SYSC 3010 T3 Final Report

RC Camera Car

<https://github.com/sysc3010-t3>

Alec D’Alessandro (101033378)

Thao-Tran Le-Phuong (100997443)

Honor Lopes (101008909)

Igor Veselinovic (10101108)

[**1. Project Description**](#_fn2wsx6isnyd) **3**

[**2. System Architecture Design**](#_fgwbr5farwzf) **3**

[2.1 Deployment Diagram](#_rz9olq2w4b7o) 3

[2.2 Message Protocol Table](#_3ot7bxandl3p) 4

[2.3 Sequence Diagram](#_vrze54gb1fsa) 6

[**3. Discussion of Results**](#_k6g5y4ta4iw6) **7**

[3.1 Added and Updated Communication Protocols](#_epiymqxfftln) 7

[3.2 Database Schema Changed](#_2bwpwh1udclt) 7

[3.3 Removed Server VPN](#_bs70ccridvpf) 8

[3.4 Hardware Changes](#_9i831993lr1y) 8

[3.5 Problems and Solutions](#_x6obdggfvcmv) 8

[**4. Contributions**](#_e0l68s39ccbc) **8**

[**5. Appendix A**](#_e6n1iekh61ex) **11**

[5.1 CarCamApp](#_iqkouymf60lx) 11

[5.2 Car-pi](#_9o7j3v8rjvha)13

[5.3 Server](#_4jep4xv6meem) 14

[5.4 Arduino](#_btq8d0gfasrm)15

### 1. Project Description

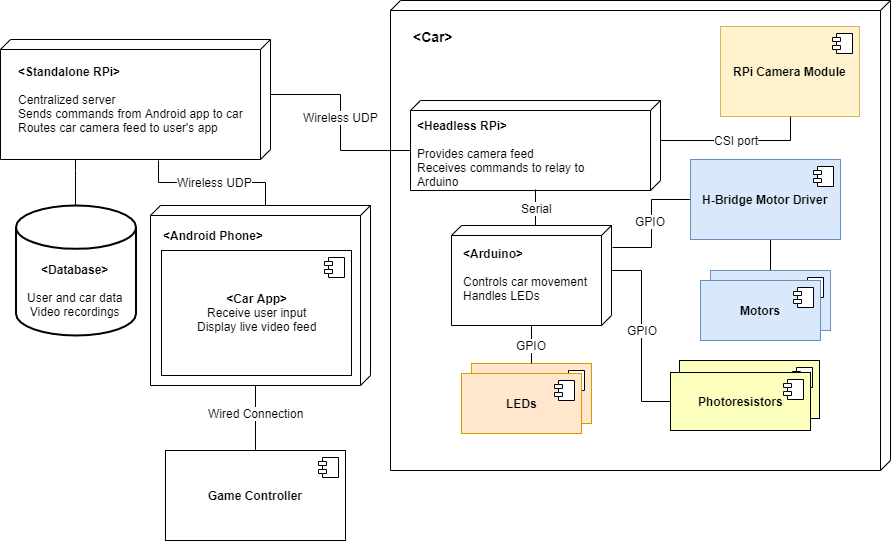
The purpose of this project was to create a remote-controlled vehicle in order to explore locations that are not visible to the user. There are many locations inaccessible or unsafe for manned vehicles such as warzones, cave systems, and biohazard spaces. These are locations where it would be unsafe to send human drivers and it would be preferable to send a remotely operated vehicle. This project would allow users to explore regions and places that are outside of their view.

The overall goal for this project was to develop a car controlled by an Android application. The functionality of this project includes a live feed video source that is being transmitted from the car and proxied through the system’s centralized server. The centralized server acts as a proxy for communication between the cars and the application. The server stores user data and associates registered cars with registered users. System users must be registered with the car through the application. The user controls the car using a game controller connected through wireless Bluetooth or a wired USB cable. The car also has automatic headlights to provide light in dimly lit areas.

### 2. System Architecture Design

#### 2.1 Deployment Diagram

The system design is shown in Figure 1. The UML diagram outlines the components necessary for implementation and the relationships between them.



**Figure 1: UML diagram for RC Camera Car system**

#### 2.2 Message Protocol Table

The RC Camera Car system uses UDP messages to send control signals between the Android application, the central server Raspberry Pi, and the Raspberry Pi that is mounted on the car. All messages in the system must be valid JSON with a mandatory “type” field which indicates the purpose of a message. The different message types, along with their unique fields and the entities that send and receive them, can be seen in Table 1.

In the system’s communication protocol, there are request messages that can have any type greater than or equal to 2, which expect to receive response messages that either have a type of 0, indicating success, or 1, indicating failure. The MOVE type of message does not expect any sort of response because these messages are constantly sent to the car, so dropped or duplicated packets are not worth trying to recover from.

**Table 1: Communication protocols**

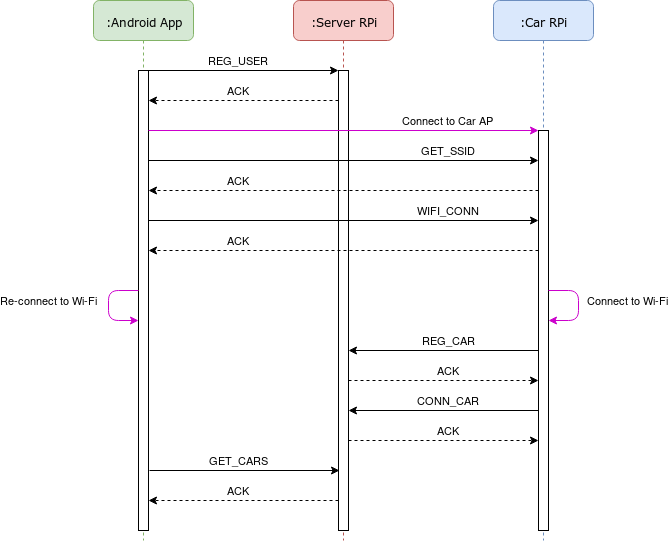
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Type #** | **Message Type Name** | **Description** | **Source** | **Destination** | **JSON Fields**  **(besides “type”)** |
| 0 | ACK (REG\_USER) | Confirm successful registration of user | **Server RPi** | **App** | “user\_id”: int |
| ACK  (LOGIN) | Confirm successful logging in of user | **Server RPi** | **App** | “user\_id”: int |
| ACK  (GET\_CARS) | Return the list of cars requested for a given user | **Server RPi** | **App** | “cars”: [  {  “name”: str,  “id”: int,  “is\_on”: bool  },  ...  ] |
| ACK  (REG\_CAR) | Confirm successful registration of user | **Server RPi** | **Car RPi** | “car\_id”: int |
| ACK  (CONN\_CAR) | Confirm successful connection of car | **Server RPi** | **Car RPi** | *N/A* |
| ACK  (GET\_SSID) | Return the list of Wi-Fi networks the car can see | **Car RPi** | **App**  (through Access Point) | “networks”: [  str,  ...  ] |
| ACK  (WIFI\_CONN) | Confirm successful connection to Wi-Fi by the car | **Car RPi** | **App**  (through Access Point) | N/A |
| ACK  (LINK) | Confirm successful linking to a registered car | **Car RPi** | **Server RPi** | N/A |
| ACK  (SET\_LED) | Confirm successful configuration of the headlights by the car | **Car RPi** | **App**  (proxied through **Server RPi**) | N/A |
| 1 | ERROR | Indicate that a request was malformed or invalid. | *ANY* | *ANY* | “error\_type”: int  “error\_msg”: str |
| 2 | REG\_USER | Register a user to the system. | **App** | **Server RPi** | “name”: str  “password”: str |
| 3 | LOGIN | Log a user in to the system. | **App** | **Server RPi** | “name”: str  “password”: str |
| 4 | GET\_CARS | Get the list of cars for a user. | **App** | **Server RPi** | “user\_id”: int |
| 5 | LINK | Link the app to a registered car. | **App** | **Server RPi** | “car\_id”: str  “user\_id”: int |
| 6 | REG\_CAR | Register a car to the system. | **Car RPi** | **Server RPi** | “user\_id”: int  “name”: str |
| 7 | CONN\_CAR | Connect an already registered car to the system. | **Car RPi** | **Server RPi** | “car\_id”: int |
| 8 | MOVE | App tells the car to move in a certain direction. | **App** | **Car RPi**  (proxied through **Server RPi**) | “x”: int  “y”: int |
| 9 | GET\_SSID | App asks for a list of available Wi-Fi networks from the car | **APP** | **Car RPi**  (through Access Point) | N/A |
| 10 | WIFI\_CONN | App tells the car which Wi-Fi network to connect to | **APP** | **Car RPi**  (through Access Point) | “user\_id”: int  “car\_name”: str  “ssid”: str  “password”: str |
| 11 | SET\_LED | App tells the car to set the LEDs to a certain mode. | **APP** | **Car RPi**  (proxied through **Server RPi**) | “state”: int |

**Table 2: Error types**

|  |  |  |
| --- | --- | --- |
| **Type #** | **Error Type Name** | **Description** |
| 0 | BAD\_REQ | Request was in an invalid format. |
| 1 | UNAUTHORIZED | Sender of the request is not authenticated. |
| 2 | SERVER\_ERR | Error occurred on the server |

#### 2.3 Sequence Diagram

The top-level sequence diagram of the RC Camera Car system is shown in Figure 2 below. This diagram shows the process of registering a new user and connecting a new car to the system. First, the user is registered on the server by request from the application using the REG\_USER protocol. The application connects to the car’s access point and tells the car which network to connect to by protocols GET\_SSID and WIFI\_CONN. After the car and app have connected to the WI-FI network, the car is registered and connected to the server by the REG\_CAR and CONN\_CAR protocols. Finally, the app requests a list of registered cars from the server.



**Figure 2: Sequence diagram for RC Camera Car system**

### 3. Discussion of Results

Three sections of the design changed during development from the original proposal. New communication protocols were added to the system. Changes were made to the database schema. The VPN was removed from the system design and was not implemented in the final project. These changes are discussed in detail in the sections below.

#### 3.1 Added and Updated Communication Protocols

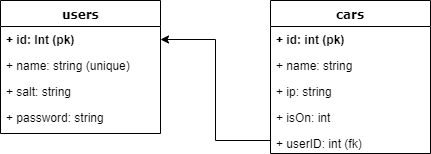
The initial project design described four protocols to be performed by the server. These were REG\_USER, LOGIN, MOVE, and Video streaming. This initial design was insufficient for providing the functionality required by the server. The communication protocols were extended to those described in Table 1. In summary the LINK, GET\_CARS, CONN\_CAR, and SET\_LED protocols were added**.** When the car authentication was implemented it revealed a need for more protocols to be handled by the server which are LINK and GET\_CARS. CONN\_CAR was separated from the LOGIN protocol to keep the code modular. SET\_LED allows for a manual override of the headlights’ states and is proxied through the server.

The video streaming protocol was initially proposed to run between the car and the application directly. The final implementation of video streaming is proxied through the server between the car and the application.

The two new protocols that are used directly between the app and the car are Get SSID and WI-FI Connect protocols. These are used during the user registration process and were initially planned to be a part of the User Registration Protocol.

#### 3.2 Database Schema Changed

During the design process, improvements were made to the database schema in order to store more relevant data. The isOn field was added to the cars table in the database. This new field is a boolean indicator describing if a car is currently connected to the application. Authentication of the car required its current status to be stored by the server. This value is set to one after the car has been connected to. The indicator is used when caching connection links the server. The updated database schema is illustrated in the image below. The previous database schema can be seen in Figure 5 of the Design Report.



**Figure 3: Database schema for users and cars**

#### 3.3 Removed Server VPN

Due to time constraints during development, the VPN was not implemented. The original purpose of the VPN was to allow the system with multiple cars to be connected on different networks. The solution used to run the system without the VPN was to keep all of the system’s elements (the cars, the server, and the application) on the same network. In order for the system to run with the system elements on different networks, the VPN would be implemented.

#### 3.4 Hardware Changes

The final implementation of the car used one light sensing photoresistor. The original design involved three photoresistors used for sensing environment brightness by an average of the three inputs. This was reduced to one photoresistor on the back of the vehicle. This was done due to a space constraint and difficulty wiring the three photoresistors in the space of the car.

#### 3.5 Problems and Solutions

One problem that occurred during project development was how much bandwidth could be used for the live video feed stream from the car to the Android application. The implemented solution was to decrease video resolution for bandwidth reducing purposes. One self-imposed constraint during development was on the complexity of the car. The car design was focused on the vehicle controls and video feed in order to limit the scope of the project. For the deployment of this system, the car was limited to Wi-Fi-enabled areas with even ground. Additionally, all system elements were connected on the same network.

### 4. Contributions

The following tables show the division of work done by team members. Table 3 describes code sections written by each team member. Table 4 shows the distribution of work on this document.

**Table 3: Code breakdown by author**

|  |  |  |
| --- | --- | --- |
| **Repository** | **File** | **Contributors** |
| CarCamApp | ConnectCar.cs, ConnectCar.xaml | **Igor** |
|  | Dashboard.cs, Dashboard.xaml | **Thao** |
|  | LoginPage.cs, LoginPage.xaml | **Thao**: Server integration  **Alec**: Design |
|  | RegisterCar.cs, RegisterCar.xaml | **Igor** |
|  | VideoStreamPage.cs,  VideoStreamPage.xaml | **Igor** |
|  | Utilities | **Thao** |
|  | UI Design | **Alec** |
| Server | handlers.py | **Igor**: Implemented handlers {LINK, CONN\_CAR}  **Honor**: Implemented handlers {REG\_USER, REG\_CAR, LOGIN, MOVE, GET\_CARS}  **Thao**: Implemented handlers {ACK, SET\_LED}  Implemented multi-threaded database connection |
|  | main.py | **Igor**: Code Architecture  **Igor, Honor, Thao**: Code for handlers |
|  | server.py | **Igor** |
|  | utils.py | **Igor**: Message type and errors  **Thao**: multi-threaded database connection |
|  | dbSchema.sql | **Thao** |
|  | TestServer.py | **Honor** |
| Car-pi | config-files/ | **Thao** |
|  | shell-scripts/ | **Thao** |
|  | tests/ | **Thao** |
|  | handlers.py | **Thao** |
|  | main.py | **Thao** |
|  | server.py | **Thao** |
|  | streaming.py | **Thao:** Implementation  **Igor:** Testing |
|  | utils.py | **Thao** |
| Arduino | hardwareControl.ino | **Thao**: hardware control code  **Igor**: LDR code update |
|  | Headlights.cpp, headlights.h | **Thao**: headlight code  **Igor**: LDR code update |
|  | Ldr.cpp, ldr.h | **Thao**: LDR code  **Igor**: LDR code update |
|  | Motor.cpp, motor.h | **Thao** |
|  | motorTest.cpp, motorTest.h | **Thao** |

**Table 4: Final Report breakdown by author**

|  |  |  |
| --- | --- | --- |
| **Section** | **Sub-section** | **Contributors** |
| 1. Project Description |  | **Honor** |
| 2. System Architecture Design | 2.1 Deployment Diagram | **Igor** |
|  | 2.2 Message Protocol Table | **Igor, Thao** |
|  | 2.3 Sequence Diagram | **Igor** |
| 3. Discussion of Results | 3.1 Added and Updated Communication Protocols | **Honor** |
|  | 3.2 Database Schema Changed | **Honor** |
|  | 3.3 Removed Server VPN | **Honor** |
|  | 3.4 Hardware Changes | **Honor** |
|  | 3.6 Problems and Solutions | **Honor** |
| 4. Contributions |  | **Honor** |
| 5. Appendix A | 5.1 CarCamApp | **Igor** |
|  | 5.2 Car-pi | **Igor** |
|  | 5.3 Server | **Igor** |
|  | 5.4 Arduino | **Thao** |

### 

### 5. Appendix A

#### 5.1 CarCamApp

This is the repository for the Android client application of the RC Camera Car system. This repository contains the C# code and XAML markup code for the client application that will communicate with the central server and the cars of the system. The application allows users to register an account, login, get their list of cars, register a new car, view a car's video stream, and control the movement of a car with any controller connected over Bluetooth or a USB cable.

**Dependencies:**

* Visual Studio
  + Xamarin.Forms

**Steps:**

1. Install Visual Studio.  
   a) Download the installer from [here](https://visualstudio.microsoft.com/downloads/).  
   b) Follow the instructions [here](https://docs.microsoft.com/en-us/xamarin/get-started/installation/?pivots=windows) to install Xamarin with Visual Studio.
2. Once Visual Studio is running, use it to open the CarCamApp.sln file located in the top-level directory of this repository.
3. Build the application.  
   a) Either connect an Android device (with Developer options and USB debugging enabled) to the computer running Visual Studio with a USB cable or run an emulator.  
   b) Click the green "run" button to build the app and launch it on the connected Android device.

**Code Structure**

* [CarCamApp.sln](https://github.com/sysc3010-t3/CarCamApp/blob/master/CarCamApp.sln): solution file for loading the project into Visual Studio
* [CarCamApp.Android/MainActivity.cs](https://github.com/sysc3010-t3/CarCamApp/blob/master/CarCamApp.Android/MainActivity.cs): contains the code for the main Activity of the app which initializes the graphical user interface (GUI) and also handles controller input
* [CarCamApp/](https://github.com/sysc3010-t3/CarCamApp/blob/master/CarCamApp):
  + [Views/](https://github.com/sysc3010-t3/CarCamApp/blob/master/CarCamApp/Views): contains the code (\*.cs) and markup (\*.xaml) for all the custom Views that make up the GUI
  + [Models/](https://github.com/sysc3010-t3/CarCamApp/blob/master/CarCamApp/Models): contains the code that represents the different components of the systems (Car.cs, User.cs, etc.), as well as utility code (Constants.cs, SocketClient.cs, etc.)
  + [Messages/](https://github.com/sysc3010-t3/CarCamApp/blob/master/CarCamApp/Messages): contains the code that represents the relevant message types for the app from the system's communication protocol

#### 5.2 Car-pi

This is the repository for the software running on the mounted Raspberry Pi on the cars in the RC Camera Car system. This repository contains the Python code for a UDP client/server that uses a single port for communication. This code base provides the behaviour for receiving control messages for moving the car and controlling the headlights (server behaviour), as well as connecting to the system's central server for initial registration and connection (client behaviour). This component of the system communicates with an Arduino over a serial connection and creates an HTTP web server for streaming a live video feed from a Raspberry Pi Camera Module. This repository also contains configuration files and shell scripts for configuring the Raspberry Pi to provide a wireless access point, while still acting as wireless client. This code and configuration has been developed and tested to run on the Raspbian operating system.

### Dependencies:

* Python 3
  + pyserial
  + picamera
* hostapd
* Dnsmasq

### Steps:

1. Set up the hardware.  
   a) Connect a Raspberry Pi Camera Module to the Raspberry Pi's Camera Serial Interface port.  
   b) Follow the instructions [here](https://www.raspberrypi.org/documentation/configuration/camera.md) to enable camera support.  
   c) Connect an Arduino to one of the Raspberry Pi's USB ports to allow for a serial connection between the two devices.
2. Set up the Raspberry Pi as a wireless access point.  
   a) Follow the instructions [here](https://www.raspberrypi.org/documentation/configuration/wireless/access-point.md).  
   b) Copy the files in config-files/ to their corresponding locations, as indicated in config-files/README.md.  
   c) Reboot the Raspberry Pi.
3. Run the car server.  
   a) Python 3 should come pre-installed on Raspbian.  
   b) Install the Python dependencies by running pip3 install pyserial picamera.  
   c) Run sudo python3 main.py 5005. The server will now be listening on port 5005 for UDP requests from apps.

### 

### Code Structure

* [server.py](https://github.com/sysc3010-t3/car-pi/blob/master/server.py): class defining the general UDP server logic for receiving and sending
* [handlers.py](https://github.com/sysc3010-t3/car-pi/blob/master/handlers.py): functions defining how the server will handle all the different message types it receives and how it will send messages as a client
* [main.py](https://github.com/sysc3010-t3/car-pi/blob/master/main.py): entrypoint for running the server; creates an instance of the Server class
* [streaming.py](https://github.com/sysc3010-t3/car-pi/blob/master/streaming.py): code for setting up an HTTP web server to host the live video stream from the camera feed
* [utils.py](https://github.com/sysc3010-t3/car-pi/blob/master/utils.py): general utility classes and functions
* [tests/](https://github.com/sysc3010-t3/car-pi/blob/master/tests): location of test code
* [config-files/](https://github.com/sysc3010-t3/car-pi/blob/master/config-files): configuration files for setting up the Raspberry Pi as a wireless access point and a wireless client
* [shell-scripts/](https://github.com/sysc3010-t3/car-pi/blob/master/shell-scripts): scripts for configuring the access point and managing Wi-Fi networks/connections

#### 

#### 5.3 Server

This is the repository for the central server of the RC Camera Car system. This repository contains the Python code for a UDP server that exposes a single port and handles request-response messages, as well as proxying control messages between client applications and the cars. The server expects that haproxy is installed on the system as it modifies the haproxy configuration to allow for the proxying of the car's live video stream. The server has been developed and tested to run on the Raspbian operating system.

**Dependencies:**

* Python 3
* sqlite3
* Haproxy

**Steps:**

1. Setup haproxy.  
   a) On Raspbian, install haproxy by running sudo apt-get install haproxy.  
   b) Copy the haproxy.cfg file from the top-level directory of this repository to /etc/haproxy/haproxy.cfg.  
   c) Run sudo systemctl restart haproxy.
2. Setup the database.  
   a) On Raspbian, install sqlite3 by running sudo apt-get install sqlite3.  
   b) Run sqlite3 RCCar.db in the top-level directory of this repository to create the database.  
   c) In the sqlite3 command-line interface, execute read dbSchema.sql to create the tables, then exit.
3. Run the server.  
   a) Python 3 should come pre-installed on Raspbian.  
   b) Run sudo python3 main.py 6006. The server will now be listening on port 6006 for UDP requests from apps and cars.

**Code Structure**

* [server.py](https://github.com/sysc3010-t3/server/blob/master/server.py): class defining the general UDP server logic for receiving and sending
* [handlers.py](https://github.com/sysc3010-t3/server/blob/master/handlers.py): functions defining how the server will handle all the different message types it receives
* [main.py](https://github.com/sysc3010-t3/server/blob/master/main.py): entrypoint for running the server; creates an instance of the Server class
* [utils.py](https://github.com/sysc3010-t3/server/blob/master/utils.py): general utility classes and functions
* [tests/](https://github.com/sysc3010-t3/server/blob/master/tests): location of test code

#### 5.4 Arduino

This is the repository for the Arduino of the RC Camera Car system. It contains the C++ code that runs on the Arduino to handle messages sent from the connected Raspberry Pi to interface with the connected hardware, such as the motors and LEDs.

### Dependencies:

* Arduino Software IDE

### Steps:

1. Install the Arduino Software IDE.  
   a) Download the installer for your operating system [here](https://www.arduino.cc/en/main/software).
2. Run the Arduino Software IDE and open hardwareControl/hardwareControl.ino within the application.
3. Compile and load the code on the Arduino.  
   a) Connect the Arduino to the computer running the application via the USB cable provided in the kit.  
   b) Press the "upload" button (to the right of the checkmark button) in the top left corner to compile the code and upload it to the Arduino.

### Code Structure

* [hardwareControl/](https://github.com/sysc3010-t3/arduino/blob/master/hardwareControl)
  + [hardwareControl.ino](https://github.com/sysc3010-t3/arduino/blob/master/hardwareControl/hardwareControl.ino): program that was written with Arduino Software to control the Arduino Board, also know as a "sketch"
  + [headlights.h](https://github.com/sysc3010-t3/arduino/blob/master/hardwareControl/headlights.h): contains C++ code for the Headlights class declaration and the associated methods
  + [headlights.cpp](https://github.com/sysc3010-t3/arduino/blob/master/hardwareControl/headlights.cpp): contains C++ code for the definitions of the Headlights class and methods
  + [ldr.h](https://github.com/sysc3010-t3/arduino/blob/master/hardwareControl/ldr.h): contains C++ code for the LDR class declaration and the associated methods
  + [ldr.cpp](https://github.com/sysc3010-t3/arduino/blob/master/hardwareControl/ldr.cpp): contains C++ code for the definitions of the LDR class and methods
  + [motor.h](https://github.com/sysc3010-t3/arduino/blob/master/hardwareControl/motor.h): contains C++ code for the Motor class declaration and the associated methods
  + [motor.cpp](https://github.com/sysc3010-t3/arduino/blob/master/hardwareControl/motor.cpp): contains C++ code for the definitions of the Motor class and methods
  + [motorTest.h](https://github.com/sysc3010-t3/arduino/blob/master/hardwareControl/motorTest.h): contains C++ code for the MotorTest class declaration and the associated methods
  + [motorTest.cpp](https://github.com/sysc3010-t3/arduino/blob/master/hardwareControl/motorTest.cpp): contains C++ code for the definitions of the MotorTest class and methods