jamk.fi

Arduino-controlled Robot

Matti Jokitulppo

Bachelor's thesis
August 2015

Degree Programme in Software Engineering
School of Technology, Communication and Transport

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

Description

Author(s)	Type of publication	Date
Jokitulppo, Matti	Bachelor's thesis	21.08.2015
		Language of publication: English
	Number of pages: 45	Permission for web publication: x

Title of publication

Arduino-controlled robot

Degree programme

Software Engineering

Tutor(s)

Manninen, Pasi
Mieskolainen, Matti
Kotkansalo, Jouko

Assigned by

JAMK University of Applied Sciences, Kotkansalo, Jouko

Abstract

The aim of the thesis was to create a 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.

The thesis contains a detailed 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 need arose.

As a result of the thesis, the customer received a fully functional robot with companion app, which could easily be demonstrated in fairs or other similar events. It could be controlled using any Android compatible phone or tablet.

Keywords/tags (subjects)

Arduino, Bluetooth, Bluetooth LE, Android, Robot

Miscellaneous

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Jokitulppo, Matti	Opinnäytetyö	21.08.2015
	Sivumäärä 45	Julkaisun kieli Englanti
		Verkkojulkaisulupa myönnetty: x

Työn nimi
Arduino-controlled robot

Koulutusohjelma
Ohjelmistotekniikan koulutusohjelma

Työn ohjaaja(t)
Manninen, Pasi
Mieskolainen, Matti
Kotkansalo, Jouko

Toimeksiantaja(t)
Jyväskylän ammattikorkeakoulu, Kotkansalo, Jouko

Tiivistelmä

Opinnäytetyön tavoitteena oli luoda Bluetooth-kommunikaatiostandardin ylitse kontrolloitava robotti Jyväskylän Ammattikorkeakoululle markkinointitarkoituksiin. Robottia voidaan käskyttää Bluetooth LE-linkin ylitse millä tahansa Android-yhteensopivalla laitteella mobiilisovellusta hyödyntäen. Robottia itseään ohjaa Arduino-mikropiiri, jota sovellus käskyttää omalla protokollallaan muodostettuaan Bluetooth-linkin.

Opinnäytetyö sisältää kattavan kuvauksen robottiprojektin rakennusprosessista, mukaan lukien 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 molemmille alustoille tarpeen vaatiessa.

Työn tuloksena tilaaja sai toimivan robotin ja mobiilisovelluksen, jonka avulla robottia voidaan helposti käskyttää ja esitellä messuilla tai muissa vastaavissa tilaisuuksissa. Sitä pystyy ohjaamaan millä tahansa Android-yhteensopivalla puhelimella tai tabletilla.

Avainsanat (asiasanat)

Arduino, Bluetooth, Bluetooth LE, Android, Robotti

Muut tiedot

Contents

Figures

Tables

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

An integrated development environment is a type of text-editor usually meant for source code. Besides editing text, IDEs often come bundled with a host of other features, such as a debugger, or a built-in compiler.

LiPo Battery

Lithium polymer batteries are a type of batteries very popular in the world on remote controlled planes. They are lightweight, hold a great deal 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 its 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. (Speedguide.net)

Servo

A servo is a special type of motor that is 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 usually have many different running processes, which take up different amounts of resources at times, it is 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.

SPI

Serial Peripheral Interface is yet another serial data protocol popular with embedded systems. It consists of one master device (usually a microcontroller, such as an Arduino) that commands one or more other devices. (Arduino.cc, 2015)

UART

UART is a two-way serial communication protocol, which means there is 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 is no clock line, the data transfer amount per second must be negotiated in advance, before starting communications.

1 INTRODUCTION

1.1 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 it was suggested that a new version could 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 prototype. It would consist mostly of an Arduino microcontroller controlling a basic two-wheel hobbyist robot chassis. A small single-board computer called the Raspberry Pi would provide a live video feed via an on-board camera mounted on top of two servos, which can pan and tilt in almost every direction. The robot could be driven using an Android phone, tablet or other compatible device, over a Bluetooth connection. It could also transcribe speech utilizing a natural speech synthesis module.

After initial thesis-related meetings, it was decided that 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. To summarize, the goal of this thesis project was to build a functional, engaging remote-controlled robot for marketing purposes.

1.2 The previous prototype

1.2.1 Overview

The earlier robot, seen in Figure 1, was mostly constructed out of excess components and materials left over from various other electronics projects. 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.

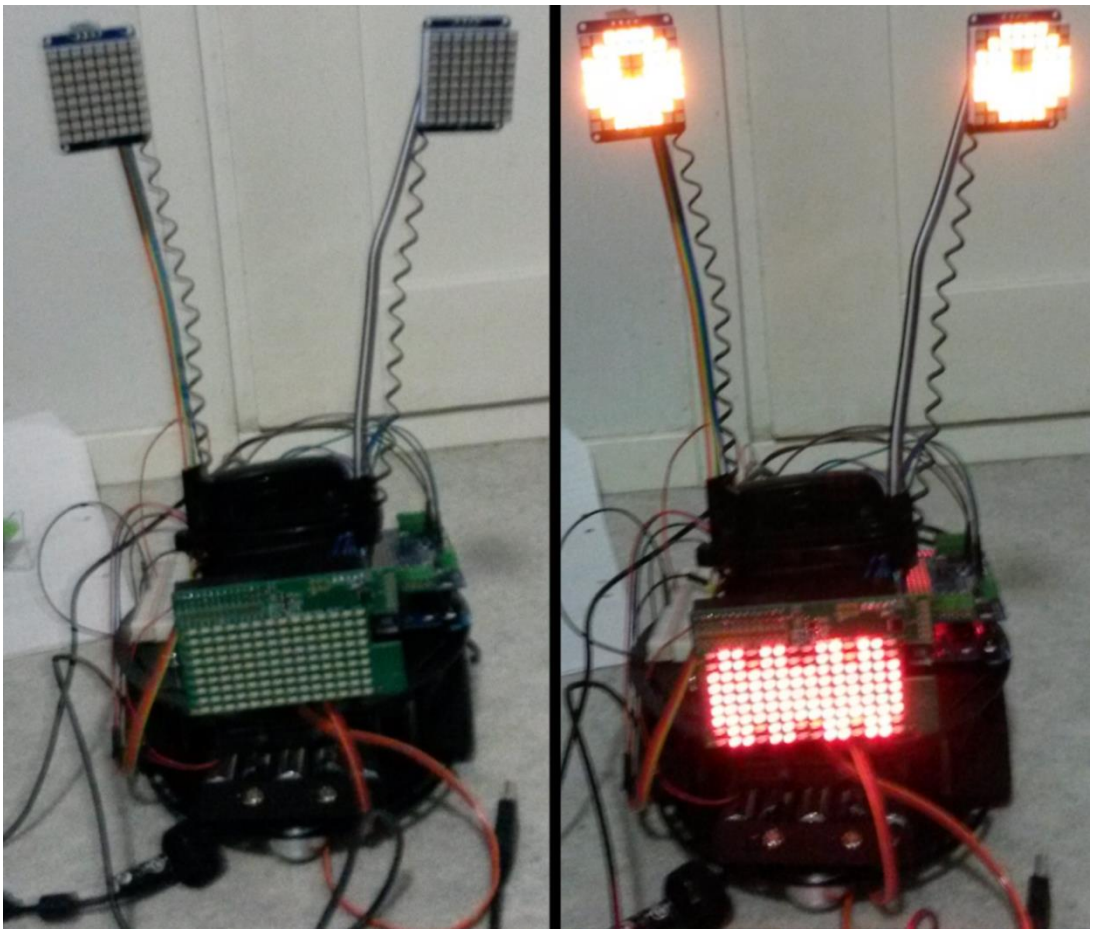


Figure 1. Previous robot prototype

1.2.2 Features

When the old robot project was first starting out, the immediate end-goal was just to have a small vehicle of some sort that could be controlled through an Android application via Bluetooth. During the building process itself, many new ideas and functionalities were thought of and implemented, essentially on-the-fly. Soon after putting together the chassis and wiring up the wheels, it was noticed that the robot looked somewhat dull, and that it needed something extra to really make it memorable. In a previous project, the builder had used small LED matrixes to add animated eyes to a Halloween mask, and so it was decided that the robot could also use a pair of eyes for enhanced visual effect. Soon after the LED eyes came the idea of using another LED matrix for a mouth. This way, the robot could have a face of sorts.

After the robot got its face, it was decided that it would also need a voice. Since the LED matrix used for the mouth came with the functionality to scroll text on it, a way of sending messages from the Android application to this LED matrix through the Arduino was implemented. Afterwards, a small speaker was attached to the robot to provide audio effects while the text was being displayed.

As a somewhat late addition to the project a webcam streaming video to the application was added. This was simply because during testing it was found out that driving the robot was quite difficult if you could not see where you were going. A Raspberry Pi single-board computer was added to provide the functionality of recording video and sending it through Wi-Fi.

1.2.3 Components

The old robot and the new version share quite a lot of parts used in common. For example, both of them are built on top of the same two-wheeled robot chassis, which is then driven through the Adafruit Motor Shield. The two also have a similar approach to controlling the robot through the Android application, although the old model uses Bluetooth Classic, and the new version utilizes the somewhat newer Bluetooth LE standard of communication. Both of the robots also contain a

Raspberry Pi streaming video footage through a webcam that is planted on top of the same make of servos. In fact, the webcam used in the new robot was the very same that was used in the old version, as it was originally borrowed from school and was seen to be a functional and proven solution.

The new robot utilizes the somewhat new NeoPixel brand of LEDs for its eyes and mouth. They sport features such as 255 different stages of brightness and a total RGB color space of 24 bits. Each NeoPixel “unit” come with their own separate LED driver, and they can be chained together to form complex light installations of varying shape and size. Meanwhile, the older robot uses two orange single-color 8x8 LED matrixes for the eyes, and a 14x9 red rectangular LED matrix called the Pi-Lite for the mouth, which can be seen in Figure 2. As the name implies, the Pi-Lite is an add-on originally meant to be used together with a Raspberry Pi. However, it was repurposed for the old robot from a previous electronics project. The Pi-Lite interprets commands sent through simple UART, which means its usage is not strictly tied to any particular brand or type of microcontroller or computer. While the single-color LEDs of the old robot might not look as impressive or bright as the NeoPixels, they do have the advantage of using significantly less power.

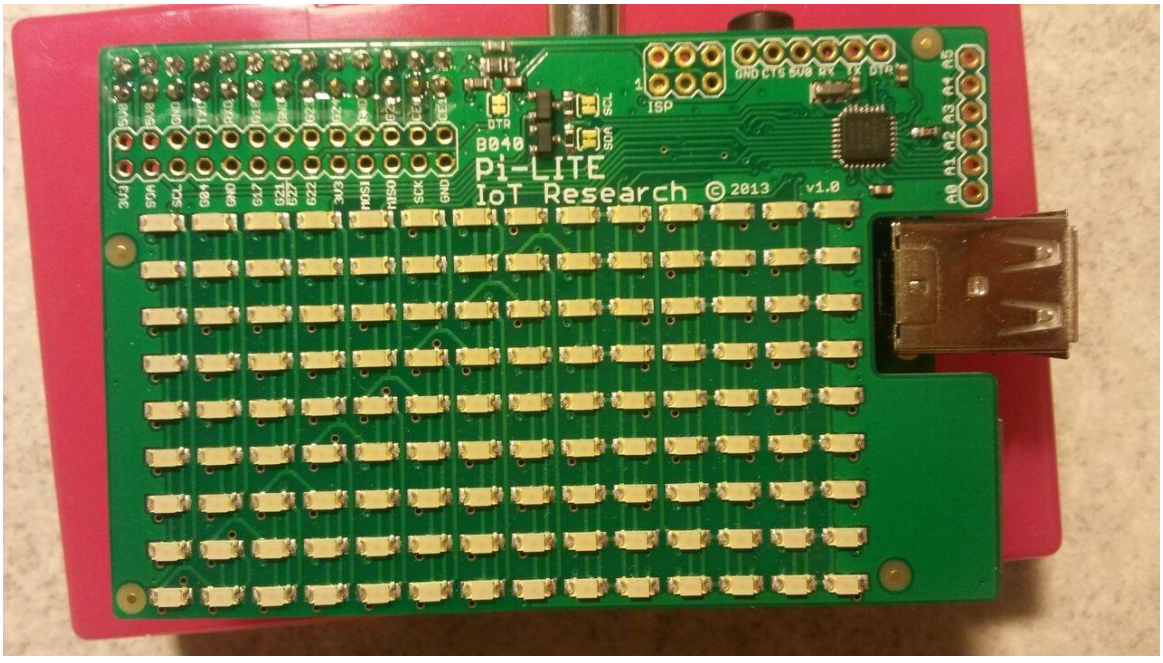


Figure 2. Pi-Lite LED shield on top of Raspberry Pi

For communication the old robot uses the HC-05 serial Bluetooth module. The HC-05 is remarkably easy to use in embedded projects, as it does not require any additional libraries to function, and it sends data through UART, which means it only needs two pins, RX and TX to communicate data back and forth. Usability aside, the HC-05 can also be acquired for very cheap online, at least when compared to newer Bluetooth LE modules.

While the new robot contains a very sophisticated and expensive speech synthesis chip, the old robot had a comparatively simple piezo speaker to provide sound effects. Piezo speakers are very lightweight and small components made from a special ceramic material, which causes the speaker to vibrate when an electronic signal passes through it. This causes the speaker to emit sound waves, the frequency of which is affected by the frequency of the signal that passes through the piezo. Piezo speakers are used in many embedded applications, such as toys, for example. In the old robot, the speaker played a continuous robotic beeping tune when it was scrolling messages on the Pi-Lite.

The old robot also included a Raspberry Pi, streaming video to the phone through a webcam. In the original project, one had to open up a Wi-Fi access point from your mobile device, which the Raspberry Pi would then attempt to join. If it joined successfully, the stream could be connected to through the Raspberry's IP address, if the phone is in the same network. This is not really an ideal solution, as it is not very intuitive for the user to have to turn on their device's Wi-Fi access point every time they want to use the robot.

All in all, 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 can be seen 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.

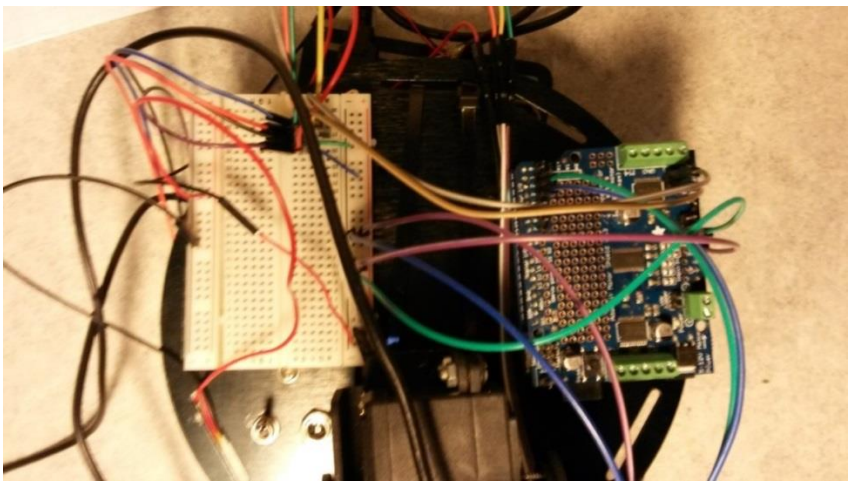


Figure 3. Top-down view of old robot wiring

1.2.4 Android application

As for the companion application used to control the robot itself, it was somewhat archaic in nature in its early versions. 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 illustrated in Figure 4. By itself, this feature does not achieve much, and it requires other elements such as LEDs to properly demonstrate its functionality. The simplicity of the end-user interface was due to the fact that the application would never be released to the public, and it only served a single purpose, to control the robot.

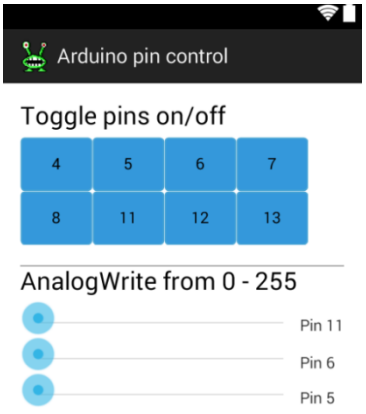


Figure 4. Old pin control activity

As was previously mentioned, the original robot was built as a final project for an Android programming course. Development of the Android application was started quite early during the course schedule, and the pitfalls and best practices in Android development were not yet fully known at the time. Thankfully, the Android side of the project was relatively small, and the lack of experience in mobile programming did not turn out to be too big of an issue.

1.3 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.

2 TOOLS AND TECHNOLOGIES

2.1 Arduino

2.1.1 Introduction

Arduino is the name of an Italy-based popular electronics platform that aims to be as easy-to-use as possible. Both the hardware and software are completely open-source. It is intended for easy and fast prototyping, and is mostly aimed towards aspiring students without a background in electronics or programming. (Arduino.cc, 2015).

Arduinos come in many different varieties. Most of them are based around the AVR-series of chips, which are manufactured by Atmel. While it is these chips that provide all the real functionality, the Arduino board simply makes using them that much easier. It can be argued that, for experienced electronics users, the true value of Arduino comes not from the boards, but the entire open-source ecosystem that has grown around them, and all the libraries and communities they provide.

Arduino was picked for this project because it is beginner-friendly, easy to develop for and has a wealth of examples and tutorials online. It is important to keep in mind that the ultimate goal of this robot project is to create something fun and approachable, that can be understood and digested by anyone.

2.1.2 Arduino shields

Arduino shields are pluggable add-ons that can be stacked on top of the Arduino board to further extend its capabilities (Arduino.cc, 2015). There exist hundreds of different kinds of shields on the market, both official and unofficial. They provide features such as wireless communication in the form of Wi-Fi or radio, easier communication with motors and servos, added sensors and much more. For this particular project, a shield called the Adafruit Motor Shield was used to control the two DC motors and two servos.

2.1.3 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 is 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, the Arduino language is basically just a set of C/C++ functions.

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

```
1 void setup() {
2   // put your setup code here, to run once:
3
4 }
5
6 void loop() {
7   // put your main code here, to run repeatedly:
8
9 }
```

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, when the microcontroller is first powered up, at the beginning of the program. The program then proceeds to run forever inside the function called loop.

2.1.4 Arduino IDE

The Arduino IDE is a multi-platform development platform for creating embedded programs that run on the many various Arduino microcontrollers. As seen in Figure 5, 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.

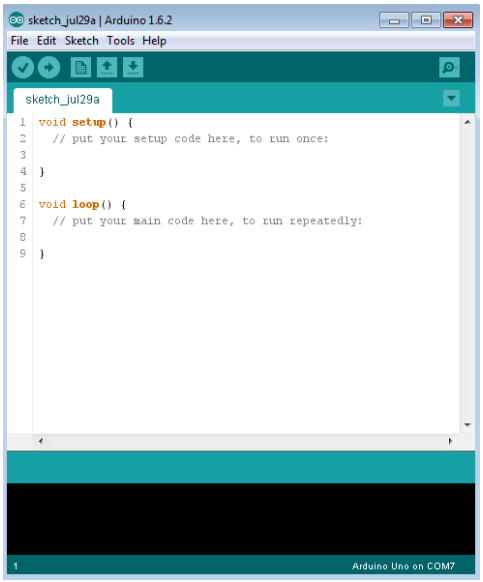


Figure 5. Arduino IDE

Deployment of the code to the microcontroller in the Arduino IDE is quite streamlined. One just needs to attach an USB cable to the to the USB port of the Arduino, pick the right board model from the IDE and upload the code. The entire Arduino IDE with its many utility libraries is actually just a user-friendly wrapper around the AVR-GCC compiler, which handles compiling the source code to a format that the Arduino's on-board ATmega can understand. After the code has been checked for errors and compiled into binary, another utility program by the name of AVRDUDE (short for AVR Downloader/UploaDEr) handles the uploading and saving of the program to the Arduino itself. Many advanced users choose to bypass relative unwieldiness of the Arduino IDE by using their preferred text editor/IDE, and just use the AVR-GCC and AVRDUDE as stand-alone programs.

2.2 Android

2.2.1 Introduction

Android is a very popular Linux-based operating system, specifically tailored for smartphones and tablets. It is estimated there are currently over one billion devices

running Android in the world (Android.com, 2015). Besides tablets and phones, Android is also found in TVs, wrist watches and cards, for instance.

Being based on the Linux kernel, Android is open-source technology. Android was originally developed by Android Inc., however the company was acquired by Google in 2005, who is the current maintainer of Android. Even though Android is technically an open source technology, most manufacturers include at least some closed-source proprietary drivers, depending on the model. This has the unwanted side-effect of many Android devices being difficult to update, as Google cannot simply release a new version of the operating system. Because of this, Android has been criticized for having a lot of fragmentation with regards to its many different versions. This makes it difficult for Android developers, as they may have to develop for many wildly differing versions of Android.

For this project the reason Android was picked as the development platform of choice was that there already existed precedent of Android being a well-suited platform from the previous project.

2.2.2 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.

2.2.3 Eclipse

Like the Arduino IDE, Eclipse is also a type of integrated development environment. While the Arduino IDE is mostly meant for beginners that are easing themselves into embedded programming, Eclipse attempts to target the enterprise audience. Eclipse is most known for its Java features, however, it also supports many other languages, such as C++ or PHP in the form of plugins. (The Eclipse Foundation, 2015).

In this particular thesis project, Eclipse was used as the main platform for developing the Android application. However, it bears mentioning that since starting this project, Eclipse has been phased out by Google as the Android IDE of choice, in favor of their own IDE Android Studio (J. Eason, 2015). A switch to Android Studio was not deemed necessary during this project, as the groundwork laid during the development of the previous robot was entirely made with Eclipse, and changing IDEs was deemed unnecessary.

2.3 Version control and management

2.3.1 Git

Git is a distributed, open-source version control system. Basically, Git works by saving and sending snapshots of a project to a remote repository. Other people can then commit and pull changes to this repository. Being distributed, with Git there is no urgent need for hosting a version management service online, because every Git repository contains the entire unabridged history of the project.

Git was originally created by Linus Torvalds, to aid in the development of the Linux kernel. Even though it is traditionally a command-line program, lacking a graphical interface, many such third-party programs have surfaced since Git's inception.

2.3.2 GitHub

GitHub is a social hosting service strictly for Git repositories. GitHub is a very popular medium for developers around the world to collaborate with each other on different

projects. Users can create and host their own public Git repositories for free, but private repositories cost a monthly fee. Users can also interact with other users by contributing to their projects, for example, in the form of pull requests. Besides just hosting repositories, it also contains many additional features such as issue tracking, a wiki for documentation and enhanced pull request flow. (Github.com)

During this project, Git and GitHub were mostly used just as a practical way to back up and version both the application code and the written thesis itself. A private repository was created for the thesis. The commit history of the repository is visualized in Figure 6.

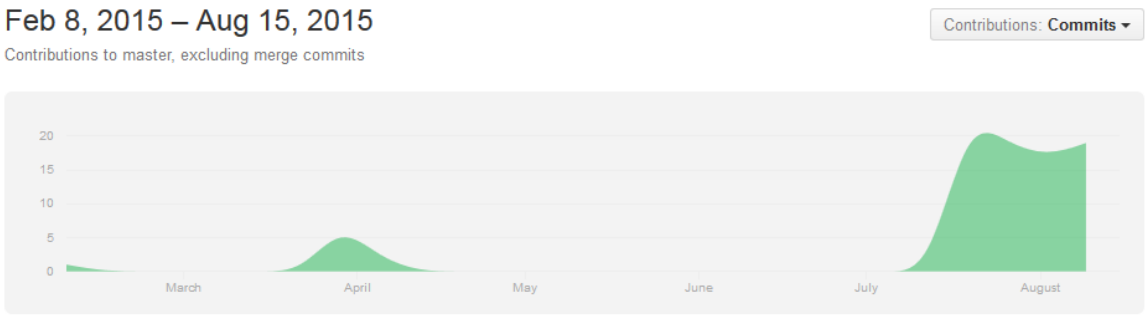


Figure 6. Thesis repository activity

3 THE ANDROID APPLICATION

3.1 Introduction

After much planning and testing, the final mobile application used to control the robot itself became quite simple in its functionality. Many features were 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, namely being easy to use and looking pleasant. Also, a lot of the different configuration settings were dropped or heavily simplified, as they did not really bring any added value to the end-user.

The application itself consists of just two main activities. For all intents and purposes, in the context of Android programming one can think of activities to be the various different “screens” that make up a whole mobile application. Each activity has a separate XML-based layout file, and its own related Java file, which contains all the logic and code related to the activity in question.

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 joystick interface.

3.2 Design and planning

3.2.1 Introduction

In terms of designing the final look of the application, most of the hard work was already done in the context of the previous project. Just the color scheme of the application was changed, from the default black-and-white palette to a darker one.

However, the fact that the application would be used by a person not familiar with the project did bring some additional design constraints that had to be kept in mind during the development process. The original Android application was developed in a rather fast pace, and it contained a lot of superfluous settings and such, mostly meant for debugging purposes, as can be seen in Figure 7. When starting up the application, the most likely intent of the user is to want to start driving around the robot as soon as possible. Therefore, it makes the most sense to give all the configurable settings functional default values, and streamline the process of getting to the controlling part of the application.

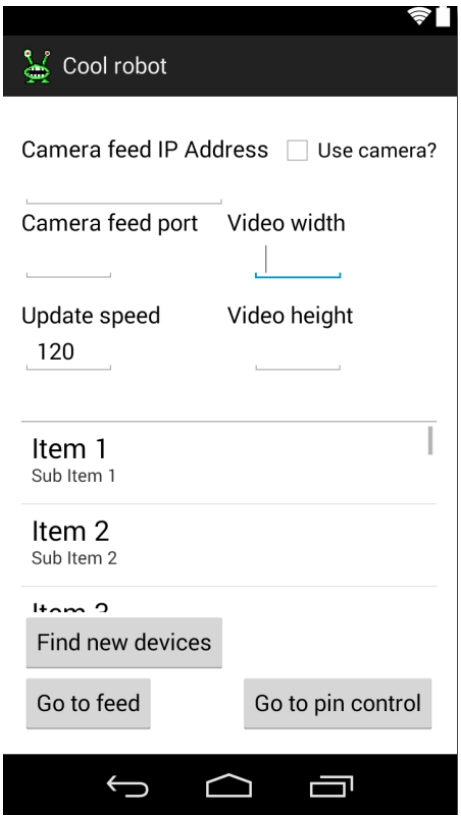


Figure 7. Old application settings activity

Of all the superfluous settings, probably the worst offenders are the camera port and video size related inputs on the topmost section. Unlike the IP address, there is really no way for either of these variables to change from its default setting of 5000 on the Raspberry Pi side. In the new version of the Android application, the port field was removed, and the two size-related inputs were replaced with a single dropdown.

Furthermore, the original mobile application also had a large list of Bluetooth devices at the center of the activity. Upon clicking the button labeled “Find new devices”, scanning of active Bluetooth devices would begin, and found devices would then be appended to the end of this list. This approach is quite necessary, mainly because it takes a lot of space and generates a great deal of unnecessary clutter. In the new version, the list would be moved to an entirely separate dialog.

Due to being quite simple in its layout, the feed activity used to control the robot did not need major redesigning. The new version turned out to be nearly identical to the original one, with the exception of some additional buttons and an indicator displaying the strength of the Bluetooth signal.

3.2.2 Usability on tablet devices

One of the objectives for the new version of the Android application was that it should be usable on Android tablets. Developing on tablet devices is quite similar to developing on mobile phones. The only major exceptions are a general lack of features very characteristic of mobile phones, such as making phone call or sending SMS. More relevant to this project in particular is the fact that tablets often have much larger screens. Instead of ordinary pixels, in the context of Android programming, screen size is more often presented in density-independent pixels, or dp for short. One dp has officially been defined as equivalent to one physical pixel on a 160 dpi screen. (Android.com).

To help developers in providing a consistent and pleasant user experience no matter what the size of their device, Android offers a variety of ways. For the most control, you can provide separate layout files for different screen sizes. In practice, one just needs to make a new subfolder under the resource folder of their Android project, and name it accordingly. For example, if one wants to specifically target big tablet devices, they might create a folder called “layout-large”, and put their size-specific layout files there.

In smaller applications such as this one, providing different layout files for different devices is not really necessary. With careful planning and styling of elements, one can create layouts that work well both on mobile and tablet devices. Besides using the aforementioned density-independent pixels for indicating the width and height of graphical elements, the Android API also offers ways to define size relative to the element’s parent container. For example, in Figure 8 the width of the two buttons labeled “Find devices” and “Go!” are defined with the value “fill_parent”. This causes the buttons to be as wide as their parent element, which in this case, is the activity itself. Because of this, the buttons look quite similar no matter what the screen size.

3.3 Settings activity

3.3.1 Layout

This is the view where the application starts in, so in essence it might be called the "main" activity of the whole application. As can be seen in Figures 8 and 9, there are many different options that can be configured through this particular activity. For example, the user can 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 is also the option to disable the video entirely, since during testing it was found out that it might slow down some older phones somewhat.

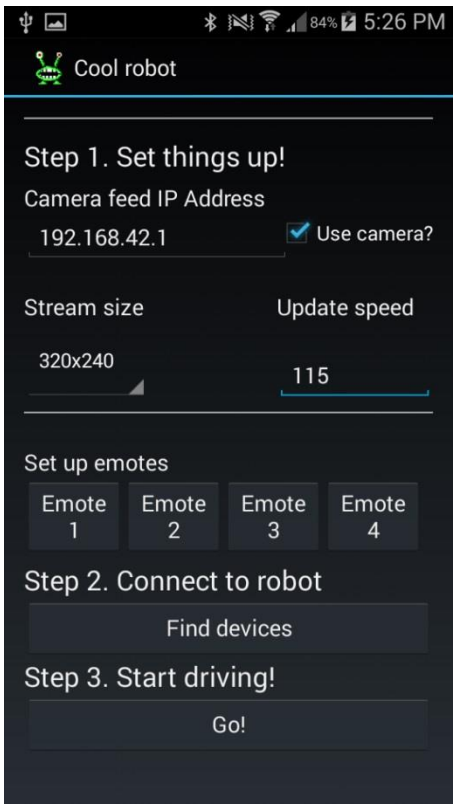


Figure 8. Settings activity

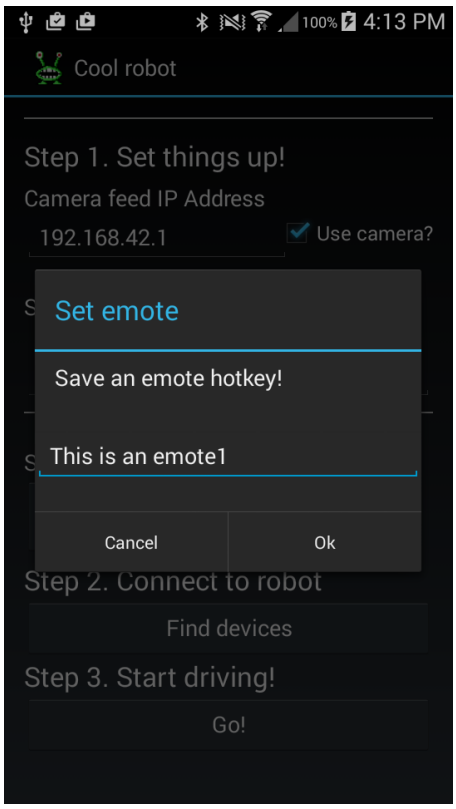


Figure 9. Setting an emote

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 driving the robot feel 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.

Below the video settings is a line of four buttons used to set up different emotes. Pressing these allow you to set up predetermined phrases for the robot to say, so on the robot feed activity you do not have to open up the message dialog every single time you want the robot to say something.

From the settings the user 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 as they are found. From this list the user can select devices and attempt to pair and form a connection, after which he or she 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, which is the robot feed activity.

3.3.2 Programming

The main functionality of the code behind the Settings activity is to simply keep tabs on some different kinds of 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 activity worth mentioning is the pairing of Bluetooth devices, which is handled in this view. 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 the stream activity.

The job of the BluetoothStreamManager 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 relevant function that writes the data to the output is as follows:

```
public void writeData(byte[] data)
{
  if (connectionState == STATE_CONNECTED)
     {
      BluetoothGattService rxService = bluetoothGatt.getService(RX_SERVICE_UUID);

      if (rxService == null)
         {
            return;
         }

  BluetoothGattCharacteristic RxChar = rxService.getCharacteristic(RX_CHAR_UUID);

            if (RxChar == null)
            {
                        return;
            }

            RxChar.setValue(data);
            bluetoothGatt.writeCharacteristic(RxChar);
     }
}
```

The BluetoothGattService is a class that represents a GATT service. GATT is an acronym that stands for Generic Attribute Profile. A Bluetooth GATT Service contains one or more predefined data formats that are called Characteristics. Each Service is identified by a unique id called an UUID. In this case, we simply want to write data to the Bluetooth LE module, so we fetch the appropriate Characteristic by a predefined UUID, set its payload of bytes to send, and then write the characteristic to the Bluetooth module.

There exists quite a large list of official standardized Bluetooth services online, from tracking blood pressure to body composition. These specifications can be found on the official Bluetooth Developer Portal. As an important side note, no such specification exists for Serial UART for whatever reason. Because of this, Adaruit has

written their own unofficial custom UART GATT server that runs on the Arduino, and can be used for simple serial communication.

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. This dialog prompts the user to return to the Settings activity and attempt to repair devices.

In the previous iteration of the Android application, the class that managed the Bluetooth connection was quite a bit more complex. Activities using the class could not write data to the Arduino directly, nevertheless, 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.

3.4 Robot feed activity

3.4.1 Layout

This is the part of the application where most of the action happens. As can be seen in Figure 10., The majority of the activity is taken by the video feed. There are some transparent controls located at the bottom. There are two “joysticks” on the bottom-left and bottom-right parts 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.

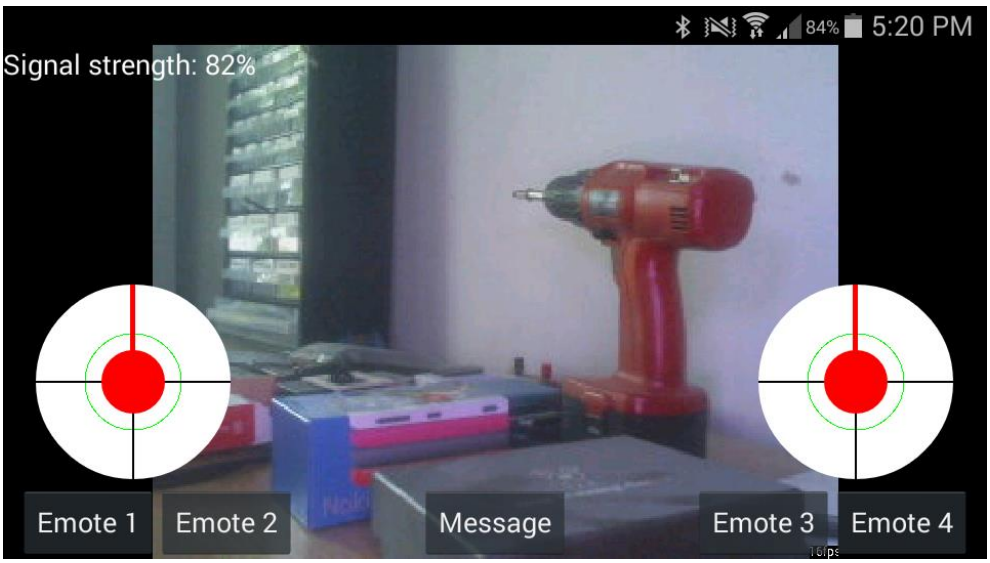


Figure 10. Robot feed activity

The button on the bottom labeled “Message” opens up a simple dialog with a text box, as seen in Figure 11. 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. The four other buttons labeled with “emote” can be used to make the robot say the predetermined phrases that were set in the settings screen.

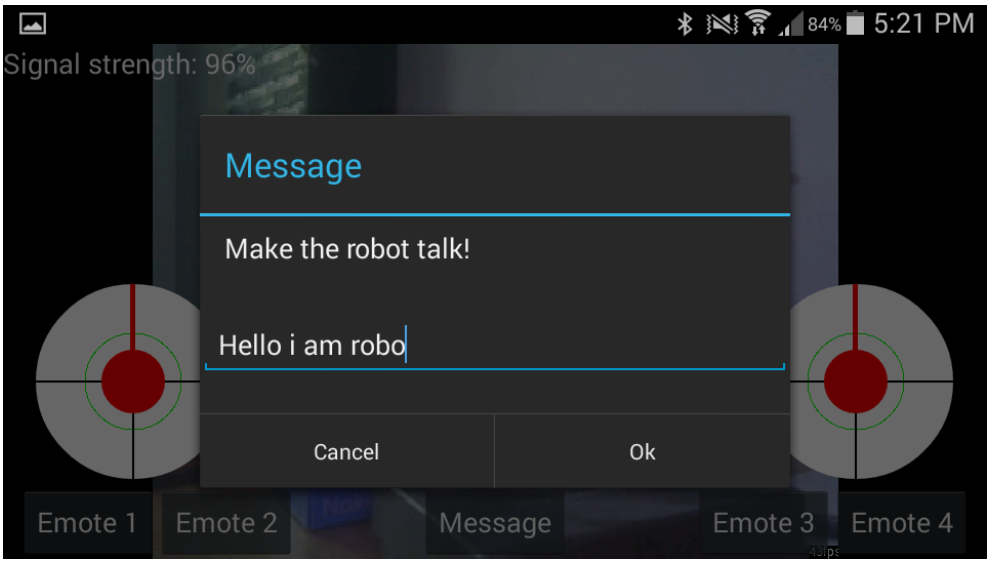


Figure 11. Robot message dialog

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 and shown to the user as a more intuitive percentage value.

3.4.2 Programming

The most important job of the logic behind the Feed activity 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 side of the embedded program. 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 webcam, server, provided the user has previously enabled that particular feature in the settings activity.

The two joystick components on the bottom of the screen 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. These changes are mostly related to the speed at which the joystick event listeners update themselves. Originally, they were too slow. However, it was a simple fix to increase their update speed, therefore improving the latency of the robot.

4 THE ROBOT

4.1 Introduction

As illustrated in Figure 12, physically the new build of the robot looks quite similar to the old one. At a glance, the most significant physical differences are the new, larger eyes and mouth. Building the actual robot somewhat bit easier than in the previous project, as there was a much clearer general idea of parts needed and how they would fit together. Most of the new challenges in the actual building phase of the

project came from the increased power requirements. All in all, building the new robot took about 50 hours, total.

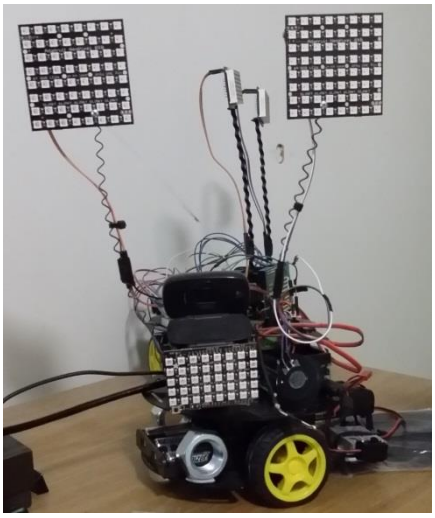


Figure 12. Finished robot

4.2 Overview of robot parts

4.2.1 Table of components

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. Refer to Figure 9 to see how the parts come together to make the final build.

Table 1. Robot parts and prices

Part	**Cost (€)**
Pan + tilt servo	30
4000 mAh LiPo 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
W-iFi adapter	20
Arduino Mega	45
	Total: 406,12 €

4.2.2 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, the Arduino Uno was used. The Uno is slightly cheaper than the Mega, and comes with somewhat less memory, pins and various other features. The Arduino Uno has 2,048 bytes of memory, and the Mega has four times as much, which is 8192 bytes. The small amount of memory in the Uno ended up being deciding factor for the upgrade, as it turns out the libraries needed to interface with the Bluetooth module and the NeoPixels needed a lot more memory than was anticipated. This put the size of the final program just over the 2,048 byte limit. The Uno and the Mega both run at a clock rate of 16MHz, which means both are equally fast in terms of computing power.

Besides having more memory, the Mega also has a lot more pins for input and output, with 54 digital pins and 16 analog pins. For reference, the Uno has 14 digital pins and 6 analog pins. However, the large amount of pins in the Mega is not really put to use in this particular project, as the upgrade was made strictly due to memory issues.

4.2.3 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 metal plates with a variety of mounting holes and rails, a standard battery pack that can hold up to 5 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. It was chosen because it had proved to be a good and affordable 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, to remove excess interference.

4.2.4 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 must be 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.

This particular make of servo was chosen because it was already tested during the previous project, and they proved to be a fine solution.

4.2.5 LiPo battery and UBEC

Nearly the whole robot is powered by a single 7.2 volt lithium polymer battery, originally designed for RC cars. The exceptions to this are the motors and the servos, which are powered from a different source. The reason for this is to avoid instability-causing voltage drops under heavy loads. 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.

Since all the components 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 output of 5 volts. 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 its role as a portable, easy-to-use power source quite adequately. The reason it was not used in the new robot was because its maximum output was only 3 amps, and the new robot uses quite a lot more than that at its peak consumption. The most significant reasons for this increased demand for power 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.

4.2.6 Adafruit Motor Shield V2

In this project an Arduino add-on called the Adafruit Motor Shield was used to control the various motors and servos. The main advantage of using a ready-made add-on such as the motor shield in is ease-of-use. Most of the “heavy lifting” can be safely left to the motor shield, and the builder does not have to worry about issues such as protecting their microcontroller from unwanted power surges.

As an additional benefit, the Motor Shield has a universal PCB board designed into it. This turned out to be of great help during the making of the project, as the final circuit could be implemented in a somewhat neat and orderly manner on this built-in board, as can be seen in Figure 13. The original robot had a separate breadboard that contained most of the physical wiring and components.

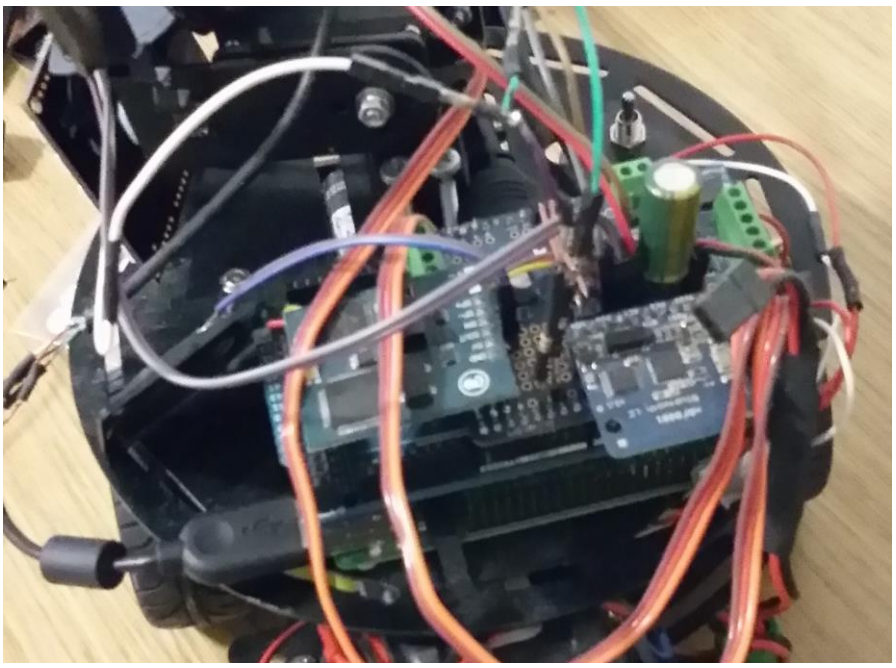


Figure 13. Back side of the robot

4.2.7 NeoPixel shield and matrix

Purely for visual show, the robot contains a total of 173 full color LEDs. Two 8x8 matrixes 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.

The NeoPixel were chosen for the new prototype because they are very bright, colorful and easy-to-use. However, it is true that choosing NeoPixels over traditional single-color LEDs did bring some additional engineering challenges, due to the large increase in both memory and power consumption. Keeping this in mind, the ultimate end-result still looks excellent, with vibrant and bright colors. In fact, the pixels are so bright, in the final they are lit up at essentially 10% of their maximum brightness. This is because at full power the colors will eventually get muddled together, not to mention extremely bright lights are just generally unpleasant to look at.

4.2.8 Emic 2 Text-to-speech module and speaker

The Emic 2 is a standalone voice synthesizer that takes in a stream of characters through UART, and then attempts to convert it to audible speech. This makes it quite

easy to integrate into embedded projects, one need only to provide it with power and send serial data for it to pronounce. Naturally, the module also needs a small speaker attached to it to function as intended.

The Emic 2 has quite a large variety of different features. It can speak English or two different dialects of Spanish, Castilian or Latin. The Emic also offers 9 different speaking voices, and the option to set its volume and rate of speech. It was chosen for this project mostly for the sake of curiosity. The Emic 2 is probably the best text-to-speech chip of its kind on the market today, as it manages to sound a lot more natural than its competitors. However, at 59.95€ the Emic 2 is also the single most expensive component in the entire robot. If a new robot were built, it would be worthwhile to try using the Raspberry Pi for speech synthesis.

4.2.9 Raspberry Pi 2 with webcam

The Raspberry Pi 2 is the newest model in the popular 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 custom 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 was not available to the public at the time. Like previously mentioned, in the original project the phone itself hosted the Wi-Fi access point, which the Raspberry would join. The reason for 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, which made the old system very unintuitive for the end-user.

Furthermore, one of the requirements for the new application was that it should be usable on tablets also. Since most Android tablets cannot use SIM cards, they do not have access to the Internet through conventional means, and can only connect to a Wi-Fi access point, not host one. For most tablet users, this missing feature makes little to no difference, but not in this particular project.

4.3 Assembly and building process

In appendix 1, you can see the way that the finished robot is put together, and how all the interconnected parts communicate. This diagram is somewhat simplified when compared to the actual set-up, but it should give a concise overview of the robot. Probably the most blatant victim of oversimplification is the power management system. In reality, the UBEC does not just power the Arduino Mega, but also every single other component in the robot, from the Raspberry Pi to the Arduino. This visualization was intentionally left out from the diagram, simply because it would take too much space. All the used pins and connections from the Arduino Mega to the various components can be seen in table 2.

Table 2. Used Arduino pins

Arduino pin	**Connection**
0	Emic 2 Serial TX
1	Emic 2 Serial RX
2	Bluetooth module SPI RDY (ready)
3	Bluetooth module SPI RST (reset)
5	NeoPixel LED eyes
6	NeoPixel LED mouth
8	Bluetooth module SPI REQ (input)
12	Bottom servo
13	Top servo
50	Bluetooth module SPI MISO (data out)
51	Bluetooth module SPI MOSI (data in)
52	Bluetooth module SPI SCK (Clock in)

In practice, the development of the robot was done in separate, isolated systems, which were tested and then finally integrated together into the finished version when they were functioning. For example, animating the various LEDs was developed entirely separately from the rest of the robot itself. During the development phase the Arduino Uno was used, as it was cheap and easy to acquire, and the large

memory requirements of the final program did not come into play until it was time to finally put the whole program together.

Much like when building the earlier version, the project was begun by assembling the chassis. This task was quite easy, as it was the exact same platform that was used in the previous project. The same goes for the servos. In Figure 14, you can see the barebones skeleton of the robot. None of the LEDs were wired up at that point, and they were temporarily attached to the robot as an attempt to sketch out the look of the final version.

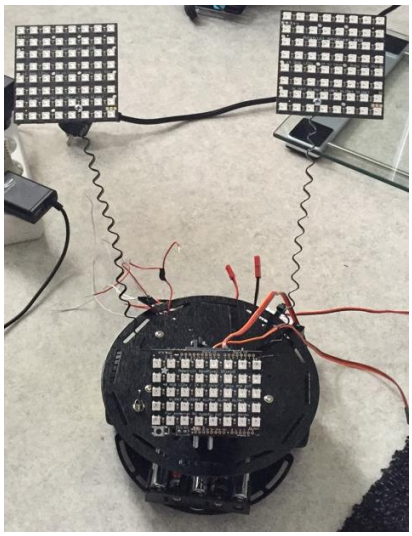


Figure 14. Bare robot chassis

After the initial body of the robot was assembled, the next phase was to test out the Bluetooth LE module. Hooking it to the Arduino was not all that difficult. The major problems came on the Android side, when attempting to form a connection between the devices. Bluetooth LE is a relatively new technology, and its usage is more prevalent on Apple devices running the iOS operating system, so there were not a lot of examples on how to use it. It bears mentioning that, at this point in the project, the Arduino Uno was used, as the memory limitations were not realized during the early phases yet.

Working with the NeoPixels turned out to be quite easy, also. Much of the same animation-related code for the eyes could be reused from the previous robot. The mouth animations, however, had to be done from scratch. This is because the NeoPixel shield is just a LED regular matrix, and does not come with the same text-

scrolling functionality as the previously used Pi-Lite component. It was at this phase that the memory limitations of the Arduino Uno became apparent, and a switch to the more powerful Mega series was done.

After the LEDs were done, it was realized that the USB power bank with a maximum output of 3 amperes would simply not be powerful enough. Many alternative solutions for enhanced power output were looked into, such as adding another 5 AA battery pack. Ultimately the LiPo battery proved to be the optimal solution, as they are generally lightweight, cheap and can output a lot of current if needed. The only foreseeable downside to using LiPos is that they need a specialized charger in order for them to be reused safely.

After putting the rest of the robot together, the last thing that remained was to set up the Raspberry Pi webcam server. At first, this seemed like an easy task, as setting up the stream had already been done in the previous robot, and all the components used were exactly the same. The only difference would be that it was the Raspberry Pi hosting the Wi-Fi access point, not the mobile device. This seemingly minor difference turned out to be an issue, as the chipset used by the USB Wi-Fi module did not correctly support hosting an access point. This is somewhat ironic, as the Wi-Fi adapter was advertised as an official Raspberry Pi product, and it would make sense for it to straight out of the box. This problem was solved by purchasing a different Wi-Fi adapter, which ended up working straight away.

5 THE EMBEDDED PROGRAM

5.1 Introduction

In essence, the whole embedded program running on the Arduino is just a big state machine. This state machine is further illustrated in appendix 2. The two main jobs of the embedded program are handling the animating of the eyes and mouth, and reading bytes from the Bluetooth module and handling them accordingly.

As previously mentioned, all Arduino programs consist of a setup function which is run once, and a main loop which is run indefinitely after the setup. In this particular program, the setup function 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. The amount of time the mouth should be animated is calculated from the length of the message after it has been completely read from the Bluetooth module.

```
281 void loop() {
282
283   BTLEserial.pollACI();
284   handleSerialInput();
285
286   currentCmdTime = mouthTimer = eyeTimer = messageCurrentTime = millis();
287
288   eyeAcc += eyeTimer - eyeLastTimer;
289   eyeLastTimer = eyeTimer;
290
291   mouthAcc += mouthTimer - mouthLastTimer;
292   mouthLastTimer = mouthTimer;
293
294   while (eyeAcc >= 50)
295   {
296     animateEyes();
297     eyeAcc = 0;
298   }
299
300
301   while (mouthAcc >= 100)
302   {
303     if (isTransmittingMessage)
304     {
305       animateMouth();
306
307
308       if (messageCurrentTime - messageStartTime >= messageEstimatedTime)
309       {
310         resetRobotMouth();
311         isTransmittingMessage = false;
312         memset(&robotMessage[0], 0, sizeof(robotMessage));
313       }
314     }
315     mouthAcc = 0;
316   }
317 }
318
```

Since the Arduino is not running anything resembling an operating system, it does have a concept similar to threads. Because of this, we use a timer system, where each iteration, we get the current time since starting up the microcontroller with the millis function of the standard Arduino library. Then, for each different timer we calculate the time that has passed since the last action. If this time reaches a certain threshold, we perform an action and reset the timer. What this does in essence is allow us to simulate the Arduino performing multiple actions every x milliseconds.

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.

5.2 Communication between Android application and robot

5.2.1 Introduction

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.

5.2.2 Motor command

Here is an example of sending a simple movement command. The square brackets represent the cells in the array, and the inner number represents their value:

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

The first number is a simple delimiter, which lets the microcontroller know that the following five characters should be interpreted as a movement command, and not as 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.

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

The fourth and fifth numbers follow the same logic 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 simply stay still.

5.2.3 Message command

Compared to the motor command, sending messages for the text-to-speech module to pronounce is somewhat simpler. Since we are talking about text, the following example will present the bytes as their ASCII character equivalents.

```
z Hello! \n
```

The byte that corresponds to the letter z, which is 122, marks the start of a new displayable message. This delimiter is followed by an undefined amount of other characters, and finally the newline character to implicate that the message has ended, and is ready to be sent to the text-to-speech module via UART. Sending this particular series of bytes from the Android application would cause the robot to say the word “Hello” out loud.

As a limitation of sending each character as a regular byte, the robot’s means of communication is strictly restricted to ASCII characters. These characters are validated on the Android side before sending the message. The message is also limited to 40 characters, both on the Android and embedded side.

6 RESULTS AND CONCLUSION

When it comes to creative projects, it can sometimes be difficult to predict the quality and nature of the final outcome. 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 great deal of this had to do with the fact that much of the work in the planning stages could be adopted from the previous project.

The most significant difficulties were 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 being a lot of documentation and examples and such. However, this was not a huge issue in the end. There were also some problems with setting up the Raspberry Pi 2 Wi-Fi access point, but this was resolved by switching the Wi-Fi adapter used to a different model.

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 to maintain bearable latency from the perspective of the user. Also, as the name “Low Energy” implies, Bluetooth LE is intended for embedded devices with fairly restrictive power requirements such as heart trackers and other wearable technologies. The power consumption of the module 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, compared to the LEDs for example. There is also the issue of cost and ease of implementation, as Bluetooth Classic modules can be acquired quite cheaply online, and there exists a wealth of documentation for their usage.

In the future, an interesting experiment would be to attempt an entirely Wi-Fi based communication method, an abandoning Bluetooth entirely. The Raspberry could run a web service that could be interacted with the Android application. The Pi would then be connected to the Arduino, and it could relay the bytes to the Arduino through UART. This could possibly increase the range of the robot by a few dozen meters or so. It is difficult to say how latency might be affected with a HTTP-based solution, but as an educated guess controlling the robot via Wi-Fi might cause some overhead, and therefore an increase in latency.

It would also be worthwhile to look into using other Bluetooth LE modules, mainly the nRF51822. They are sold by the brand name Bluefruit LE Friend by Adafruit, and they come in both SPI and UART variety. The nRF51822 is somewhat cheaper and a bit more powerful than the nRF8001. That aside, the more important distinction between the LE Friend and the currently used nRF8001-based module is the fact that the library used to interact with the LE Friend is significantly more compact. The Bluefruit library only takes about 426 bytes in total, while the nRF8001-based library is over twice as big, taking about 1,029 bytes. Changing modules would mean that the robot could switch back to using the Uno, as there would be no need for the extra memory of the Mega 2560. For reference, the Uno has a total amount of 2,048 bytes of dynamic memory, while the Mega 2560 has a maximum of 8,192 bytes.

All in all, both the customer and the builder consider this project and its end result to be a definite success. The new robot looks visually impressive, its code has less bugs and it has more features. In in Figure 15, you can see the new robot and the earlier robots can be seen, side by side. Hopefully the new robot will bring joy to people who get to experience it, and may it inspire others to get into electronics and programming for many years to come.

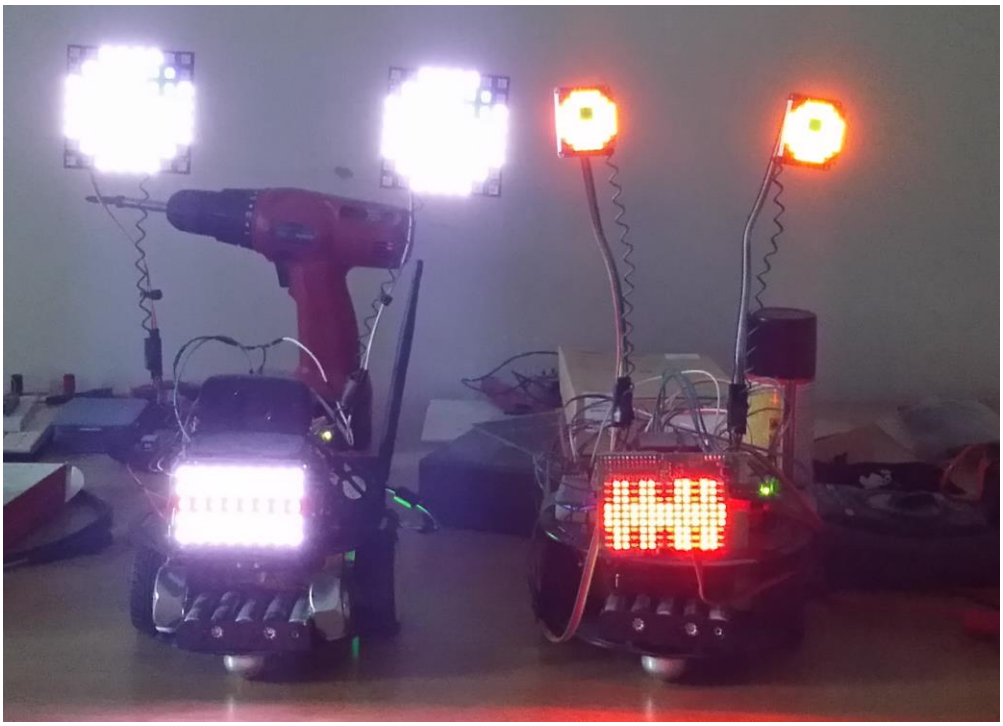


Figure 15. The new robot and the old robot

REFERENCES

Arduino.cc. What is Arduino? Accessed on 4 August 2015. Retrieved from https://www.arduino.cc/en/guide/introduction

Arduino.cc. A Brief Introduction to the Serial Peripheral Interface (SPI). Accessed on 5 August 2015. Retrieved from https://www.arduino.cc/en/Reference/SPI

Arduino.cc. Arduino Shields. Accessed on 4 August 2015. Retrieved from https://www.arduino.cc/en/Main/arduinoShields

Android.com – Supporting Multiple Screens. Accessed on 17 August 2015. Retrieved from http://developer.android.com/guide/practices/screens_support.html

Android.com – History. Accessed on 4 August 2015. Retrieved from https://www.android.com/history/

Eason, J. 2015. The Android Developers Blog. Accessed on 4 August 2015. Retrieved from http://android-developers.blogspot.fi/2015/06/an-update-on-eclipse-android-developer.html

GitHub.com. Features. Accessed on 6 August 2015. Retrieved from https://github.com/features

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

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

Speedguide.net. How does RSSi (dBm) relate to signal quality (percent) ? Accessed on 8 August 2015. Retrieved from http://www.speedguide.net/faq/how-does-rssi-dbm-relate-to-signal-quality-percent-439

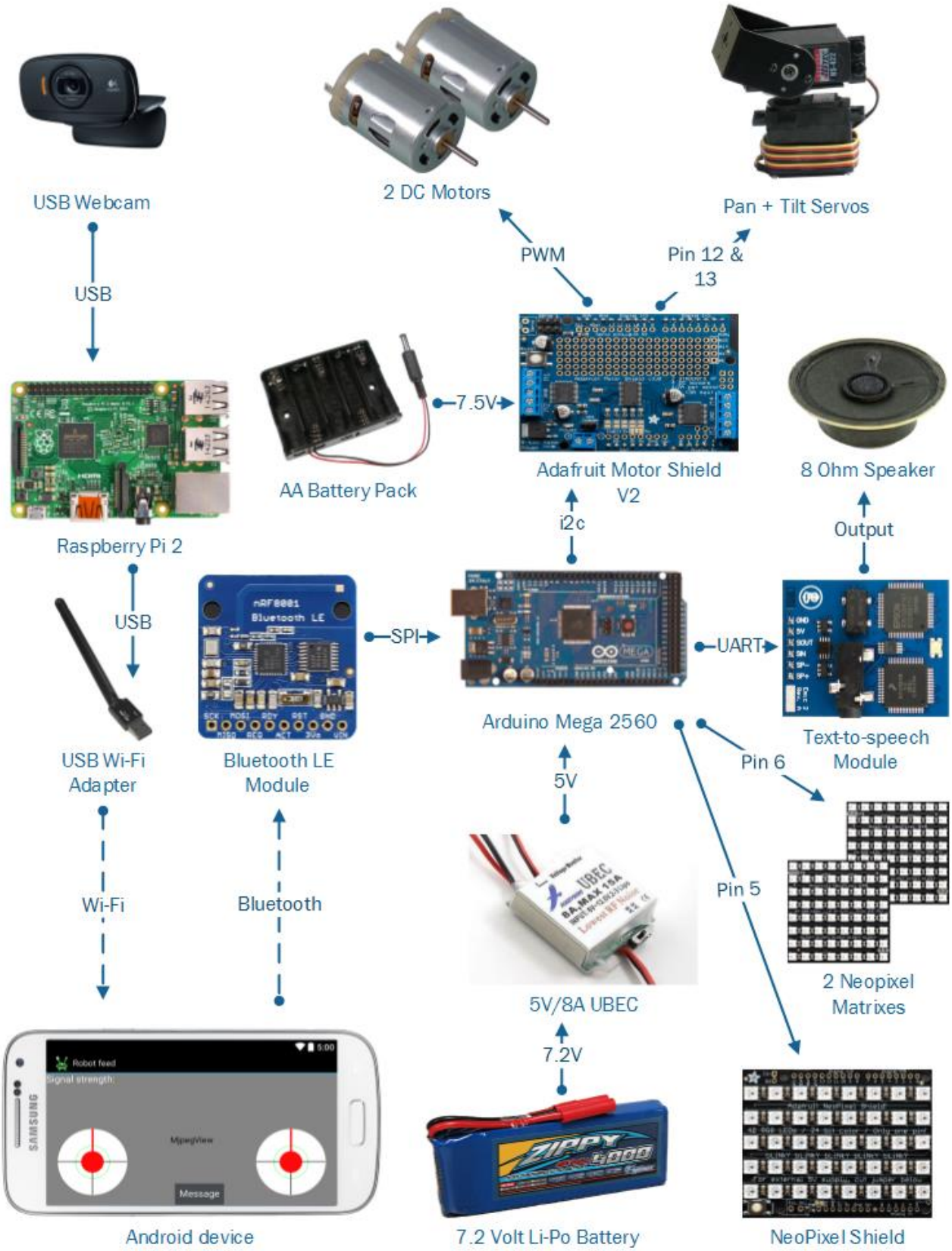
Stroustrup, B. 2014. Lecture: The essence of C++. Accessed on 29 July 2015. Retrieved from https://www.youtube.com/watch?v=86xWVb4XIyE

The Eclipse Foundation. Desktop IDEs: Accessed on 4 August 2015. Retrieved from https://eclipse.org/ide/

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

APPENDICES

Appendix 1. Robot parts diagram



Appendix 2. Robot UML activity diagram