# Motion-Controlled RC Car

**Final project report**

Name: Marcel Troscianko

Matric No: 40173086

Abstract

With the recent rise in Virtual Reality and motion control technology, surprisingly little advancements have contributed to real-world, physical toys and gadgets. This project aims to show that the same technology used for moving characters in a virtual game can be used to control a physical toy car. This will be done by creating a head-mounted sensing system connected wirelessly to a self-constructed remote-control car mounted with a movable camera – something that could very well be replaced by a true VR headset in future versions of the project. For now, the car and its camera will be connected to an Android mobile app, which will be built to help calibrate the car in case of calibration drift as well as to display the output of the cameras.

The devices will be linked wirelessly and the processing will be done on the headset so it can be connected to any other model of car and keep its calibration.

The project aims to prove that the technology used in drones and VR can be used for innovative, immersive toys. The project aims to deliver the Remote Control (RC) Car, the Headset Motion-Sensitive Controller, and the Android mobile app.

The deliverables from the project will be a drone, a sensing and controller headset, and a companion app for calibration

**Declaration of authorship: I confirm that the work submitted is mine and that wherever possible the work of others has been clearly acknowledged and referenced.**

**Signature: Marcel Troscianko Date: 09/04/2017**

Contents

[Motion-Controlled RC Car 1](#_Toc479544845)

[1 Table of Figures 3](#_Toc479544846)

[2 Introduction 4](#_Toc479544847)

[2.1 Background/Context 4](#_Toc479544848)

[2.2 Aims and Deliverables 4](#_Toc479544849)

[3 Design Choices 4](#_Toc479544850)

[3.1 Introduction 4](#_Toc479544851)

[3.2 Design Choices – RC Car 4](#_Toc479544852)

[3.3 Design Choices – Motion-Sensitive Controller 6](#_Toc479544853)

[3.4 Design Choices – Mobile App 7](#_Toc479544854)

[4 Test Plan – RC Car 9](#_Toc479544855)

[5 Test Plan – Motion-Sensitive Controller 10](#_Toc479544856)

[6 Test Plan – Mobile Application 11](#_Toc479544857)

[7 Test Results 11](#_Toc479544858)

[8 Conclusions and Future Work 11](#_Toc479544859)

[9 Self-Evaluation 12](#_Toc479544860)

[Bibliography 12](#_Toc479544861)

[Appendix A – System Software 12](#_Toc479544862)

[Appendix B – System hardware 27](#_Toc479544863)

[Appendix C – Detailed test results 28](#_Toc479544864)

[1 RC Car 28](#_Toc479544865)

[2 Motion-Sensitive Controller 28](#_Toc479544866)

[3 Mobile Application 29](#_Toc479544867)

[Appendix D – Initial Project Proposal 29](#_Toc479544868)

[Arduino Head-Mounted Drone Controller 29](#_Toc479544869)

[1 Introduction 30](#_Toc479544870)

[1.1 Background/context 30](#_Toc479544871)

[1.2 Aims and deliverables 30](#_Toc479544872)

[Appendix E – Interim report 32](#_Toc479544873)

[Title: Head-Controlled Arduino Vehicle 32](#_Toc479544874)

[1.1 Deliverables 32](#_Toc479544875)

## Table of Figures

[Figure 1 (left) and Figure 2 (right). Left: “Practical 5” design, Right: Design used in this project) 5](#_Toc479545030)

[Figure 3: Motion-Sensitive Controller Device 6](#_Toc479545031)

[Figure 4: App “Home” Page 8](#_Toc479545032)

[Figure 5: App “Calibrate” Page 8](#_Toc479545033)

[Figure 6: App "Select Device" Page 8](#_Toc479545034)

[Figure 7: App "Control" Page 9](#_Toc479545035)

[Figure 8: Interim Car Build 27](#_Toc479545036)

[Figure 9: IPP Timeline 31](#_Toc479545037)

[Figure 10 (Left) and Figure 11 (Right): Interim Report controller and RC car circuits respectively 33](#_Toc479545038)

[Figure 12: Interim Report Demo App 34](#_Toc479545039)

[Figure 13: Interim Report Timeline 34](#_Toc479545040)

## Introduction

### Background/Context

In the first weeks of the module, we had to think of a project for the coursework. Having recently played videogames that include remote-controlled drones and being intrigued by them at a younger age, I have decided to create one for myself. Remote-controlled cars were already part of the tutorials though, so simply recreating that as the full project wouldn’t be enough. It was decided that instead of using a mobile phone for direct control, it would use a head-mounted remote control with a gyroscope – the car would turn when the remote-control turns, thus allowing headset control, and the mobile phone would simply be used to adjust the car’s calibration.

Doing this shouldn’t be difficult – both the car and the controller would need their own Arduino microcontrollers and Bluetooth transceivers to work in sync, as well as motors, a gyroscope, and an extra Bluetooth device to connect the mobile app. The app itself needn’t be complex in visual design – it’s simply a calibration app after all. It will consist of simple screens with buttons and text fields.

The inspiration for this came from a session I had on a VR headset, therefore it would make sense to also include a camera on the car that would stream directly to the mobile app – this way, future iterations of the project could replace the single camera with two of them, and it could be moved over to VR headset control and streaming allowing first-person control of a car. VR, however, is far outside the bounds of the funds dedicated to this project so this iteration won’t include that.

### Aims and Deliverables

These have not changed from the IPP and Interim Report and they are as follows:

* Custom-built remote controlled car.
* Headset remote control with gyroscope
* Mobile app for calibration

## Design Choices

### Introduction

This project was the first of such kind I’ve created. Not much was known about the ins and outs of hardware development besides the lectures and practicals taken at University, and as such the project had to be well designed and the components extensively researched to ensure everything would work. Most components were bought specifically for the project too, so replacing components could take it over the deadline. This section is broken down into four sections – one for each isolated deliverable of the project, and one with the listing of all components and their statistics.

### Design Choices – RC Car

This was the simpler of two hardware systems to build so it’s the first to be covered. The hardware itself was simple – it uses the same basic setup as the tutorial [1], but with the addition of two Bluetooth transceivers to connect both the remote control and mobile app. The Arduino was powered by a simple 9V battery, but the motors required a separate battery pack so extreme care had to be taken to avoid frying components when using both currents on the same circuit.

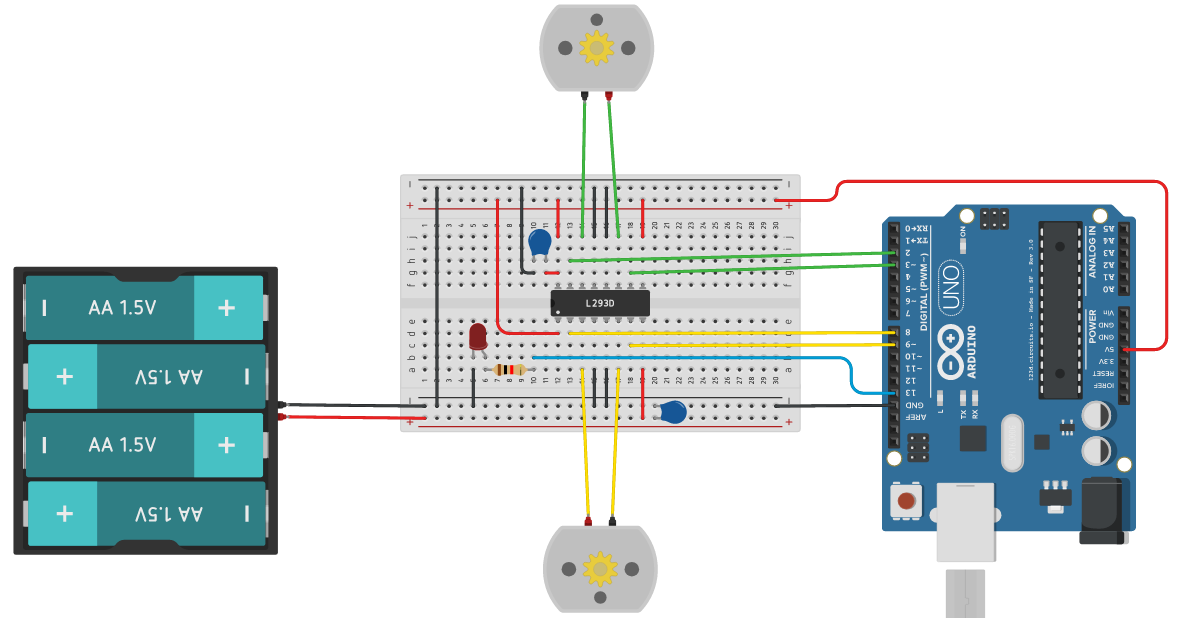
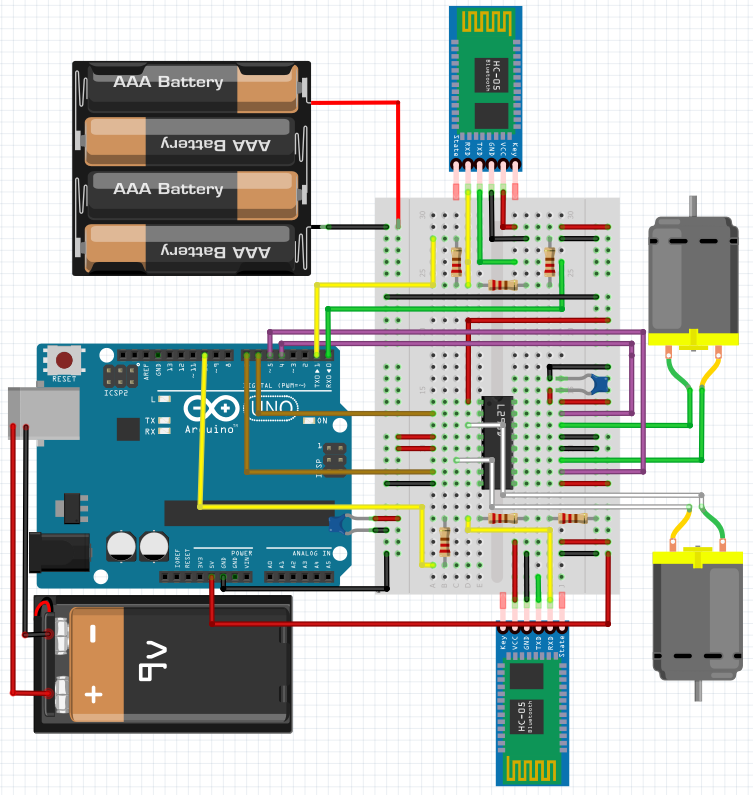
 

Figure 1 (left) and Figure 2 (right). Left: “Practical 5” design, Right: Design used in this project)

The choice between using Bluetooth and Wifi for this was tough. So much so that both were investigated for how they may complement the project, a table of which is shown below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Range | Progamming | Power Consumption | Cost |
| Wifi | 100m | Difficult | High | Medium |
| Bluetooth | Stable: 10m  Max: 100m | Easy | Low / Medium | Low |

It is also worth to note that Bluetooth can sustain at most 7 devices – though this is more than enough for this project.

Note: A comprehensive list of devices and their specifications is included in Appendix B: System Hardware

After some consideration, Bluetooth was chosen as the networking device for this project because it’s both easier to manage in-code making the app easier to develop and, more crucially, a low power consumption. The maximum distance is a potential problem for a real-life application, but 10m is more than enough for the coursework demonstration.

The software for this proved trickier. Instead of simply hardcoding in the direction, it would have to read them via Bluetooth. The way this was tackled was with the development of a simple communication protocol based on JSON. In JSON, the way information is stored is the field name followed by its value, e.g. *{ “left\_turn”: 50.0 }*. The mobile app or remote control will first tell the car *what* it is to do, following by *how much* of it to do. This counteracts any issues with sending over vectors and or complex commands. The commands also don’t necessarily have to be immediately readable – an Arduino-based Bluetooth information package should be as small as possible to as many as possible to be sent. The high-level logic of this system can be seen below:

|  |
| --- |
| System receives a single character from the controller (‘l,’ ‘r,’ ‘f,’ b.’).   * The system sets the “expecting” flag to the direction character (i.e. EXP\_NEXT = EXP\_RIGHT) * The system is now expecting an integer value.   The system receives an integer value from the controller.   * The system turns the motors defined by the current direction flag by the integer value |

This was the basic view until it became clear that the motors can’t be turned by different amounts. This was a completely unforeseen complication with many sub-problems – the spin-up time of motors makes turning inaccurate and the turning amount is no longer the angle, instead the time the motor will be turning for. This put a lot more emphasis for the calibration inside the controller. This also meant that the controller could no longer send the angle every half-second or so and update the angle within the car because adjusting the turning angle after motors have sped up would result in a much greater difference than if they are not.

A fix to this was to instead add the angle over time on the controller and send through the value on a less regular basis – this way by the time a new command is send the old turn was already finished and the new turn-commands on the motor had to deal with the spin-up time, and made angles more equivalent. Another measure was put in place by square rooting the turn-angle and multiplying it by a calibrated amount to make sure bigger turn angles didn’t translate into exponentially bigger turns as, again, the spin-up only affected small angles.

Another issue was the planned camera – the budget for this project ran out before one could be purchased and it was decided that it isn’t as important as all the other features.

### Design Choices – Motion-Sensitive Controller

The controller was the easier of two devices to design physically. I was surprised in how hard it was to find word resembling this project. I have finally found someone who had done something similar enough to learn from [2], though they did it using very different hardware. It included only three devices, two of which have already been designed as part of the RC Car. The device had to be placed on a very small board and, thankfully, the Arduino Nano chosen for this device had male pins and fitted right onto the breadboard. Below is a near design of what the final project would look like.

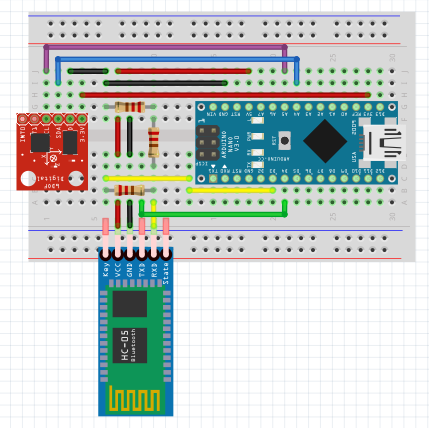


Figure 3: Motion-Sensitive Controller Device

The only difference between this and the final product is the IMU used – Fritzing, the modelling application used, did not have the IMU used in the project.

While wiring the device up was trivial, another unforeseen issue came into light when it came to coding: very few libraries exist for the selected IMU, and most that do haven’t been updated in such a long time they no longer work. The project came to a halt when building this device because of that. Thankfully, a library was eventually found [3] making the device was easy enough to program and put together.

Initially, the Yaw values were meant to be used on bow the controller and the RC car, but it was discovered that Yaw was the single inaccurate value on this IMU – turning the device 360 degrees barely moved the Yaw from -20 to 80 degrees which itself fluctuated more with pitch and roll than with yaw. This was clearly very problematic as it meant the devices could no longer each be fitted with their own IMU and be expected to simply send absolute degree values and align themselves. It meant that instead the controller would have to send a difference of angle or rotating time to the RC Car, which meant that sending too many values or values too large or too small would quickly misalign the devices. Unfortunately, there was no way to safely fix this so the focus of the project had to shift from aligning two devices to instead trying to calibrate how much information to send to keep the devices manually aligned.

This, of course, meant that near-perfect alignment could no longer be an aim for the project. The first alignment issue was something obvious that was somehow missed in the IPP and Interim Report – motors have spin-up time at which they aren’t spinning at maximum velocity. This means that the angle of a two-second turn would *not* be half as much as the angle of a four-second turn. A partial solution to this was put together and simple – square-rooting the angle before sending it off to the car would reduce the turning-time on large angles while increasing the turning-time on small angles.

Finally, no appropriate headset was available during development time, so using it in the final project had to be scrapped. It’s not the most important part of the project design-wise, seeing as the device can easily put on a headset at will, but it’s a noteworthy change as the project was initially inspired by Virtual Reality headsets.

### Design Choices – Mobile App

The Mobile App was initially meant to be the interactive core of the project – it would display the camera output and control the RC car alongside the controller. Near the end of the project, however, it became apparent that multiple Bluetooth devices could not be connected to a single Arduino Uno, that being besides the difficulty in acquiring a small camera module. When this was discovered, the mobile app development stopped because the app wouldn’t be used with the RC car instead of the controller.

A substantial portion of the app was developed however, with only the device selector being scrapped. The app was designed to be as simple and intuitive to use as possible. It’s meant to be used to calibrate and control the car, and making that complicated goes against the idea of a companion app. The app has four screens: Home, Calibrate, Select Device, and Control. The following paragraphs will go through them one by one and explain the workings behind them.

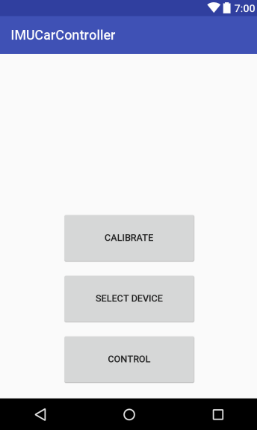


Figure 4: App “Home” Page

The first page that comes up is the Home screen. All that’s on it visually are the buttons to the three other pages, but as soon as it’s opened it automatically attempts to connect to the RC car Bluetooth device. It does so using the Vars singleton class (Appendix A – System Software). This makes it easy to send information to the Bluetooth device without having to re-connect on every screen.

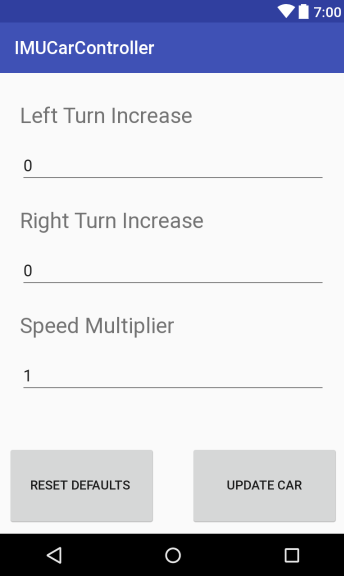


Figure 5: App “Calibrate” Page

The first of the three pages available on the Home page, Calibration allows the user to change how much the vehicle turns left and right, as well as the speed forward and back. These are additions and multipliers applied to the values sent from the controller to the RC car.

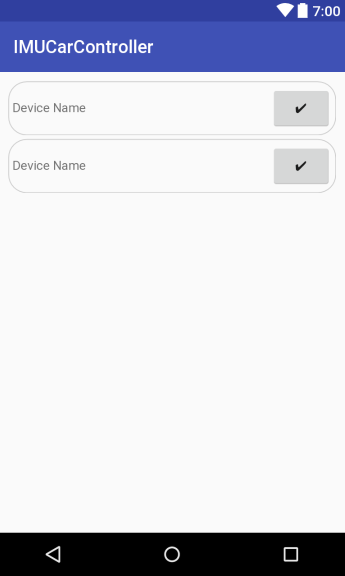


Figure 6: App "Select Device" Page

This is the only non-functional page in the final product. Initially, this page was meant to show available Bluetooth devices in the area and pressing the checkmark would pair the devices together.

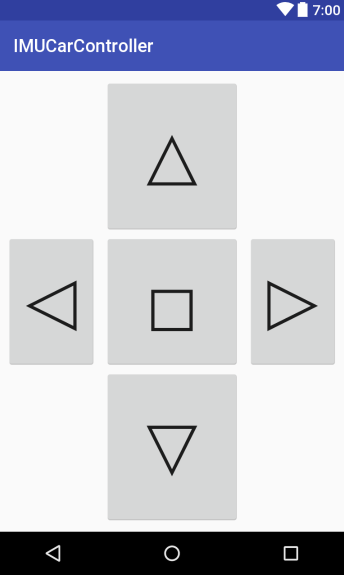


Figure 7: App "Control" Page

The Control screen allows the user to assume direct control of the RC car. Pressing the directional buttons makes the car move forward, back, or turn to the sides until another button is pressed. The stop button simply stops any action.

Lastly, there is the Vars singleton class. This class handles Bluetooth connections and data transfer. This was quite a challenge to create, and was adapted but strongly based on the code openly shared by user “Jon17” on the Arduino forums [4].

## Test Plan – RC Car

The RC Car has a very basic set of tests. It doesn’t have to truly process data – that’s the job of the mobile app and controller. Instead, the tests must ensure the controller does what it’s told. The tests are therefore composed of binary closed questions.

|  |  |  |
| --- | --- | --- |
| # | Test Name | Test Method |
| 1 | Controller Connection | The controller will send “controller connected” to the car. The car will print whatever it receives to the Serial Monitor. |
| 2 | Mobile App Connection | The mobile app will send “app connected” to the car. The car will print whatever it receives to the Serial Monitor. |
| 3 | Controller + App Connection | The controller will send “controller connected” and the mobile app will send “app connected” to the car. The car will print whatever it receives to the Serial Monitor. |
| 4 | Controller Left Turn | The controller will be turned left and the behaviour of the car will be observed. If it turns left a similar amount, it’ll be considered a pass. |
| 5 | Controller Right Turn | The controller will be turned right and the behaviour of the car will be observed. If it turns right a similar amount, it’ll be considered a pass. |
| 6 | Controller Drive Forward | The controller will be tilted forwards for two seconds then tilted back again and the behaviour of the car will be observed. If it drives forward for two seconds then stops, it’ll be considered a pass. |
| 7 | Controller Drive Backward | The controller will be tilted backwards for two seconds then tiled back again and the behaviour of the car will be observed. If the car drives backwards for two seconds then stops, it’ll be considered a pass. |
| 8 | Mobile App Left Turn | The “left” button will be held down for two seconds on the mobile app. If the car turns left for two seconds then stops, it’ll be considered a pass. |
| 9 | Mobile App Right Turn | The “right” button will be held down for two seconds on the mobile app. If the car turns right for two seconds then stops, it’ll be considered a pass. |
| 10 | Mobile App Drive Forward | The “up” button will be held down for two seconds on the mobile app. If the car drives forward for two seconds then stops, it’ll be considered a pass. |
| 11 | Mobile App Drive Backward | The “down” button will be held down for two seconds on the mobile app. If the car drives backward for two seconds then stops, it’ll be considered a pass. |
| 12 | Mobile App Calibration | The calibration values on the mobile phone will be increased by 2 between update, and updated three times. If the values in the RC Car change, it’ll be considered a pass. |

## Test Plan – Motion-Sensitive Controller

The controller isn’t connected to as many devices as the RC Car so it has less tests, but the tests are a lot better defined because of the varying type of data it handles and processes.

|  |  |  |
| --- | --- | --- |
| # | Test Name | Test Method |
| 1 | Slow Turn Left | The controller will be slowly turned left 180 degrees. The device will send messages to the Serial Monitor with the current angle. If the values are below 0, are consistent, and reset periodically it’ll be considered a pass. |
| 2 | Fast Turn Left | The controller will be quickly turned right 180 degrees. The device will send messages to the Serial Monitor with the current angle. If the values are below 0, are noticeably higher than with the “Slow Turn Left” test, and reset periodically it’ll be considered a pass. |
| 3 | Slow Turn Right | The controller will be slowly turned right 180 degrees. The device will send messages to the Serial Monitor with the current angle. If the values are above 0, are consistent, and reset periodically it’ll be considered a pass. |
| 4 | Fast Turn Right | The controller will be quickly turned right 180 degrees. The device will send messages to the Serial Monitor with the current angle. If the values are above 0, are noticeably higher than with the “Slow Turn Right” test, and reset periodically it’ll be considered a pass. |
| 5 | Tilt Forward | The controller will be tilted forwards for three seconds, then reset. The device will send messages to the Serial Monitor with either a “forward” or “back” message. If the device consistently prints “forward” while the device is tilted over a pre-determined angle then stops after three seconds, it’ll be considered a pass. |
| 6 | Tilt Backward | The controller will be tilted backwards for three seconds, then reset. The device will send messages to the Serial Monitor with either a “forward” or “back” message. If the device consistently prints “back” while the device is tilted over a pre-determined angle then stops after three seconds, it’ll be considered a pass. |

## Test Plan – Mobile Application

The mobile application was the hardest to write, but it has the simplest of tests – It was developed used the Android Studio sdk so most things like buttons are guaranteed to work correctly. Instead, the calibration and detection/connection must be tested as these are the only self-written parts.

|  |  |  |
| --- | --- | --- |
| # | Test Name | Test Method |
| 1 | Device List Printing | The “Select Device” button will be tapped. If it correctly loads a list of available devices, it’ll be considered a pass. |
| 2 | Device Selection | The checkmark next to a device from the “Select Device” list will be tapped. If it successfully connects, it’ll be considered a pass. |
| 3 | Defaults | After the test in “RC Car – Test 12” is conducted, the “Reset Defaults” button will be pressed in the “Calibrate” menu. If the calibration values reset to “0,” “0,” and “1,” it’ll be considered a pass. |

## Test Results

Most of the results proved successful (besides some of the mobile app tests created for unfinished features). There was an unexpected testing failure however. It turned out that Arduino Uno has huge problems when it comes to listening to two Bluetooth transceivers on one microcontroller. This was a huge hit to the project as it means the project could no longer run by connecting both a mobile app and motion-controller to an RC motor. Thankfully, the testing of the hardware came about before the mobile app was finished so time could be saved on that.

Nevertheless, this limited the project to either having to change the code between using the mobile app and controller (thus invalidating any changes the mobile app pushes to the calibration), or showing the mobile app on the side but scrapping it as part of the larger project. The latter was deemed the more responsible decision.

An interesting point to bring up now – this could very well have been spared had I decided to go with WiFi over Bluetooth. Had this been known, WiFi would have been chosen as the RC Car node. It would have been harder to write code for, but that’s a minor downside when compared to the upside of having it work at all.

The full test results can be seen in “Appendix C – Detailed Test Results.”

## Conclusions and Future Work

Looking over the outcome, it has changed a lot from the original work though this was mainly due to unforeseen technological issues with no real workarounds for this implementation. If this project were to continue into future iterations, there are a few changes that would have to be made:

1. Helmet-Mounting – I didn’t have any helmets to use, but in future the controller would be mounted on a helmet. This would be an easy implementation, and the starting tilt and minimum angle would be calibrated in the first few seconds after being turned on.
2. Camera – The funds were already very high for this project so the idea of using a camera had to be scrapped as it was a simple (though semi-expensive) streaming service from the controller. In a future release, a camera would be mounted on the front of the RC car so it could stream to the mobile app. Speaking of which,
3. WiFi – The Bluetooth modules on the devices would be replaced by WiFi modules to allow both the controller *and* the app to connect.
4. Dual-Camera VR – If the price of Virtual Reality comes down, it should be possible to install two calibratable cameras on the RC car, and stream those directly to each eye of the VR headset. This would create an incredible simulation of Virtual Reality in real life – one could look up at themselves and see themselves in full 3D.

## Self-Evaluation

The project is clearly missing a few pieces, but I would consider the project as a whole a success because even though not all the aims were achieved, the reason I set out to create this was to test the use of motion controls in the direct control of a vehicle – this was proven possible and, surprisingly, rather easy.

If I were to start the project again from scratch, I’d have gone with WiFi from the very start which would save most of the issues with the project. Alongside that change, I would borrow the Arduino Uno from university so I could afford a mini camera.

The research I’ve done was thorough so I wouldn’t change how I went about that – except perhaps, again, looking more into Bluetooth issues.

Thanks to the hands-on approach to the coursework and emphasis on personal research, I feel this project will help me in many future projects as I feel more prepared do this again.

# Bibliography

|  |  |
| --- | --- |
| [1] | B. Davison, “Napier,” 23 February 2017. [Online]. Available: http://www.soc.napier.ac.uk/~40000812/Modules/SET09118/Practicals/w7\_motors.html. |
| [2] | “PID Control Arduino Drones MPU 6050 MPU 9250 Gyro Accelerator,” Electronoobs, [Online]. Available: http://www.electronoobs.com/eng\_robotica\_tut6.php. |
| [3] | Brent, “MPU-9250 Hookup Guide,” SparkFun Electronics, [Online]. Available: https://learn.sparkfun.com/tutorials/mpu-9250-hookup-guide. |
| [4] | Jon17, “Arduino Forums,” 1 April 2013. [Online]. Available: https://forum.arduino.cc/index.php?topic=157621.0. |

# Appendix A – System Software

This section contains all the source code of the project, besides the XML code from Android Studio as it’s got very little to do with the logic of the program.

This document will be sent over via the internet, so the source code has been shrunk so as not to take as much space. The first two files are the Arduino IMU controller device and RC Car files respectively. Following them is the Android Studio Java code.

|  |
| --- |
| Sketch\_imu.ino |
| #include <SoftwareSerial.h>  #include "quaternionFilters.h"  #include "MPU9250.h"  #define AHRS true  #define SerialDebug true  char c = ' ';  const int PhotoCell = A0;  String current = "";  const int pinRed = 4;  bool onRed = false;  double oldYaw = 0;  int counter = 0;  MPU9250 myIMU;  SoftwareSerial BTserial(2, 3);  void setup() {  Wire.begin();  Serial.begin(9600);  BTserial.begin(38400);  pinMode(pinRed, OUTPUT);  byte c = myIMU.readByte(MPU9250\_ADDRESS, WHO\_AM\_I\_MPU9250);  Serial.print("MPU9250 "); Serial.print("I AM "); Serial.print(c, HEX);  Serial.print(" I should be "); Serial.println(0x71, HEX);  if (c == 0x71) {  Serial.println("MPU9250 is online...");  myIMU.MPU9250SelfTest(myIMU.selfTest);  Serial.print("x-axis self test: acceleration trim within : ");  Serial.print(myIMU.selfTest[0],1); Serial.println("% of factory value");  Serial.print("y-axis self test: acceleration trim within : ");  Serial.print(myIMU.selfTest[1],1); Serial.println("% of factory value");  Serial.print("z-axis self test: acceleration trim within : ");  Serial.print(myIMU.selfTest[2],1); Serial.println("% of factory value");  Serial.print("x-axis self test: gyration trim within : ");  Serial.print(myIMU.selfTest[3],1); Serial.println("% of factory value");  Serial.print("y-axis self test: gyration trim within : ");  Serial.print(myIMU.selfTest[4],1); Serial.println("% of factory value");  Serial.print("z-axis self test: gyration trim within : ");  Serial.print(myIMU.selfTest[5],1); Serial.println("% of factory value");  float gyroBias[3] = {0, 0, 1.0f};  myIMU.calibrateMPU9250(gyroBias, myIMU.accelBias);  myIMU.initMPU9250();  Serial.println("MPU9250 initialized for active data mode....");  byte d = myIMU.readByte(AK8963\_ADDRESS, WHO\_AM\_I\_AK8963);  Serial.print("AK8963 "); Serial.print("I AM "); Serial.print(d, HEX);  Serial.print(" I should be "); Serial.println(0x48, HEX);  myIMU.initAK8963(myIMU.factoryMagCalibration);  Serial.println("AK8963 initialized for active data mode....");  if (SerialDebug) {  Serial.print("X-Axis sensitivity adjustment value ");  Serial.println(myIMU.factoryMagCalibration[0], 2);  Serial.print("Y-Axis sensitivity adjustment value ");  Serial.println(myIMU.factoryMagCalibration[1], 2);  Serial.print("Z-Axis sensitivity adjustment value ");  Serial.println(myIMU.factoryMagCalibration[2], 2);  }  }  else {  Serial.print("Could not connect to MPU9250: 0x");  Serial.println(c, HEX);  while(1);  }  }  void loop() {  if (myIMU.readByte(MPU9250\_ADDRESS, INT\_STATUS) & 0x01) {  myIMU.readAccelData(myIMU.accelCount);  myIMU.getAres();  myIMU.ax = (float)myIMU.accelCount[0]\*myIMU.aRes; // - accelBias[0];  myIMU.ay = (float)myIMU.accelCount[1]\*myIMU.aRes; // - accelBias[1];  myIMU.az = (float)myIMU.accelCount[2]\*myIMU.aRes; // - accelBias[2];  myIMU.readGyroData(myIMU.gyroCount);  myIMU.getGres();  myIMU.gx = (float)myIMU.gyroCount[0]\*myIMU.gRes;  myIMU.gy = (float)myIMU.gyroCount[1]\*myIMU.gRes;  myIMU.gz = (float)myIMU.gyroCount[2]\*myIMU.gRes;  myIMU.readMagData(myIMU.magCount);  myIMU.getMres();  // automatically calculated  myIMU.magBias[0] = +470.;  myIMU.magBias[1] = +120.;  myIMU.magBias[2] = +125.;  myIMU.mx = (float)myIMU.magCount[0]\*myIMU.mRes\*myIMU.factoryMagCalibration[0] -  myIMU.magBias[0];  myIMU.my = (float)myIMU.magCount[1]\*myIMU.mRes\*myIMU.factoryMagCalibration[1] -  myIMU.magBias[1];  myIMU.mz = (float)myIMU.magCount[2]\*myIMU.mRes\*myIMU.factoryMagCalibration[2] -  myIMU.magBias[2];  }    myIMU.updateTime();  MahonyQuaternionUpdate(myIMU.ax, myIMU.ay, myIMU.az, myIMU.gx\*DEG\_TO\_RAD,  myIMU.gy\*DEG\_TO\_RAD, myIMU.gz\*DEG\_TO\_RAD, myIMU.my,  myIMU.mx, myIMU.mz, myIMU.deltat);  if (!AHRS)  {  myIMU.delt\_t = millis() - myIMU.count;  if (myIMU.delt\_t > 50) {  myIMU.count = millis();  } // if (myIMU.delt\_t > 500)  } // if (!AHRS)  else  {  // Serial print and/or display at 0.5 s rate independent of data rates  myIMU.delt\_t = millis() - myIMU.count;  // update LCD once per half-second independent of read rate  if (myIMU.delt\_t > 50) {    if (fabs(myIMU.gz) > 1.0) {  oldYaw += myIMU.gz + 0.14;  }  //Serial.print("Pitch: ");  //Serial.println(myIMU.yaw);  if (counter == 3 && fabs(oldYaw) > 1.0) {  myIMU.pitch = -asin(2.0f \* (\*(getQ()+1) \* \*(getQ()+3) - \*getQ() \*  \*(getQ()+2)));  myIMU.pitch \*= RAD\_TO\_DEG;  //Serial.print("Pitch: ");  //Serial.println(myIMU.pitch);    String str = String((long)(sqrt(fabs(oldYaw))) \* 7.5);  //Serial.print(" gz = "); Serial.println( oldYaw, 2);  if (oldYaw > 0.0) {  BTserial.write("r\n");  }  else {  BTserial.write("l\n");  }    for (int i = 0; i < str.length(); i++) {  BTserial.write(str.charAt(i));  }  BTserial.write("\n");  if (myIMU.pitch > 25.0) {  BTserial.write("f\n");  }  else if (myIMU.pitch < -10.0) {  BTserial.write("b\n");  }  else {  BTserial.write("s\n");  }  Serial.print(" Change = "); Serial.println( oldYaw, 2);  //Serial.println(" Change2 = " + str);    oldYaw = 0;  }  counter++;  if (counter > 5)  counter = 0;    myIMU.count = millis();  myIMU.sumCount = 0;  myIMU.sum = 0;  } // if (myIMU.delt\_t > 500)  } // if (AHRS)    if (BTserial.available())  {  c = BTserial.read();  Serial.println(c);    if (c == ' ' || c == '\n') {  current = "";  }  else {  current += c;  }    if (current == "red") {  if (onRed) {  digitalWrite(pinRed, LOW);  }  else {  digitalWrite(pinRed, HIGH);  }  onRed = !onRed;  }  }    // Keep reading from Arduino Serial Monitor and send to HC-05  if (Serial.available())  {  c = Serial.read();  Serial.println(c);    BTserial.write(c);  }  } |
| Sketch\_car.ino |
| #include <SoftwareSerial.h>  SoftwareSerial BTserial(2, 3); // RX | TX  SoftwareSerial BTphone(11, 10); // RX | TX  int motor\_left[] = {6, 7};  int motor\_right[] = {8, 9};  int led = 13;  long leftTurn = 0;  long rightTurn = 0;  long timeStart;  long timeLast;  int spd = 1;  int right\_spd = 0;  int left\_spd = 0;  char c = ' ';  String current = "";  bool onRed = false;  char expectingNext;  #define N\_NONE 0  #define N\_LEFT 1  #define N\_RIGHT 2    void setup() {  int i;  for(i = 0; i < 2; i++){  pinMode(motor\_left[i], OUTPUT);  pinMode(motor\_right[i], OUTPUT);  pinMode(led, OUTPUT);  }    Serial.begin(9600);  Serial.println("Arduino is ready");  Serial.println("Remember to select Both NL & CR in the serial monitor");    // HC-05 default serial speed for AT mode is 9600  //BTphone.begin(38400);  BTserial.begin(38400);  timeStart = millis();  }    void loop() {  // If the car is meant to be turning, turn  if (leftTurn > 0 || rightTurn > 0) {  Serial.println("Started Turn");    // If both rightTurn and leftTurn have values in them, take away the difference  // and use the larger one  if (rightTurn > leftTurn) {  rightTurn -= leftTurn;  }  else if (leftTurn > rightTurn) {  leftTurn -= rightTurn;  }  if (leftTurn > 0) {  Serial.println("Left Turn");  leftTurn -= millis() - timeLast;    digitalWrite(motor\_left[0], LOW);  digitalWrite(motor\_left[1], HIGH);    digitalWrite(motor\_right[0], HIGH);  digitalWrite(motor\_right[1], LOW);    if (leftTurn <= 0) {  leftTurn = 0;    digitalWrite(motor\_left[0], LOW);  digitalWrite(motor\_left[1], LOW);  digitalWrite(motor\_right[0], LOW);  digitalWrite(motor\_right[1], LOW);  }  }  else if (rightTurn > 0) {  Serial.println("Right Turn");  rightTurn -= millis() - timeLast;  digitalWrite(motor\_left[0], HIGH);  digitalWrite(motor\_left[1], LOW);    digitalWrite(motor\_right[0], LOW);  digitalWrite(motor\_right[1], HIGH);  if (rightTurn <= 0) {  rightTurn = 0;    digitalWrite(motor\_left[0], LOW);  digitalWrite(motor\_left[1], LOW);  digitalWrite(motor\_right[0], LOW);  digitalWrite(motor\_right[1], LOW);  }  }  }  timeLast = millis();  while (BTserial.available()) {  c = BTserial.read();  Serial.println(c);    if (c == ' ' || c == '\n') {  Serial.println("Done read");  Serial.println("current: " + current);  if (expectingNext == N\_NONE) {  if (current == "red") {  if (onRed) {  digitalWrite(led, LOW);  }  else {  digitalWrite(led, HIGH);  }    onRed = !onRed;  }  else if (current == "l") {  expectingNext = N\_LEFT;  }  else if (current == "r") {  expectingNext = N\_RIGHT;  }  else if (current == "blue") {  drive\_forward(1000);  motor\_stop(25);  }  else if (current == "f") {  digitalWrite(motor\_left[0], HIGH);  digitalWrite(motor\_left[1], LOW);    digitalWrite(motor\_right[0], HIGH);  digitalWrite(motor\_right[1], LOW);  }  else if (current == "b") {  digitalWrite(motor\_left[0], LOW);  digitalWrite(motor\_left[1], HIGH);    digitalWrite(motor\_right[0], LOW);  digitalWrite(motor\_right[1], HIGH);  }  else if (current == "s") {  digitalWrite(motor\_left[0], LOW);  digitalWrite(motor\_left[1], LOW);    digitalWrite(motor\_right[0], LOW);  digitalWrite(motor\_right[1], LOW);  }  }  // If it's a left/right degree message  else {  if (expectingNext == N\_LEFT) {  leftTurn += current.toInt() + left\_spd;  Serial.println("Ldist: " + String(leftTurn));  expectingNext = N\_NONE;  }  else if (expectingNext == N\_RIGHT) {  rightTurn += current.toInt() + right\_spd;  Serial.println("Rdist: " + String(rightTurn));  expectingNext = N\_NONE;  }  }    current = "";  }  else {  current += c;  }  }    /\*  // Disabled because only one BT device can be connected  // This handles the phone input  if (BTphone.available()) {  c = BTphone.read();    if (c == 'f') {  digitalWrite(motor\_left[0], HIGH);  digitalWrite(motor\_left[1], LOW);    digitalWrite(motor\_right[0], HIGH);  digitalWrite(motor\_right[1], LOW);  }  else if (c == 'b') {  digitalWrite(motor\_left[0], LOW);  digitalWrite(motor\_left[1], HIGH);    digitalWrite(motor\_right[0], LOW);  digitalWrite(motor\_right[1], HIGH);  }  else if (c == 'l') {  digitalWrite(motor\_left[0], LOW);  digitalWrite(motor\_left[1], HIGH);    digitalWrite(motor\_right[0], HIGH);  digitalWrite(motor\_right[1], LOW);  }  else if (c == 'r') {  digitalWrite(motor\_left[0], HIGH);  digitalWrite(motor\_left[1], LOW);    digitalWrite(motor\_right[0], LOW);  digitalWrite(motor\_right[1], HIGH);  }  else if (c == 's') {  digitalWrite(motor\_left[0], LOW);  digitalWrite(motor\_left[1], LOW);    digitalWrite(motor\_right[0], LOW);  digitalWrite(motor\_right[1], LOW);  }  }    \*/  // Keep reading from Arduino Serial Monitor and send to HC-05  if (Serial.available())  {  c = Serial.read();  Serial.println(c);    }  } |
| Home.java |
| **package** com.mktg.imucarcontroller;  **import** android.bluetooth.BluetoothAdapter; **import** android.bluetooth.BluetoothDevice; **import** android.content.Intent; **import** android.support.v7.app.AppCompatActivity; **import** android.os.Bundle; **import** android.view.View; **import** android.widget.Button; **import** android.widget.Toast;  **import** java.io.IOException; **import** java.util.UUID;  **public class** Home **extends** AppCompatActivity {   **private** Vars **vars**;    @Override  **protected void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.***activity\_home***);   **vars** = Vars.*getInstance*();   Button btn\_calibrate = (Button)findViewById(R.id.***btn\_calibrate***);  Button btn\_select\_device = (Button)findViewById(R.id.***btn\_select\_device***);  Button btn\_control = (Button)findViewById(R.id.***btn\_control***);   btn\_calibrate.setOnClickListener(**new** View.OnClickListener() {  @Override  **public void** onClick(View v) {  startActivity(**new** Intent(Home.**this**, Calibrate.**class**));  }  });   btn\_select\_device.setOnClickListener(**new** View.OnClickListener() {  @Override  **public void** onClick(View v) {  startActivity(**new** Intent(Home.**this**, SelectDevice.**class**));  }  });   btn\_control.setOnClickListener(**new** View.OnClickListener() {  @Override  **public void** onClick(View v) {  startActivity(**new** Intent(Home.**this**, Control.**class**));  }  });   CheckBt();  Connect();   Vars.*getInstance*().*BTThread*();  }   **private void** CheckBt() {  Vars.*getInstance*().*mBluetoothAdapter* = BluetoothAdapter.*getDefaultAdapter*();   **if** (!Vars.*getInstance*().*mBluetoothAdapter*.isEnabled()) {  Toast.*makeText*(getApplicationContext(), **"Bluetooth Not Enabled!"**, Toast.***LENGTH\_SHORT***).show();  }   **if** (Vars.*getInstance*().*mBluetoothAdapter* == **null**) {  Toast.*makeText*(getApplicationContext(), **"Bluetooth Not Found!"**, Toast.***LENGTH\_SHORT***) .show();  }  }   **public void** Connect() {  BluetoothDevice device = Vars.*getInstance*().*mBluetoothAdapter*.getRemoteDevice(**"00:21:13:00:61:E8"**);  Vars.*getInstance*().*mBluetoothAdapter*.cancelDiscovery();  **vars**.**btSocket** = **null**;  **try** {  **vars**.**btSocket** = device.createRfcommSocketToServiceRecord(UUID.*fromString*(**"00001101-0000-1000-8000-00805F9B34FB"**));  **vars**.**btSocket**.connect();  } **catch** (IOException e) {  **try** {  **vars**.**btSocket**.close();  } **catch** (IOException ee) {   }  }  } } |
| Calibrate.java |
| **package** com.mktg.imucarcontroller;  **import** android.app.Activity; **import** android.os.Bundle; **import** android.view.View; **import** android.widget.Button; **import** android.widget.EditText;  **import** com.mktg.imucarcontroller.R;  **public class** Calibrate **extends** Activity {   @Override  **protected void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.***activity\_calibrate***);   Button btn\_update = (Button)findViewById(R.id.***btn\_update***);   btn\_update.setOnClickListener(**new** View.OnClickListener() {  @Override  **public void** onClick(View v) {  Vars.*getInstance*().**sendSpeed** = **"q"** + ((EditText)findViewById(R.id.***edit\_speed***)).getText().toString();  Vars.*getInstance*().**sendLeft** = **"w"** + ((EditText)findViewById(R.id.***edit\_left***)).getText().toString();  Vars.*getInstance*().**sendRight** = **"e"** + ((EditText)findViewById(R.id.***edit\_right***)).getText().toString();  }  });  } } |
| SelectDevice.java |
| **package** com.mktg.imucarcontroller;  **import** android.support.v7.app.AppCompatActivity; **import** android.os.Bundle;  **public class** SelectDevice **extends** AppCompatActivity {   @Override  **protected void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.***activity\_select\_device***);  } } |
| Control.java |
| **package** com.mktg.imucarcontroller;  **import** android.app.Activity; **import** android.bluetooth.BluetoothAdapter; **import** android.bluetooth.BluetoothDevice; **import** android.bluetooth.BluetoothSocket; **import** android.content.Intent; **import** android.os.Bundle; **import** android.util.Log; **import** android.view.View; **import** android.widget.Button; **import** android.widget.Toast;  **import** java.io.IOException; **import** java.io.InputStream; **import** java.net.Socket; **import** java.util.UUID;  **public class** Control **extends** Activity {   Vars **vars**;  @Override  **protected void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.***activity\_control***);   **vars** = Vars.*getInstance*();   Button btn\_forward = (Button)findViewById(R.id.***btn\_forward***);  Button btn\_back = (Button)findViewById(R.id.***btn\_down***);  Button btn\_left = (Button)findViewById(R.id.***btn\_left***);  Button btn\_right = (Button)findViewById(R.id.***btn\_right***);  Button btn\_stop = (Button)findViewById(R.id.***btn\_stop***);   btn\_forward.setOnClickListener(**new** View.OnClickListener() {  @Override  **public void** onClick(View v) {  Vars.*getInstance*().**sendCode** = **"f"**;  }  });   btn\_back.setOnClickListener(**new** View.OnClickListener() {  @Override  **public void** onClick(View v) {  Vars.*getInstance*().**sendCode** = **"b"**;  }  });   btn\_left.setOnClickListener(**new** View.OnClickListener() {  @Override  **public void** onClick(View v) {  Vars.*getInstance*().**sendCode** = **"l"**;  }  });   btn\_right.setOnClickListener(**new** View.OnClickListener() {  @Override  **public void** onClick(View v) {  Vars.*getInstance*().**sendCode** = **"r"**;  }  });   btn\_stop.setOnClickListener(**new** View.OnClickListener() {  @Override  **public void** onClick(View v) {  Vars.*getInstance*().**sendCode** = **"s"**;  }  });  } } |
| Vars.java |
| **package** com.mktg.imucarcontroller;  **import** android.bluetooth.BluetoothAdapter; **import** android.bluetooth.BluetoothSocket; **import** android.util.Log;  **import** java.io.IOException; **import** java.io.OutputStream;  */\*\*  \* Created by marce on 06/04/2017.  \*/* **public class** Vars {  **private static** Vars *instance*;  **public** String **sendCode** = **""**;  **public** String **sendSpeed** = **""**;  **public** String **sendLeft** = **""**;  **public** String **sendRight** = **""**;  **public** OutputStream **outStream**;  **public** BluetoothSocket **btSocket**;  **public boolean stop** = **false**;  **public static** BluetoothAdapter *mBluetoothAdapter*;   **public static** Vars getInstance() {  **if** (*instance* == **null**) {  *instance* = **new** Vars();  }   **return** *instance*;  }   **public static void** BTThread() {  Thread thread = **new** Thread(**new** Runnable() {   @Override  **public void** run() {  **while** (!*instance*.**stop**) {  **if** (*instance*.**sendCode** != **""**) {  **try** {  *instance*.**outStream** = *instance*.**btSocket**.getOutputStream();  } **catch** (IOException e) {  Log.*d*(**"KaiDroid"**, **"Failed to send."**, e);  }   String message = *instance*.**sendCode**;  **byte**[] msgBuffer = message.getBytes();   **try** {  *instance*.**outStream**.write(msgBuffer);   message = *instance*.**sendSpeed**;  msgBuffer = message.getBytes();  *instance*.**outStream**.write(msgBuffer);   message = *instance*.**sendLeft**;  msgBuffer = message.getBytes();  *instance*.**outStream**.write(msgBuffer);   message = *instance*.**sendRight**;  msgBuffer = message.getBytes();  *instance*.**outStream**.write(msgBuffer);  Log.*d*(**"KaiDroid"**, **"DATA SENT!!!!."**);  } **catch** (IOException e) {   }   *instance*.**sendCode** = **""**;  }  }  }  });   thread.start();  } } |

# Appendix B – System hardware

The mini breadboard used in the controller, the car chassis and the motors used in the project were all borrowed from the school and returned prior to writing this report. As such, only an old video of the car without Bluetooth connections is available:

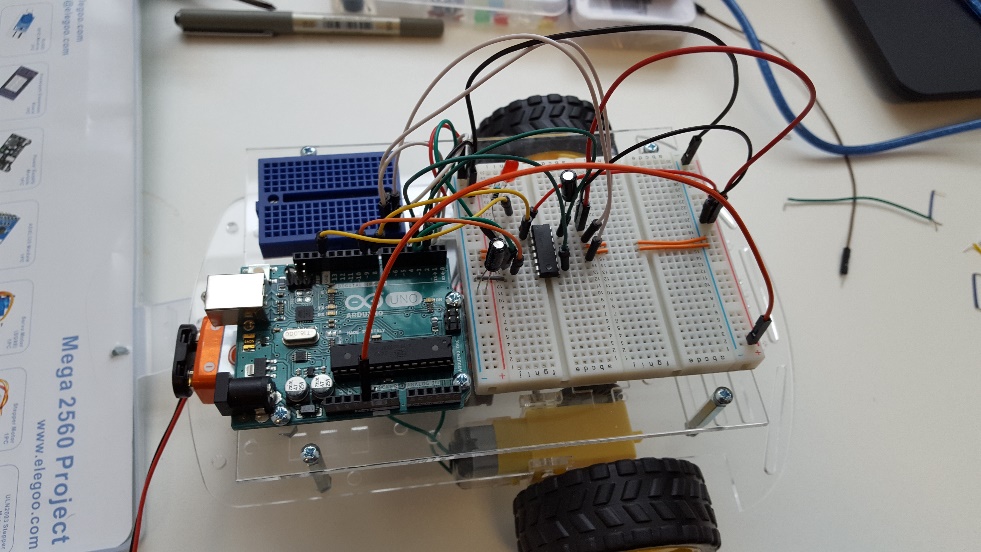


Figure 8: Interim Car Build

|  |  |
| --- | --- |
| **Device** | **Details** |
| SparkFun IMU Breakout - MPU-9250 | What: 9 Degrees of Freedom IMU.  Use: Gyroscope  Gyroscope Operating Current: 3.2mA  Accelerometer Operating Current: 450µA  Magnetometer Operating Current: 280µA  VCC Range: 2.4V – 3.6V  Price: $14.95 |
| Resistors | What: Resistors used in diagrams  Resistance: 1kΩ |
| WINGONEER HC-05 Wireless Bluetooth Transceiver | What: Bluetooth Transceiver Module  Use: Connecting devices in project  VCC: 3.6V – 6V  Operating Current: 10mA - 30mA  Stable Connection Distance: 10m  Rx Voltage: 3.3V  Price: £6.49 |
| Arduino Uno | What: Programmable device for RC Car  Microcontroller: ATMEGA 328  Clock Speed: 16MHz  RAM: 32 KB  Input Voltage: 7V – 12V  Operating Voltage: 5V  Price: £17.30 |
| Arduino Nano | What: Programmable device for motion controller  Microcontroller: ATMEGA 328  Clock Speed: 16MHz  RAM: 32KB  Input Voltage: 7V – 12V  Operating Voltage: 5V  Price: £21.60 |

# Appendix C – Detailed test results

## RC Car

|  |  |  |
| --- | --- | --- |
| # | Test Name | Test Outcome |
| 1 | Controller Connection | Success. |
| 2 | Mobile App Connection | Success. |
| 3 | Controller + App Connection | Failure – using more than one Bluetooth device disables both devices on an Arduino Uno, and the power draw is too high on an Arduino Mega. The device will have to be used in two parts instead of being used as a triple-device project. |
| 4 | Controller Left Turn | Success. |
| 5 | Controller Right Turn | Success. |
| 6 | Controller Drive Forward | Success. |
| 7 | Controller Drive Backward | Success. |
| 8 | Mobile App Left Turn | Success. |
| 9 | Mobile App Right Turn | Success. |
| 10 | Mobile App Drive Forward | Success. |
| 11 | Mobile App Drive Backward | Success. |
| 12 | Mobile App Calibration | Success. |

## Motion-Sensitive Controller

|  |  |  |
| --- | --- | --- |
| # | Test Name | Test Outcome |
| 1 | Slow Turn Left | Success. |
| 2 | Fast Turn Left | Success. |
| 3 | Slow Turn Right | Success. |
| 4 | Fast Turn Right | Success. |
| 5 | Tilt Forward | Success. |
| 6 | Tilt Backward | Success. |

## Mobile Application

|  |  |  |
| --- | --- | --- |
| # | Test Name | Test Outcome |
| 1 | Device List Printing | Failed – device listing was scrapped. |
| 2 | Device Selection | Failed – device listing was scrapped. |
| 3 | Defaults | Success. |

# Appendix D – Initial Project Proposal

# **Arduino Head-Mounted Drone Controller**

**Initial Project Proposal**

Name: Marcel Troscianko

Matric No: 40173086

Abstract

The recent rise in VR and motion controls has yielder results on in the virtual world – this project aims to bring those benefits to real-world controllable drones. A simple version of this will be done by creating a head-mounted sensing system to control a drone, a basic drone with wheels[1] and a movable camera, and a mobile application to help with the control of the drone as well as displaying its output (something that would be replaced by a VR headset in future iterations).

The devices will be linked via Bluetooth and the processing will be done either on the drone or the headset – the mobile app will simply change calibration and display pre-calculated results. The mobile app will be very simple in user interface unless the hardware proves easier to create than anticipated.

The deliverables from the project will be a drone, a sensing and controller headset, and a companion app for calibration.

**Declaration of authorship: I confirm that the work submitted is mine and that wherever possible the work of others has been clearly acknowledged and referenced.**

**Signature : Marcel Troscianko Date: 15/02/2017**

## Introduction

### Background/context

Multiple games have recently come out with controllable drones (Watch\_Dogs 2 and Rainbow Six: Siege to name a couple), and with the recent rise of Virtual Reality and motion controls the concept of a head-mounted drone controller springs to mind.

The basic idea is to create a helmet or goggle mounted sensing system that detects the user’s movements and translates those into commands for a little drone on wheels holding a camera, such that the camera moves in-sync with the user’s head. The basic prototype will relay the video from the camera to the mobile app, while the full product would include two cameras linked to a VR headset so that the user can see what the drone sees along with full 3D perspective.

### Aims and deliverables

* Controllable wheeled drone with cameras
* Headset with sensors for drone control
* Mobile app for drone control and headset calibration

Design choices

The project will consist of three main parts:

* The drone will be the simplest deliverable to create – it’ll be a simple drone with a mounted camera. The drone should be remote controllable, and will be able to move forward/backward, turn left/right, and tilt the camera up/down.
* The mobile application will be the centre of control for the project – it’ll consist of the camera output, a set of calibration options, and a simple layout of up-down-left-right. The user interface will be very simple seeing as the time spent on this project is very limited and there may be problems in properly implementing the two other deliverables.
* The head-mounted sensing system will consist of multiple devices measuring the user’s movements. It is uncertain as of yet whether the main processing will happen here or on the drone itself, but both have their pros and cons – the processing should be done before any information is transmitted, but having a clunky unit on the user’s head is very counter-intuitive. The head piece itself could also come in two different forms – a helmet would be easier to use, but mounting it on a set of goggles would definitely be more comfortable for the user. The final goal is to use the built-in features of a VR headset and output directly to its screens, but that will most likely prove too advanced for this course. Using gyroscopes and accelerometers with Arduino has been done many times before so it should be easy enough to implement [2].

The information will most likely be transmitted via Bluetooth – the maximum range is approximately 100 metres which is enough for a remote-controlled vehicle.

Method

The project will be put together using two separate Arduino boards communicating with each other using a slave-master relationship [3]. While the drone can be put together quite loosely, the head-piece will mostly likely be welded together so as not to be too heavy or clunky to wear. The mobile app will be written in Android Studio while the Arduino software will be written using the Arduino development environment. The specific hardware components are yet to be selected so no diagram can be legitimately drawn up, but they must be the smallest components available to avoid building a heavy kit. The components will be bought online so freedom of choice is big.

Below is a diagram representing the timeline of the project development.



Figure 9: IPP Timeline

The first prototype will be just a phone-controlled drone as well as a headset-controlled sphere in a 3d environment to calibrate the headset. Once the headset is calibrated well, the next prototype will be a proper headset-controlled drone.

A major limitation is my lack of experience with wireless transmission – the information is very likely to either be sent more slowly than the headset records it or with information being long, so the car may go more and more out of calibration with time. A way to counter that would be to make up for that on the side of the headset, but I’m far too inexperienced with wireless technology to know whether this would fix the issue for sure, and it would make the headset far heavier than need be.

Another issue is the accuracy of freely available and inexpensive accelerometers and gyroscopes.

Finally, mounting the equipment on a headset comfortably could prove too time-consuming to be feasible.

References

* [1] Simple RC car for beginners Tutorial - <http://www.instructables.com/id/Simple-RC-car-for-beginners-Android-control-over-/>
* [2] Arduino 5 Minute Tutorials: Lesson 7 – Accelerometers, Gyros, IMUs - <http://www.robotshop.com/blog/en/arduino-5-minute-tutorials-lesson-7-accelerometers-gyros-imus-3634> (May 2, 2012)
* [3] RC Hobby Controllers and Arduino - <https://www.sparkfun.com/tutorials/348> (May 22, 2012)

# Appendix E – Interim report

# Title: Head-Controlled Arduino Vehicle

**Interim report**

Name: Marcel Troscianko

Matric No: 40173086

Introduction

The idea is inspired by the recent rise of Virtual Reality and motion controllers – a head-mounted toy car controller. It would be a helmet or goggle mounted sensing system that detects the user’s movements and translates those into commands for a little car with a camera, such that the camera moves in-sync with the user’s head. The basic prototype will relay the video from the camera to the mobile app, while the full product would include two cameras linked to a VR headset so that the user can see what the drone sees along with full 3D perspective.

### Deliverables

* Controllable wheeled vehicle (little toy)
* Headset with sensors
* Mobile app

The report won’t go extremely in-depth as I already had to cut out a couple of pages worth of basic information to keep within the three-page limit.

Current status

Having been waiting for payment to buy an Arduino kit for use at home and dealing with a low group size issues for our Group Project I haven’t been able to create a physical device just yet but I have been able to create a Bluetooth-enabled mobile app in Android Studio to test and eventually calibrate the android devices. I have also been able to select devices online for purchase when my money comes in. The hardware is listed below:

|  |  |  |
| --- | --- | --- |
| IMU | MPU-9250 | I decided I’ll need to go for both an accelerometer and a gyroscope to control the device. I’ll need two of them (one for headset, one for vehicle) and a magnetometer could come in handy if it turns out the accelerometer and gyroscope calibrations drift over time. I went for a breakout one because one of them must be mounted on the headset which must be as small as possible. |
| Motors | <Undetermined> | I haven’t yet decided on the motors to use, but they’ll need to be 3V-9V |
| Vehicle Controller | Arduino Uno | It’s the type we’ve used at university and therefore the one I’m most comfortable using |
| Headset Controller | Arduino Micro | This is the easiest to use micro-sized Arduino and has a near-identical pin layout to the Uno |
| Bluetooth Device\* | Bluetooth HC-05 | This is a commonly used Bluetooth device so it has multiple online tutorials.  \*NOTE: May change to WiFi module if distance is too short. |
| H-Bridge | H-Bridge Motor Driver 1A - SN754410 | Recommended on tutorials I’ve looked into |

These will be used to create devices shown in the diagrams below.

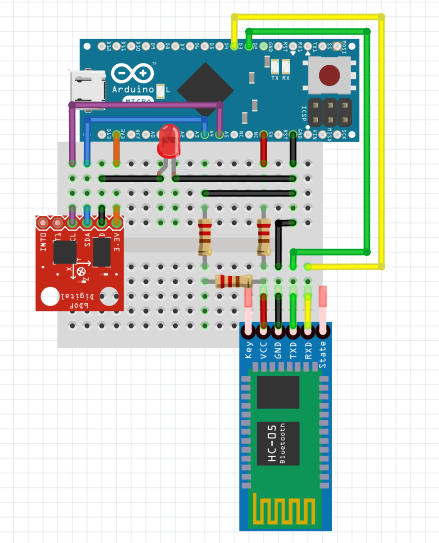
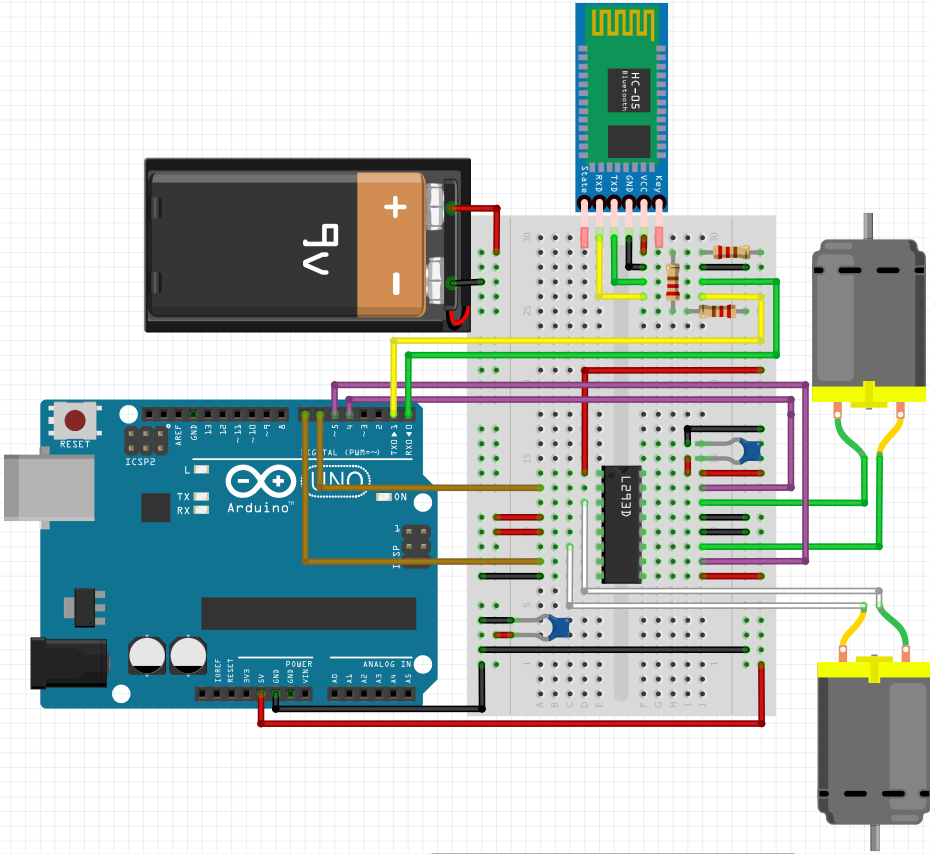
 

Figure 10 (Left) and Figure 11 (Right): Interim Report controller and RC car circuits respectively

The diagram on the left represents the headset device which has been made as small as possible. The diagram on the right represents the vehicular device. It’s not very small but it has no reason to be so that’s not a problem. It shows a 9V battery but that’s not correct – this is the only battery of that type that the diagramming tool I used allowed for. I haven’t been able to find a camera I’m happy with and due to the amount of work in my other two modules I haven’t been able to look at the coding side of camera usage, so I decided to leave that aspect of the project until I have a functioning vehicle.

Looking into drone development for extra hints I found that an IMU which includes a magnetometer could be used to see the device’s alignment with the planet’s magnetic field. This could actually be useful if I put one a 9 Degrees of Freedom IMU on the headset device and a regular magnetometer on the vehicle, because this would allow the vehicle to align itself with the controller-headset at the beginning of operation and check for alignment every few seconds to fix desyncronisation issues. The precision of this hasn’t been confirmed yet though so I’m not including it in the diagrams or description.

Next comes the Android application. This is where most of the time went into thus far to work with Bluetooth, though not much of it could be tested outside of the practical sessions. When tested, however, it’s shown to be working with regular Bluetooth but ***not*** Bluetooth Low Energy modules – they seem to use a different protocol entirely. Below is a set of screenshots of the application inside Android Studio.

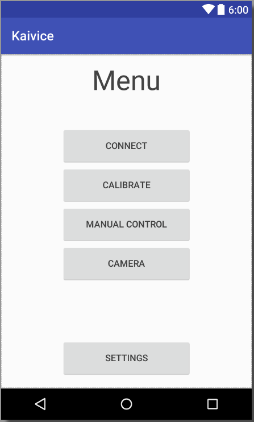
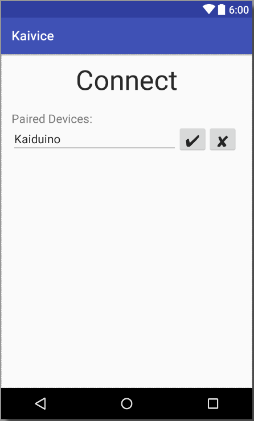
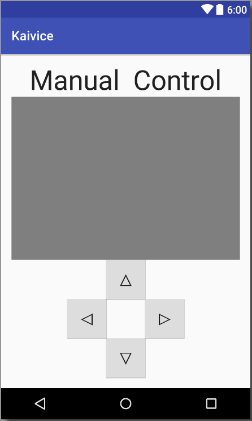
    

Figure 12: Interim Report Demo App

NOTE: The grey fields are buffered video players (Video Views) used for the camera.

The project is a little behind schedule due to problems with other modules, but is still fully expected to be finished way before the hand in date. Below is the original schedule presented in the Initial Project Proposal:



Figure 13: Interim Report Timeline

Future programme of work

The device has not yet been built, but the parts will be purchased soon so building will be the first task – this shouldn’t be an issue as the device has already been designed and should function correctly once wired up. The device won’t initially use any soldering as there may be issues in design, the modules must be soldered have also been bought in a breadboard-usable format.

A camera module will be incorporated into the design and connected to the Android app, though I’ve made the choice that will have to wait until the tests prove the device works so as not to push too far without being certain the project will succeed.

Issues and concerns

There first problem that might occur would be testable within the next week – Bluetooth may prove to be too weak a transmitter for all the information from the IMU, though this will be easily fixed by exchanging it for a WiFi module. Even if that’s easy to do, it may not be enough to stream video from the vehicle.

Another problem I might run into is calibration falling apart very quickly due to the wireless nature of information transfer – the slightest loss of information is enough to throw calibration off. A way to fix it could be to add up movements for a longer time and sending a bigger change through, though that may cause the vehicle to feel laggy.

Finally, and perhaps most obviously, soldering on the headset device may very well become an issue – unless it’s perfects designed from the start, the device may need to be scrapped and rebuilt.

References

Not many resourced had to be looked up – most of this project is built using either university-taught information or pre-existing knowledge.

PID control arduino drones mpu6050 mpu9250 gyro accelerometer - <http://www.electronoobs.com/eng_robotica_tut6.php>

Streaming Video in Android Apps - https://code.tutsplus.com/tutorials/streaming-video-in-android-apps--cms-19888