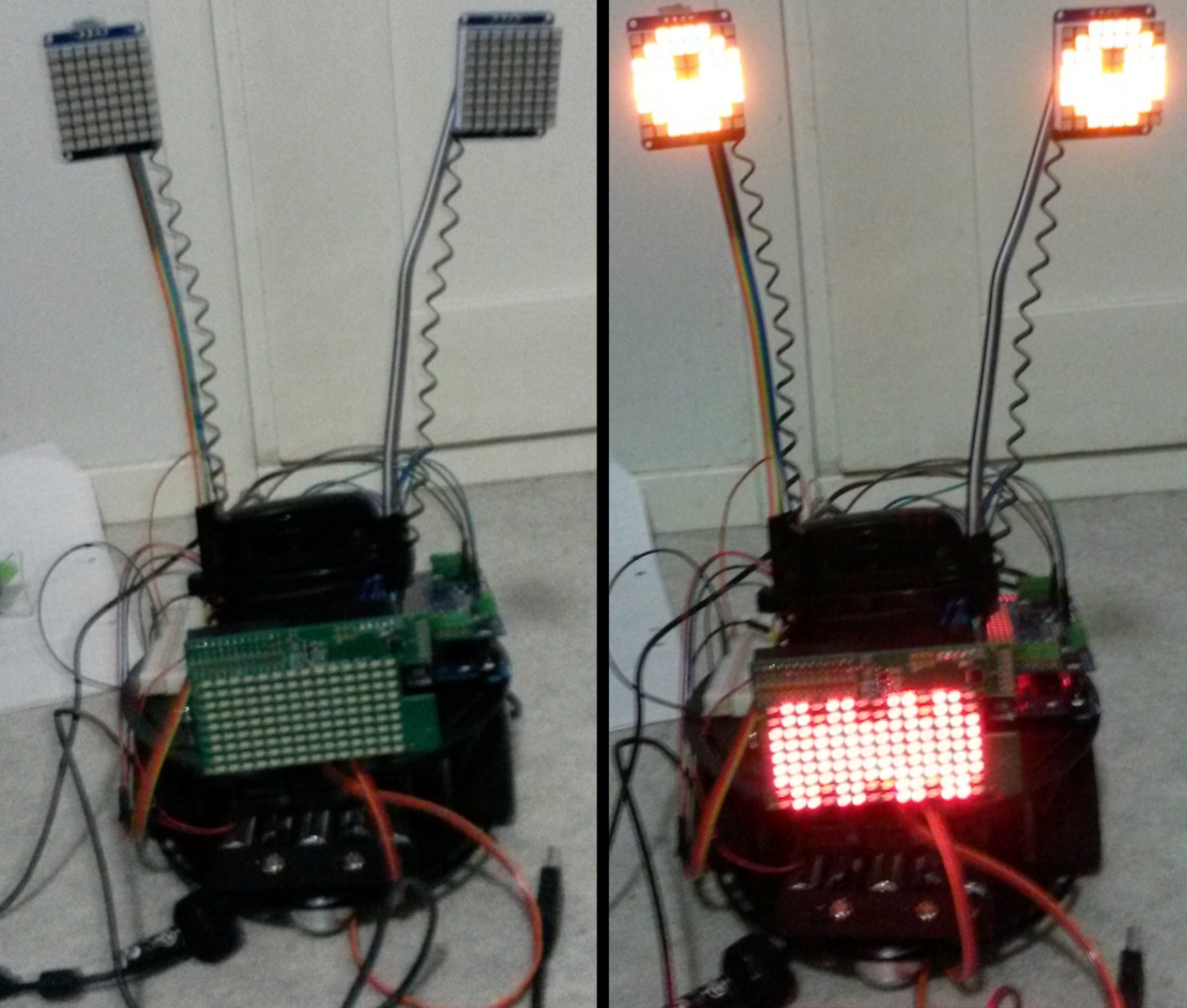


Figure 1. Robot prototype, powered off and on

The earlier robot, build for the aforementioned mobile programming course, was mostly constructed out of excess components and materials left over from various other electronics projects, as figure 1 demonstrates. Exceptions to this are the two-wheel robot chassis itself, which lays the basis for all the other various parts, and the Adafruit Motor Shield, which is an add-on for the Arduino microcontroller used to better integrate various motors and servos into the robot. Those parts were purchased specifically for this project.

The components the old robot consists of are as follow:

* Chassis with two DC motors and mounting holes
* Three LED matrixes, animated to function as eyes and the mouth
* Arduino Uno microcontroller with Motor Shield add-on
* 5 volt external battery bank to function as a power source for the microcontroller, LEDs and other embedded components
* Separate AA battery holder to power the robot movement
* HC-05 Bluetooth Classic serial module to function as an intermediate between the microcontroller and the mobile application
* A small piezoelectric speaker to provide audio effects
* A Raspberry Pi microcomputer powering a webcam

Also, here is a short overview of the different features of the previous robot prototype

* Robot can move around on wheels, with pan and tilt functionality for camera
* Robot can be controlled by an Android device
* Robot can stream video to phone with the help of the Raspberry Pi
* User can send messages the robot can scroll by on its “mouth”
* Arduino’s unused pins are able to be controlled by the application.

As for the companion application used to control the robot itself, it was somewhat archaic in nature. It used the default black-on-white styles of the Android environment, and contained many features that most people would find difficult to understand. Examples of such features are the ability to adjust the voltage of the Arduino’s pins, as you can see in figure 2. By itself, this feature doesn’t achieve much, and it requires other elements such as LEDs to properly demonstrate their functionality.

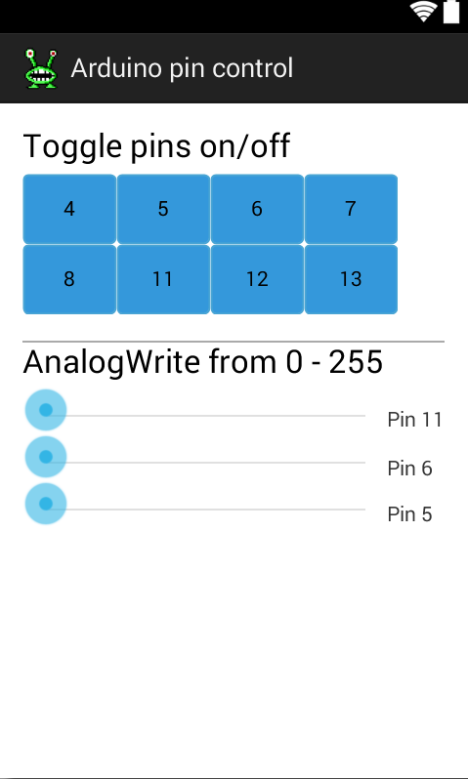


Figure 2. The pin control view

The first robot was more of a proof-of-concept, and its development did not have any particular end goals or results in mind. New features were added at the whim of the builder. This had the unfortunate side-effect of the actual physical wiring of the robot becoming quite messy and unattractive in the end, as you can see in figure 3. More clean and organized schematics for the wiring were decided to be an important point to keep in mind when starting the new build.

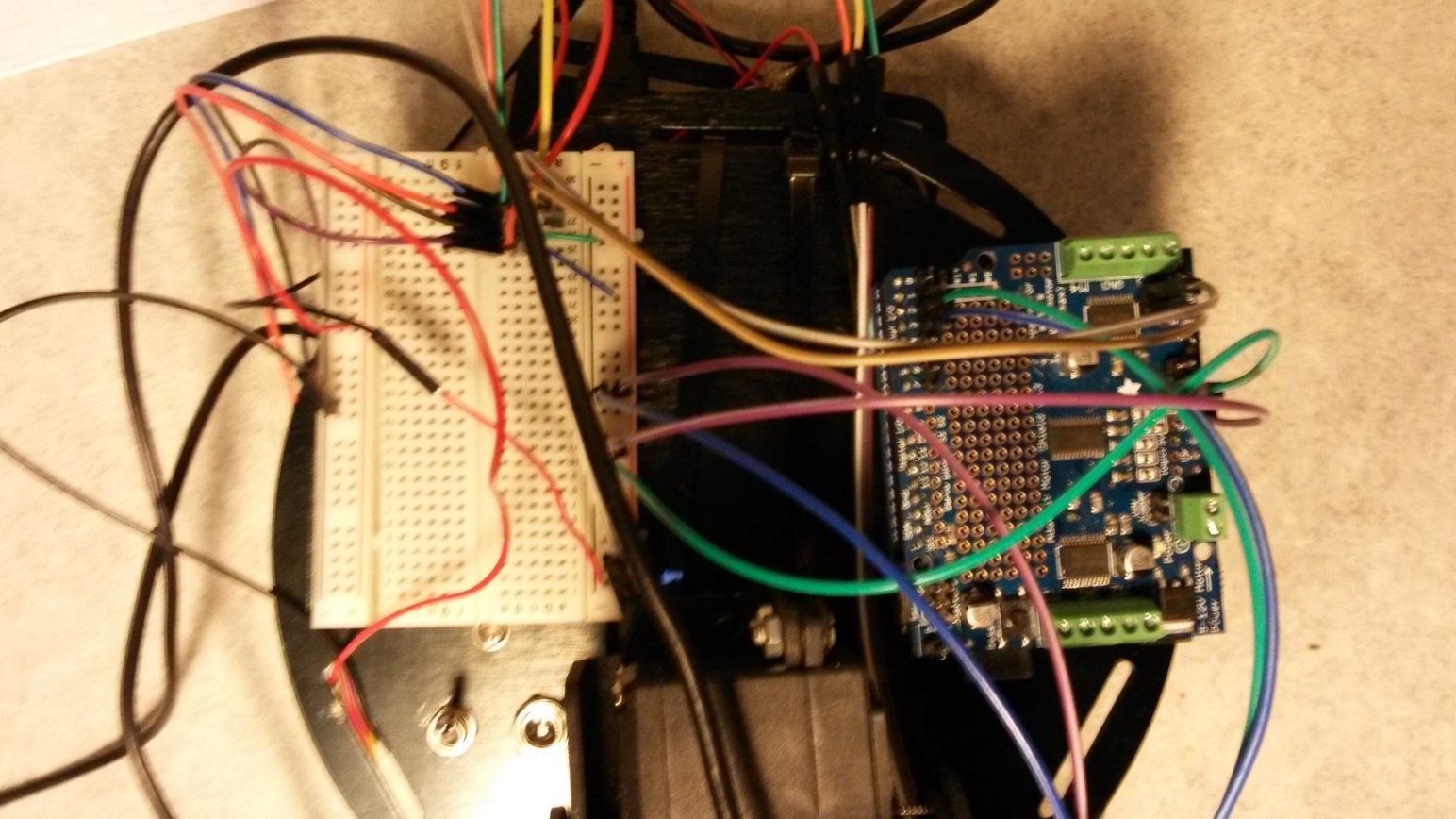


Figure 3. Top-down view of old robot wiring

## Objectives for the new project

During the initial thesis-related meetings, relatively large creative freedom was given with regards to the end product. However, it was decided that it would be very nice to have the mobile application also function properly on tablet devices. Furthermore, the two main requirements for the robot were agreed as ease-of-use and visual attractiveness. How these requirements should be fulfilled was not discussed at length.

It was decided that, as a technical experiment, the newly built robot would be built using Bluetooth Low Energy as the method of choice to transmit data between the mobile application and the microcontroller. The previous robot utilized the older Bluetooth Classic standard.

The main reason for this change was the promise of lower latency between the two systems, which would theoretically increase the responsiveness of the robot’s controls. The practical maximum speed packets could be transmitted with the previous prototype was once in around 115 milliseconds. Any faster output and the Arduino microcontroller could not keep up, and the packets would keep piling on in the serial buffer, which would eventually cause a fatal crash.

## Tools and technologies

### Arduino IDE

The Arduino IDE is a multi-platform development platform for creating embedded programs that run on the many various Arduino microcontrollers. As you can see from Figure 4, it is rather basic in its functionality when compared to many other modern IDEs. However, for small-scale hobbyist projects its relative lack of modern features, such as code auto-completion, is not an issue.

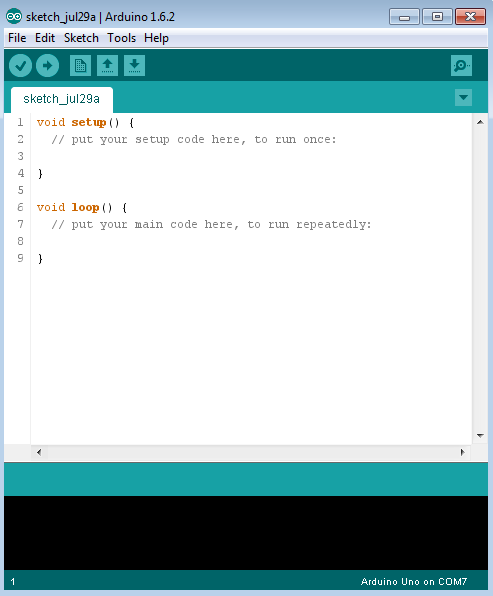


Figure 4. Screenshot of the Arduino IDE

### Eclipse

### C**++ and the Arduino language**

C++ is an object-oriented superset of the C language, originally developed by Bjarne Stroustrup from 1979. (Learncpp.com, 2007). Since its inception, it has influenced many other programming languages, such as C# and Java. It’s mostly designed for system programming and embedded, performance-intensive systems. (B. Stroustrup, 2014.)

While C++ is said to be a superset of the C language, The language used to program the Arduino family of microcontroller can be thought of as a subset of C++. While many of the familiar paradigms of C++ programming, such as object-oriented programming, are mostly too memory-intensive for embedded usage, is the Arduino language basically just a set of C/C++ functions.

The following is an example of a bare-minimum Arduino program:

The setup- and loop functions are the only two required functions that must always exist in a valid Arduino program. The setup function is called once, at the beginning of the program. The program then proceeds to run forever inside the loop function.

### Java

Much like C++, Java is also an object-oriented general purpose programming language. .(J. Gosling 2015) It was created by James Gosling in 1995, who was working for Sun Microsystems at the time. Sun Microsystems was acquired by Oracle Corporation in 2010, who have continued its development and maintenance ever since.

Java has quite a many uses, and one of the more popular uses is in programming mobile applications for the Android platform. In this particular project, the logic of the robot itself is programmed in C++ as mentioned before, and the mobile application used to control it is programmed in Java.

# The Android application

## Introduction

After much planning and testing, the final mobile application used to control the robot itself became quite simple in its functionality. A lot of functionality was dropped from the previous prototype (namely, the ability to control the voltage of the Arduino microcontroller’s different pins), because it was decided that they did not really fit the final purpose of the new robot prototype, being easy to use and looking pleasant. Also, many of the different configuration settings were dropped or heavily simplified, as they, too, did not bring any added value to the end-user.

The application itself consists of just two main activities. For all intents and purposes, one can think of activities in the context of Android programming to be the various different “screens” that make up a mobile application. When the user launches the application, they’re first greeted with a settings screen. After the user has paired their phone with the robot’s Bluetooth LE module, they can move on to the driving activity. Here they can control the robot with a game-like joystick interface.

## Activities

### Settings activity

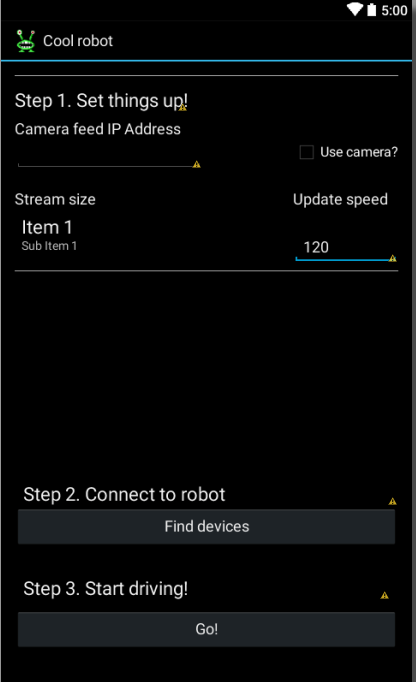


Figure 5. Android settings activity

This is the view where the application starts in, so in essence you might call it the “main” activity of the whole application. In it, you can for example configure camera settings. Since the feed is sent via Wi-Fi, the IP address of the Raspberry Pi might not be the same in all cases. Therefore it makes the most sense to let the user configure the webcam IP address to whatever they wish. There’s also the option to disable the video entirely, since during testing it was found out that it might unnecessarily slow down some older phones.

You can also change the update speed in milliseconds for the video feed view. This controls how often data is sent to the robot. Changing this to a lower number makes the robot more responsive, but setting it too low may cause the robot to act erratically if the Bluetooth module crashes due to too much data in its buffer. In the worst case scenario, this can lead to the robot driving out of control, and in need of being physically being restarted.

From the settings you can also pair with new Bluetooth devices. The dark gray button at the middle titled “Find devices” opens a pop-up window and initiates the discovery process for new Bluetooth devices and appends them to a list. From this list you can select devices and attempt to pair and form a connection, after which you can move on to the second activity.

After a successful connection to the robot has been formed, the user can move on to the next activity.

### Robot feed activity

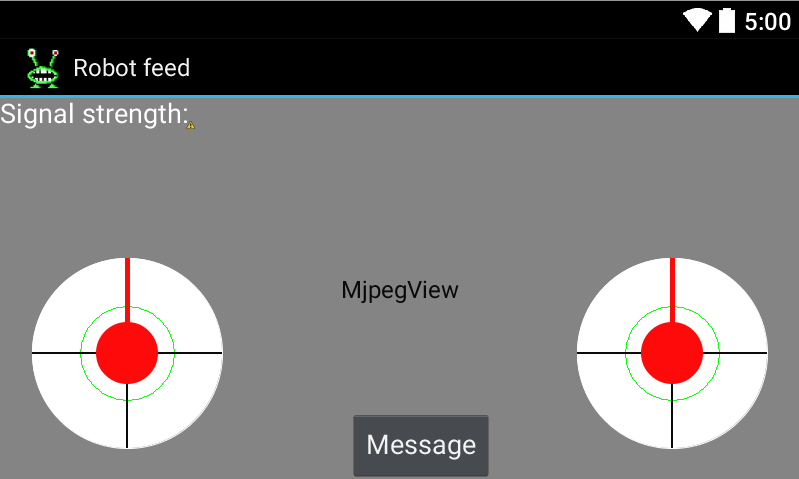


Figure 6. Robot feed activity

This is the part of the application where most of the action happens. The majority of the activity is taken by the video feed, with some transparent controls located at the bottom. There are two “joysticks” on the bottom of the screen. The left joystick is used to control the movement of the robot. So for example, if the user pushes the left joystick to the left, the robot will start turning to the left. The right joystick is used to pan and tilt the camera around. The further away from the center of the joystick you pull the faster the robot will move and react.

The button on the bottom labeled “Message” opens up a simple dialog with a text box. In it, you can type in text for the robot to display. This will cause the robot to actually speak the message out loud, using an on-board speech synthesis chip.

There is also a simple indicator on the top left for the strength of the Bluetooth connection. Signal strength is received from the module as an RSSI (received signal strength indicator), which is then then calculated into a more intuitive percentage value.

## Application classes

### MjpegInputStream & MjpegView

The camera feed uses the popular MJPEG (Motion JPEG) format to read and handle data from the webcam server running on the Raspberry Pi. The Pi is hosting a GStreamer-pipeline, which is a popular open-source framework for handling multimedia of different kinds. MjpegInputStream is the bytestream that fetches data from the pipeline, and MjpegView is the UI widget which displays it. These two classes were gotten from a GStreamer-specific fork of the original MJpegViewer-project for Android. See here: <https://bitbucket.org/coisme/simplemjpegview_gst/wiki/Home>

### JoyStickView

This is the joystick component, two of which are used on the video feed. They have an event-listener that updates on the joystick moving, and you can get their angle, power and direction out of it. This particular component is originally developed by GitHub-user Zerokol, with some modifications made here and there to ensure the best fit for the project at hand (<https://github.com/zerokol/JoystickView>)

### ApplicationState

The primary functionality of the ApplicationState class is to pass a reference to the BluetoothStreamManager class around between different Activities. Generally speaking, in object-oriented programming languages such as Java, having a single large class such as this one is considered a bad practice. However, for such a small application such as this one, it was decided that this is an acceptable solution, since a complex object could not be passed between activities easily.

### BluetoothStreamManager

The main functionality of this class is ensuring data is properly transmitted to the robot. It offers a simple interface for common Bluetooth related actions such as connecting to a device, getting the strength and state of the Bluetooth connection, and transmitting data. The other activities of the application can then use this class to send commands to the Arduino, which are then captured in the device’s output.

The BluetoothStreamManager also holds a reference to the current activity, so it can display an error dialog if it loses connection to the Bluetooth module, which prompts the user to return to Settings and attempt to repair devices.

In the previous iteration of the Android application, this class was somewhat more complex. Activities using the class could not write data to the Arduino directly, but instead they would push data to a queue. A separate thread would then write the data from this queue to the output stream. This entire queue system was decided to be quite over-engineered for this particular use-case, since there was not really all that much data being transmitted between the different activities.

### Feed

This is the class corresponding to the Feed activity. Its main function is to send data to the Arduino every x milliseconds, where x is the update speed value specified on the settings screen. Data is transmitted as an array of bytes, with is then interpreted on the embedded side. Most of the time, this byte array contains values that tell the robot what it should do in terms of movement. The two joysticks on the screen are used to get user input, which these motor valued are then calculated from. This communication protocol will be further extrapolated on in the upcoming sections.

The activity also contains a button that opens up a dialog where the user can send things for the robot to say. It also contains an implementation of the AsyncTask class, which is used to fetch the video feed asynchronously from the robot, provided the user has enabled that feature from the settings activity.

### Settings

This class, corresponding to the Settings activity, just keeps tabs of some input fields used for configuration for the rest of the app. These settings are saved and loaded from Android’s SharedPreferences, which is a simple way to store values in between application sessions.

The other main functionality of this particular class worth mentioning is the pairing of Bluetooth devices, which is handled in this class. The user can press a button to initiate scanning for new, unknown devices. If any are found they’re then appended to the end of the list. Clicking on an item in the list initiates pairing with a Bluetooth device. The Bluetooth device is then paired with the phone and the output stream is passed on to BluetoothStreamManager, and the user can proceed to other activities.

# The robot

## Introduction

As you can see from figure 7, the new build of the robot looks quite similar to the old one. At a glance, the biggest differences are the new, larger eyes and mouth. Building the actual robot was a bit easier than in the previous project, as there was a much clearer general idea of parts needed, and possible pitfalls to overcome.

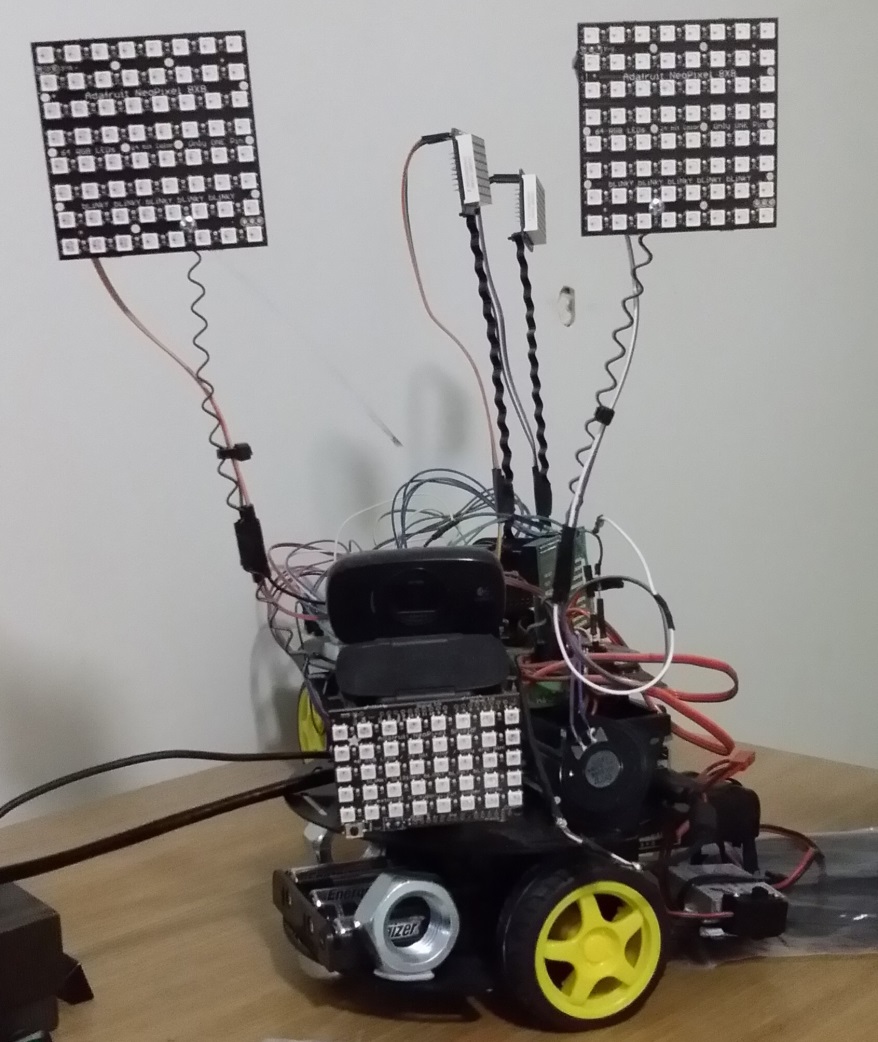


Figure 7. Finished robot

## Overview of parts

Table 1. Robot parts & prices

|  |  |
| --- | --- |
| Part | Cost (€) |
| Pan + tilt servo | 30 |
| 4000 mAhLiPo 7.2V battery + 5V/9A UBEC | 40,2 |
| 2 NeoPixel NeoMatrix 8x8 | 59 |
| NeoPixel Shield | 26,61 |
| Adafruit Motor Shield V2 | 17,56 |
| Emic 2 Text-to-Speech module | 59,95 |
| Small speaker | 1,95 |
| Robot chassis with motors | **56** |
| Bluetooth LE module (NRF8001) | 34,85 |
| Assorted wiring, resistors, etc. | 15 |
| Raspberry Pi 2 (Loaned from school) | N/A |
| Webcam (Loaned from school) | N/A |
| WiFi adapter | 20 |
| Arduino Mega | 45 |
|  |  |
| Total: 406,12 € |  |

Table 1 contains a short listing of all the different parts purchased during the project. Some of the parts were loaned from school for the purpose of this project, so their price is excluded from the listing.

### Arduino Mega 2560

Functioning as the “brains” of the robot is an Arduino Mega 2560 microcontroller. The Arduino handles essentially all the functionality, from animating the eyes and mouth and moving the wheels and servos to reading data from the Bluetooth module.

In the original project, an Arduino Uno was used. The Uno is a slightly cheaper hobbyist microcontroller, with a bit less memory, pins and other features. The small amount of memory in the Uno was the deciding factor for the change, as it turns out the libraries needed to interface with the Bluetooth module and the NeoPixels needed a lot more memory than was anticipated. The Uno and the Mega both run at 16MHz which means both are equally fast in computing power.

### Robot chassis

This is the “body” of the robot, so to speak. The kit itself contains two cheap DC motors and a caster ball for movement, two different platforms with a variety of mounting holes and rails, a standard battery pack that can hold up to 4 AA batteries and a switch that is meant to be used to turn the aforementioned battery pack off and on.

This is the same hobby kit that was used in the previous prototype, as it had proved to be an affordable and good solution. However, the cheap motors ended up causing some issues, as when the robot reached a certain speed, they started to generate a critical amount of electrical noise which caused the Arduino to operate in an unstable manner. This was fixed quite easily by soldering some noise suppression capacitors to the motors and the metal chassis.

### Pan + tilt servo

Originally intended for mounting a security camera, this component consists of two servos laid on top of each other in a manner that allows a movement range of almost 360 degrees. Naturally, this range is limited on the side of the embedded code, as otherwise the eye stalks could potentially get stuck on the wiring on the back of the robot.

On the top servo sits the webcam that is connected to the Raspberry. The two-servo system gives the camera a lot of flexibility, and allows it to capture video at a variety of different angles. The eyes and mouth are also mounted to the top servo.

### LiPo battery and UBEC

Nearly the whole robot is powered by a single 7.2 volt lithium polymer battery, originally meant for RC cars. The exceptions to this are the motors and the servos, which are powered from 5 AA batteries. The reason for this is to avoid instability-causing voltage drops under heavy loads. Therefore, it’s better to keep the power sources entirely separate. Since all the components, from the Raspberry Pi and Arduino are rated for 5 volts, the LiPo battery is connected to a Universal Battery Elimination Circuit. The purpose of this component is to take in a high amount of voltage, and then convert it to a steady 5v output. This output can be then safely used without fear of burning out sensitive circuitry with too much voltage.

The components of the original robot were powered from a standard 5 volt USB power bank. This power bank was originally meant to be used to recharge mobile devices, but it served as a portable, easy-to-use power source quite well. The reason it didn’t make it into the new robot was because its maximum output was only 3 amperes, and the new robot uses quite a lot more than that at its peak. The biggest reasons for this increased power consumption are the new RGB LEDs of the mouth and eyes. Each NeoPixel uses about 20 milliamps of current, and there are a total of 173 LEDs in this entire project. By itself, this totals up to a potential draw of 3.6 amperes.

While one is planning out a project that incorporates microcontrollers interacting with servos or motors or other “heavy-duty” components, one should always remember to separate their power supplies. Therefore, the two DC motors and the two servos are powered separately, with a regular battery pack that takes up to five AA batteries.

### Adafruit Motor Shield V2

In the world of Arduino, there exist many different kinds of add-ons called Shield. They can be stacked on top of an Arduino microcontroller to further enhance their functionality in many different ways. One such shield is the Adafruit Motor Shield, which is used in this project to control the various motors and servos.

The main advantage of using a ready-made add-on such as the motor shield in this a project like this is ease-of-use. Most of the “heavy lifting” can be safely left to the motor shield, and the builder doesn’t have to worry about issues such as protecting their microcontroller from unwanted power surges. https://learn.adafruit.com/adafruit-motor-shield-v2-for-arduino/overview

### NeoPixel shield & matrix

Purely for visual show, the robot contains a total of 173 full color LEDs. Two 8x8 form the robot’s eyes, and a 9x5 rectangular piece functions as its mouth. Mounted on top of two “stalks” cut from a plate of metal, the eyes blink and move around in a somewhat realistic fashion. The mouth also moves when the robot is talking.

### Emic 2 Text-to-speech module & speaker

The Emic 2 is a voice synthesizer that takes in a stream of characters, and then attempts to convert it to speech. This makes it quite easy to integrate into embedded projects, you just send serial data for it to pronounce. Naturally, the module also needs a small speaker attached to function.

### Raspberry Pi 2 with webcam

The Raspberry Pi 2 is the newest model in the series of single-board computers made by the Raspberry Pi Foundation. For all intents and purposes, one can think of it as a small computer that has the ability to run distributions of the GNU/Linux operating system. The sole job of the Raspberry in this project is to host a GStreamer-based server that constantly outputs video from the on-board webcam. To aid in this task, the Raspberry also hosts its own Wi-Fi access point. If the user of the mobile application wants to access the video feed, they must first join this network the Pi is hosting.

In the original project, a Raspberry Pi 1 model B+ was used. The reason for this was simply that the new version wasn’t available to the public yet at the time. Also in the original project the phone itself hosted the Wi-Fi access point, which the Raspberry would join. The reason for essentially switching the hosting around is that it allows multiple devices to access the feed without restarting. It was also easy to forget to turn on your phone’s access point before starting the robot

## Assembly

# The embedded code

## Introduction

In essence, the entire embedded program is essentially one big state machine. The program handles animating the eyes, reads bytes from the Bluetooth module and reacts accordingly.

Like was mentioned in chapter 1.4.3, all Arduino programs consist of a setup function which is run once, and a main loop which is run indefinitely after the setup function. In this particular program, the setup function mostly handles initializing the various components which together make up the robot. For example, it starts the serial connection to the text-to-speech chip and Bluetooth module, sets up the correct brightness and state for the various LEDs, and sets up the servos to their correct initial positions to make the robot face forwards.

The main loop checks for and reads bytes in the serial buffer of the Bluetooth module. It also handles the animations of the eyes and mouth, the latter only if the robot is currently speaking via the text-to-speech chip.

All in all, the robot has three possible states. These states are MOTOR, MESSAGE and NOTHING. With the exception of the “NOTHING”-state, these states directly correspond to the different kinds of messages the Android application can send. “NOTHING” just means that the program is currently idle, and is waiting for further input from the Android application.

If the robot is in “MOTOR”-state, it means that the next bytes it will receive will indicate the various speeds and states it should move its wheels and servos. Likewise, if the robot is in “MESSAGE”-state, it means that the all the incoming bytes should be interpreted as letters, which are then sent to the text-to-speech chip after a final delimiter has been received.

## Communication between Android application and robot

Every message send to the microcontroller is sent and read as arrays of bytes, as was outlined in the earlier chapter. For all intends and purposes, in the context of this application a byte can be thought of as a numeric value between 0 and 255. However, since bytes in the Java language are designed to be from -128 to 127, and the bytes in C++ are from 0 to 255, it was found out that they cannot be directly transmitted. To overcome this issue, the values sent from the Android application are from 0 to 127. They are multiplied by 2 on the Arduino side to get the correct values.

### Motor command

Here’s an example of sending a simple movement command. The brackets represent the cells in the array:

[[123], [70], [64], [66], [127], [1]]

The first character is a simple delimiter, which lets the microcontroller know that the following five characters should be interpreted as a movement command, and not anything else. Unlike for the other commands, there is no need for an ending delimiter, as the microcontroller itself knows that the robot command is only six bytes long at best.

The next byte tells the direction the leftmost motor should move in. This byte corresponds to either the ASCII character F for forward, B for backwards for or R (82) release, which is for the wheel in question staying put. The third byte is the speed the leftmost motor should exert, from 0 to 127. In this example, the leftmost motor is told to spin forward at half of its maximum speed.

The next two characters are the same as the last two, except they’re for the right motor. In this case, the two numbers tell the motor to start moving backward at full speed.

The last character is related to positioning the two servos. It is a numbers from 0 to 9, which tells the two servos, which direction they should move to. This movement is relative to their current position. One can think of these as the eight cardinal directions, plus an extra value that means both servos should stay still.

Here’s a method of displaying text on the robot’s “mouth”. This time, let’s use ASCII values.

z …. \n

This one is pretty simple. The z-symbol (or the byte corresponding to it, whichever way you look at it) marks the start of a new displayable message, followed by an undefined amount of characters to scroll on the matrix, and finally the newline character ends it.

Finally, I have to implement a way to adjust the state of the Arduino’s pins. Handling digital pins is very simple, since their state is either on or off. We can just send a single byte corresponding to the pin we want to toggle.

[[120], [5], [60]]

The first byte implicates that the command should be interpreted as AnalogWriting a pin. The following value tells the microcontroller which pin it should set the state of. The next character tells the microcontroller the value it should write. In this case, it’s 60, which means full wave length. The command is then terminated with a closing character.

You can also toggle the pins on and off, from 0 to 5 volts. It works in sort of the same way as the AnalogWrite.

[[121], [5], [0]]

The first byte is, as always, the delimiter. The second byte is the number of the pin in question, and the third value is either 0 or 1, for off or on, respectively. Like the motor commands, AnalogWrite and Toggle neither require an ending delimiter.

# Results and conclusion

When it comes to creative work, it can sometimes be difficult to predict the final quality. The initial requirements, both functional and non-functional, were somewhat loose for this project. However, all things considered, the end result of this thesis far exceeded the expectations of the initial assignment.

Developing the application and embedded code itself was surprisingly easy. A lot of this had to do with the fact that much of the work was adopted from the previous project. The biggest difficulties where with utilizing Bluetooth LE on the Android side, as its support is relative new in the Android ecosystem, being supported only from Android 4.3 and upwards. This resulted in there not existing a lot of documentation. However, this was not a huge issue in the end.

Using Bluetooth LE for the communication between the application and the robot was an interesting experience. However, Bluetooth LE is mostly meant for devices that need to transmit data to the host device relatively rarely, which is not the case for this particular project, as it needs to transmit data every 200 milliseconds or so. Also, as the name “Low Energy” implies, Bluetooth LE is intended for embedded devices with fairly restrictive power requirements such as heart trackers or other wearable technologies. The amount of power is not really an issue in this case, since the difference between Bluetooth Classic and Bluetooth is still measured in dozens of milliamps, which do not really make a noticeable difference. There’s also the issue of cost and ease of implementation, as Bluetooth Classic modules can be acquired quite cheaply, and there exists a wealth of documentation for their usage.

# References

Understanding RC LiPo Batteries. Accessed on 27 July 2015. Retrieved from <http://www.rchelicopterfun.com/rc-lipo-batteries.html>

Stroustrup, B. 2014. Lecture: The essence of C++. Accessed on 29 July 2015. Retrieved from https://www.youtube.com/watch?v=86xWVb4XIyE

Learncpp.com, 2007. Introduction to C/C++. Accessed on 29 July 2015. Retrieved from http://www.learncpp.com/cpp-tutorial/03-introduction-to-cc/

Gosling, J.; Joy, B; Steele, G.; Bracha, G.; Buckley, A., 2015. The Java Language Specification. Accessed on 29 July 2015. Retrieved from https://docs.oracle.com/javase/specs/jls/se8/jls8.pdf