Software Engineering for Distributed and Interactive Systems.

James Ravenhill

1 Introduction

The main outcome at the beginning of the project was to create a simple and easy to used piece of software which allows the user to control the buggy without being too complex. Both the hardware and software have been an ongoing development based on feedback given through multiple tests, which a few friends generously took part in. The main objectives of the project are listed below but not limited to:

- Reliability - Unreliable connection/data transfer is a disadvantage and creates more problems. This will be explained later in more detail.
- Multiple methods of communication - By using WiFi, Bluetooth, IR remote and PC.
- Scalability - There should be no software problems while developing the increased load of sensor/capabilities of the buggy.
- Maintainability - comments/documentation is clear to allow future developers to understand the project decisions as they happen.
- Autonomous - The buggy has the ability to move in and out of an autonomous mode.

In this document short snips of the main code will be used in explanations. The main raw code can be found in the submitted documents or on my *Github* account.

2 Design

In terms of hardware design, the overall aesthetic is not very relevant. This being said, proper cable management and reducing the risk of damaging hardware was a great challenge due to the limited mounting points on the buggy. This challenge was overcome by using blue tac or Velcro to attach the hardware to the buggy without damaging the hardware in the process.

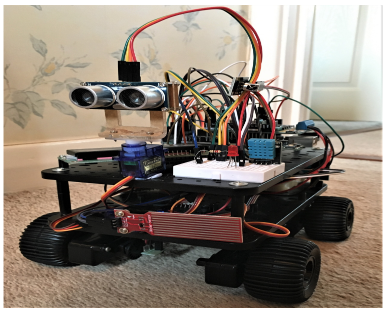
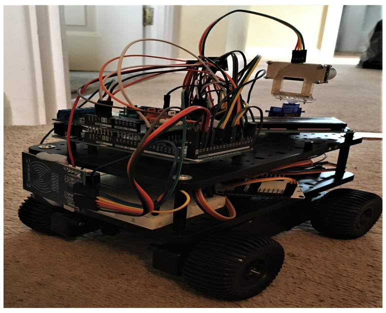
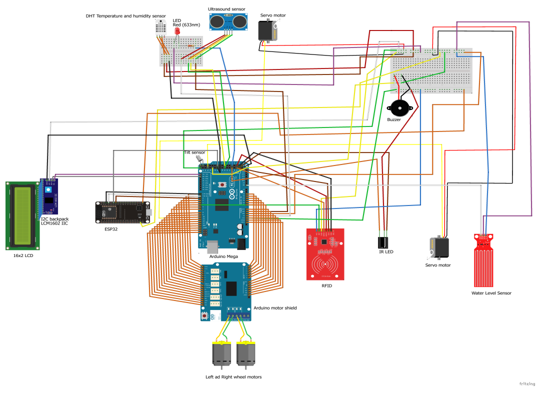



Figure 1: Left: Image of Buggy from the front left corner.

Figure 2: Right: Image of Buggy from rear right corner.

Below are two images of the main schematic of the buggy and a general communications diagram.



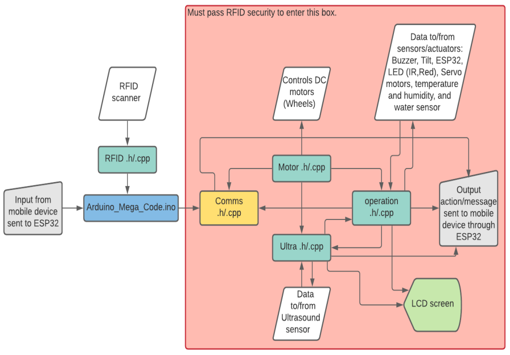

Must pass RFID security to enter this box.
RFID scanner
RFID .h/.cpp
Input from mobile device sent to ESP32
Arduino_Mega_Code.ino
Comms .h/.cpp
Controls DC motors (Wheels)
Motor .h/.cpp
Data to/from sensors/actuators: Buzzer, Tilt, ESP32, LED (IR,Red), Servo motors, temperature and humidity, and water sensor
operation .h/.cpp
Output action/message sent to mobile device through ESP32
Ultra .h/.cpp
Data to/from Ultrasound sensor
LCD screen

Figure 3: Left: Schematic of Buggy.

Figure 4: Right: Communication diagram of the Buggy

The software for the Arduino has been set up into header (.h) files and source code (.cpp) files where they are then called in the main code. There are many benefits to this such as the code is organised and tidy allowing future and current developers to debug with ease, addition of any new code can be easily integrated into the system and the code can be easily maintained.

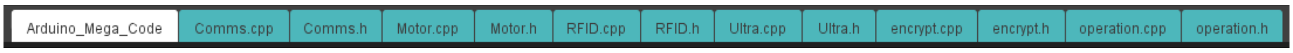

Arduino_Mega_Code | Comms.cpp | Comms.h | Motor.cpp | Motor.h | RFID.cpp | RFID.h | Ultra.cpp | Ultra.h | encrypt.cpp | encrypt.h | operation.cpp | operation.h

Figure 5: Tabs for Arduino mega main code.

In Figure 4 you can see how the system communicates between source files. The main system that operates the buggy can not be accessed without first passing the RFID security tag challenge. Further development to encrypt data that is sent between the mobile device and ESP32 needs to be done, a attempt to create a rotation 13 encryption has been attempted but needs to be placed correctly within the architecture of the system.

3 Implementation

The software for the ESP32 comprised of individual sketch files which correspond to the use of WiFi or Bluetooth and for both due to the ESP32 being dual-core. Transmission control protocol (TCP) has been used when using WiFi instead of user diagram protocol (UDP) as is it more reliable, has Flow control so will not overwhelm receiver and has congestion control. A 'handshake' is used to initiate communication and then multiple packets are sent. TCP may not be as fast as UDP but due to the project objectives, reliability is preferred. This being said TCP is slower by a very small margin so is not a large hindrance on the usability of the system.

A distributed system is a system which coordinates is actions by communicating between components via passing messages. The example for this project are listed and explained below.

- Mobile device-ESP32-Arduino maga: The mobile device communicated to the ESP32 through WiFi (TCP) or Bluetooth, the message that is received by the ESP32 in then passed onto the Arduino mega by universal asynchronous receiver/transmitter (UART). UART is not a communication protocol but is rather a physical circuit where serial data is transmitted and received. This emphasises the reliability factor as no data will be lost over the wired connection. When the Arduino receives a message/command the source code will then action this. If feedback is to then be sent back to the Mobile device it will do so in reverse process, the data will be sent to the ESP32 via UART and then be sent to the mobile device by WiFi/Bluetooth.

- IR signal-Arduino: The IR LED input signal was initially received in the mini-bread board but this cause problems while testing due to high volumes of noise, it was therefore moved directly onto the Arduino mega. This therefore means that the IR remote can by used at the same time as the mobile device. The LCD used the I2C bus to print messages on the screen when temperature, humidity and water sensor readings are requested.

- PC-Arduino: The PC is connected directly to the Arduino mega and is controlled using the serial monitor. It is very useful for conducting quick simple tests, if new code is added to the software for example.

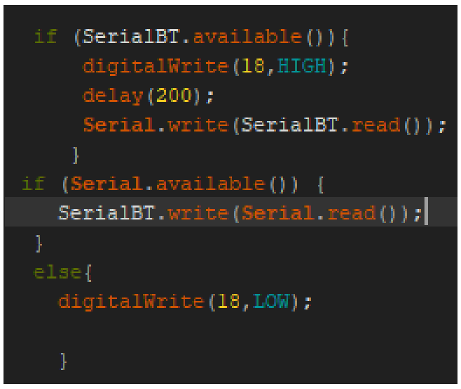
- ESP32 disconnection feature: When the ESP32 disconnects from the WiFi router or Bluetooth then a signal will be sent to the Arduino mega. When this signal is recieved the buggy will stop, the LCD screen will be cleared and a red LED will light up to alert the user.

The interactivity between the buggy and the user (besides the obvious mobile device (WiFi/Bluetooth), PC communication and IR remote) consist of the following lower levels of interactiveness:

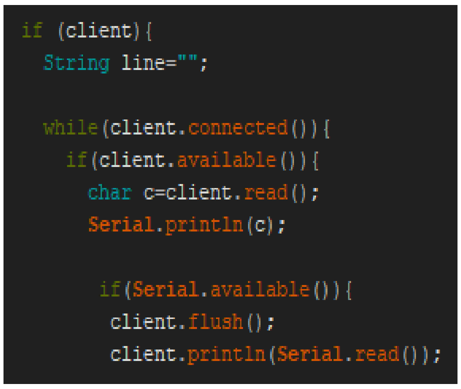
- RFID scanner- This provides security as the RFID tag/card must be presented before any use of the buggy can take place.
- Tilt sensor- This sends a message to the mobile device/PC aswell as sounding an alarm to communicate to the user that the buggy has tipped over due to complex/uneven terrain.
- Disconnection LED- The red LED on the front of the buggy lights up when the WiFi/Bluetooth connection drops.
- Ultrasound sensor- This interacts with its surroundings to gain distance information. This information can then be actioned on accordingly, for example while in an autonomous mode distance information will be used to make decisions and avoid obstacles.

These may seem like simple solutions but they are effective and prove effective during testing. They also provide a level of security to the buggy, for example if a new RFID tag is to be added it can only be done so by adding it to the source code.

when communications have been received from the mobile device by the ESP32 the ESP32 will receive this message and send it to the Arduino via UART as mentioned above. Any messages sent from the Arduino are communicated back in the same way.



```
  if (SerialBT.available()){
      digitalWrite(18,HIGH);
      delay(200);
      Serial.write(SerialBT.read());
     }
 if (Serial.available()) {
    SerialBT.write(Serial.read());
  }
  else{
    digitalWrite(18,LOW);

    }
```

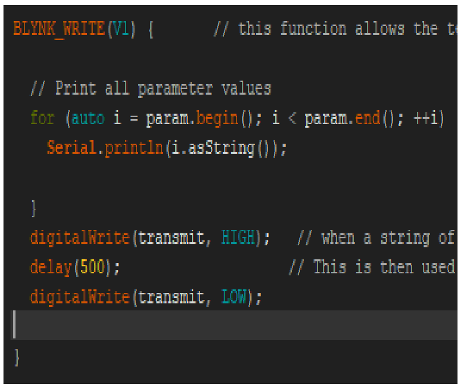


```
if (client){
  String line="";

  while(client.connected()){
    if(client.available()){
      char c=client.read();
      Serial.println(c);

       if(Serial.available()){
        client.flush();
        client.println(Serial.read());
```

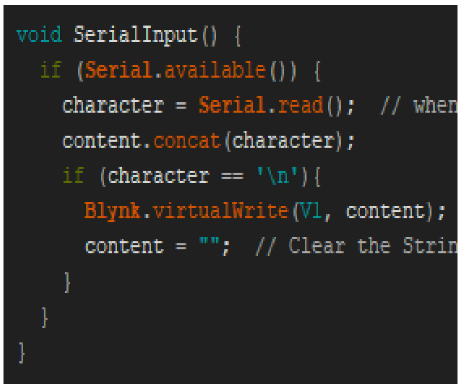
Figure 6: Left: ESP32 Bluetooth main loop code.

Figure 7: Right: ESP32 standard Wifi main loop code.

Blynk is a library that when connected to its app on any mobile device provides a very aesthetic design for the user, it is also very customizable for varying project requirements. The way it has been used in my case is of a similar procedure but is not as straightforward as the Bluetooth or standard WiFi set up. As you can see below, two functions are created with `'BLYNK_WRITE'` being a Blynk library that reads commands from the mobile device and sends them to the Arduino and `'SerialInput'` being the function which sends messages from the Arduino to the mobile device.



```
BLYNK_WRITE(V1) {       // this function allows the t

  // Print all parameter values
  for (auto i = param.begin(); i < param.end(); ++i)
    Serial.println(i.asString());

  }
  digitalWrite(transmit, HIGH);   // when a string of
  delay(500);                    // This is then used
  digitalWrite(transmit, LOW);

}
```

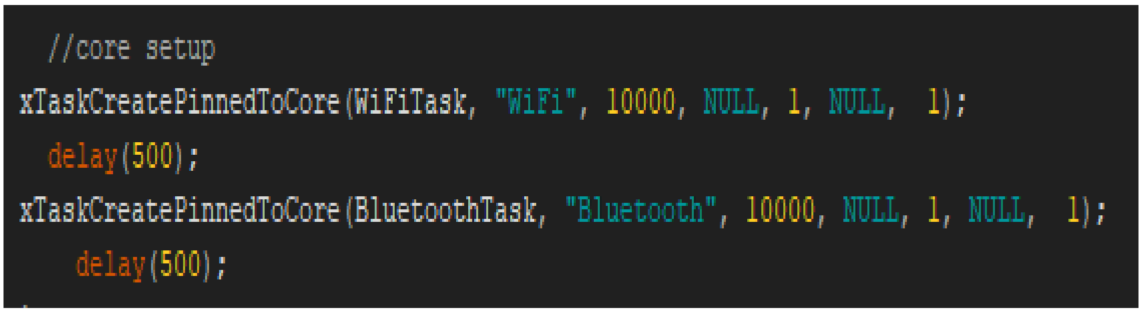


```
void SerialInput() {
  if (Serial.available()) {
    character = Serial.read();  // when
    content.concat(character);
    if (character == '\n'){
      Blynk.virtualWrite(V1, content);
      content = "";  // Clear the Strin
    }
  }
}
```

Figure 8: Left: ESP32 Blynk read from the mobile device and send messages to the Arduino Mega.

Figure 9: Right: ESP32 Blynk read meaasges from arduino and send to mobile device

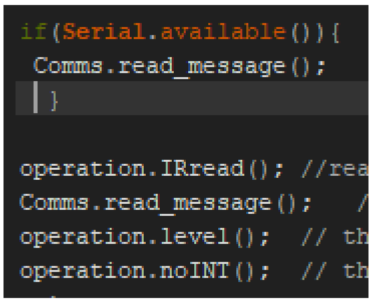
The ESP32 is a Dual core micro-controller and in theory can be used for a combined Bluetooth and WiFi mode. As you can see from the figure below this was attempted and does work, however there is one issue issues. Because the procedure for sending messages back to the mobile devices are virtually the same for WiFi and Bluetooth the ESP32 has no distinction between which device it has received a command from and therefore which device to send messages back to.



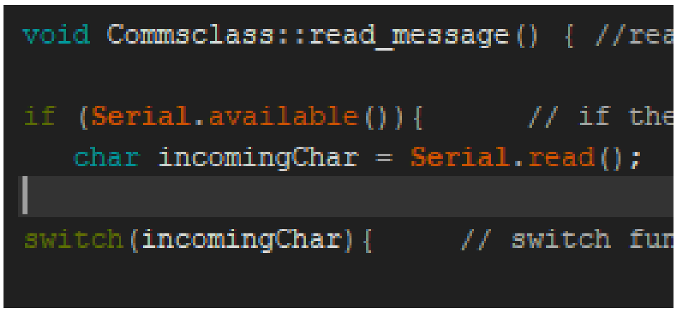
```
  //core setup
xTaskCreatePinnedToCore(WiFiTask, "WiFi", 10000, NULL, 1, NULL,  1);
  delay(500);
xTaskCreatePinnedToCore(BluetoothTask, "Bluetooth", 10000, NULL, 1, NULL,  1);
    delay(500);
```

Figure 10: Setup for ESP32 dual core WiFi and Bluetooth.

Looking at the figure below you can see the main loop for the Arduino mega. The commands here are reading the serial monitor when the buggy is connected to the PC as well as the IR remote control reader and the ESP32 reader (this reads messages from the mobile device). Other commands that monitor the tilt sensor and the Wifi/Bluetooth connection are at the bottom of Figure 6.



```
if(Serial.available()){
 Comms.read_message();
 }

operation.IRread(); //rea
Comms.read_message();   /
operation.level();  // th
operation.noINT();  // th
```



```
void Commsclass::read_message() { //rea

if (Serial.available()){       // if the
   char incomingChar = Serial.read();

switch(incomingChar){      // switch fun
```

Figure 11: Left: Arduino mega main loop

Figure 12: Right: Read incoming message

When a command is received from the mobile device it will enter a `switch` loop, here is where commands are executed. each command corresponds to a function which exist in other source code files.

4 Testing

The variety of testing methods used from start to finish have varied from unit and integration testing to acceptance and regression testing. The unit testing has taken place when a new operation/function has been added to the source code and is performed by myself for verification. The integration tests have then been performed by others to make sure that the new function works within the rest of the system and performs how it is expected to perform (acceptance testing). As the project has developed regression testing has been used to verify that any changes have not affected the prior system.

One of the first problems discovered was with the ultrasound sensor (USS), in complex environments the USS would not recognise the correct distance and the buggy would therefore crash. Complex environments such as chair legs and/or approaching a wall at a 45° angle. The reason for the USS difficulty in getting the correct distances is due to its transmitter-receiver setup, if either one is blocked then the correct distance cannot be obtained and the buggy crashes. For this reason, to reduce any damage to the hardware a tilt sensor was added. This feature when triggered will stop any movement of the buggy and will send a warning signal to the mobile device terminal.

///insert pic of unstable buggy message aswell as a chair leg infront of the ultrasound sensor with a distance reeding.

When the decision to add a IR remote which is able to control the buggy was made, initially there were issues. Noise from ambient light, other IR sources and even electrical signals within the breadboard caused overflow issues. Any command received from the IR remote would not be read and processed.

///add IR overflow screenshot

What I found was happening is that any electrical signal to or from the ultrasound sensor was being spread throughout the pins ports in the breadboard (This is due to it being a small cheap breadboard) and this interfered with the IR LED electrical signal. To over come this the signal pin for the IR LED was relocated to the main Arduino mega board, This reduced the noise considerably.

Of course loss of communication is a important problem. To combat this the ESP32 monitors WiFi/Bluetooth connection and if any connection is lost the ESP32 sends a signal to the Arduino Mega. This signal will stop the buggy and turn the front red LED light on. No message is sent to the mobile device as there is no connection so it would be pointless. When the connection is lost the LCD is also cleared. The ESP32 will attempt to restart the setup instructions to regain connection.

//// add photos os bluetooth disconnect with led lit up

Some basic tests on the accuracy of the ultrasound, temperature, humidity and water sensor were also conducted. When testing the accuracy of the ultrasound sensor a ruler was placed at the base of the sensor and a book was moved closer in intervals of 5cm starting from 120cm. The recorded result was then noted and compared against the actual result.

///Add photo of ultrasound setup and results.

When It came to testing the temperature, ///////

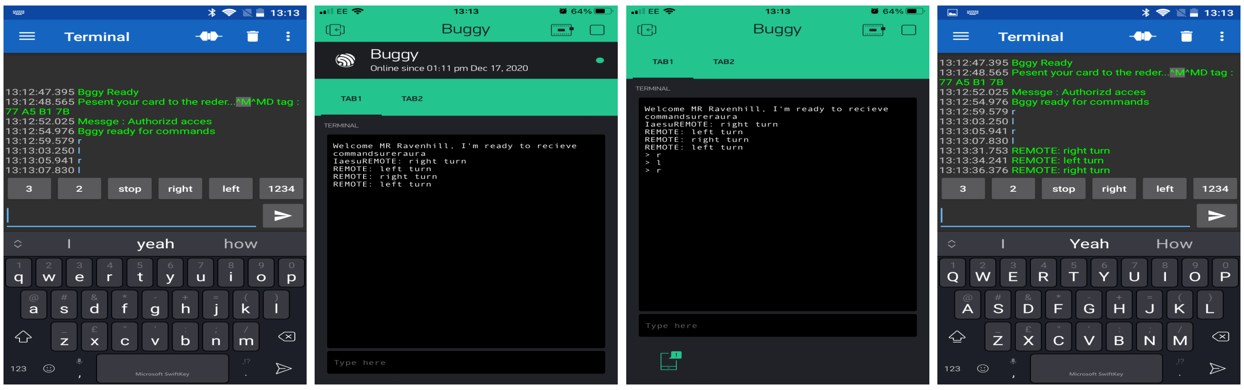
The water sensor was slightly different when it came to testing. The software is setup to provide a ranged output for the sensor reading (Empty, low, medium and High). These ranges can be calibrated in the source code, but I still tested the sensor to see how the difference in water levels affected the output. The sensor was split up into intervals and water is applied in cumulative segments. The output is then recorded, the output can be recalibrated depending on how the user wants it to be set up.

/// do water sensor test.

5 Evaluation

//// pros and cons of WIFI and bluetooth. //// limitations to each distributive method.

As you can see below when a command is sent from the Bluetooth terminal any messages responding to this command come back through the WiFi terminal and visa versa.



```
13:12:47.395 Bggy Ready
13:12:48.565 Pesent your card to the reder...^M^MD tag :
77 A5 B1 7B
13:12:52.025 Messge : Authorizd acces
13:12:54.976 Bggy ready for commands
13:12:59.579 r
13:13:03.250 l
13:13:05.941 r
13:13:07.830 l
```

```
Welcome MR Ravenhill, I'm ready to recieve
commandsureraura
IaesuREMOTE: right turn
REMOTE: left turn
REMOTE: right turn
REMOTE: left turn
```

```
Welcome MR Ravenhill, I'm ready to recieve
commandsureraura
IaesuREMOTE: right turn
REMOTE: left turn
REMOTE: right turn
REMOTE: left turn
> r
> l
> r
```

```
13:12:47.395 Bggy Ready
13:12:48.565 Pesent your card to the reder...^M^MD tag :
77 A5 B1 7B
13:12:52.025 Messge : Authorizd acces
13:12:54.976 Bggy ready for commands
13:12:59.579 r
13:13:03.250 l
13:13:05.941 r
13:13:07.830 l
13:13:31.753 REMOTE: right turn
13:13:34.241 REMOTE: left turn
13:13:36.376 REMOTE: right turn
```

Figure 13: From Left to Right: Bluetooth terminal giving commands. WiFi terminal receiving messages. WiFi terminal sending commands. Bluetooth terminal receiving messages.

6 Code quality/remarks

////this may get cut if needed as in more an non written task ////discuss multi and single core methods in a ReadMe file!!!

///Layers- layered system to allow for easy maintenance, ease of updating/developing the system. ADD A DIAGRAM OF THE LAYERS HERE.

7 Demonstration

////film the video and then refer to it for the following subsections

7.1 Distribution

///Define what a distributed system is

7.2 Interactive

///Define what an interactive system is

7.3 Performance

8 Conclusion

9 Reference Library and appendices

9.1 Github account

Click here for Github hyper link

or copy and paste the URL into your web browser. https://github.com/james8268/Buggy

List of Figures

9.2 Third party libraries

`arduino.h`, `LiquidCrystal_I2C.h`, `IRremote.h`, `Wire.h`, `dht.h`, `SPI.h`, `MFRC522.h`, `WiFi.h`, `WiFiClient.h`, `BlynkSimpleEsp32.h`, `Servo.h` and `BluetoothSerial.h`

9.3 Test Comments

Test 1

Getting to grips with the Bluetooth, 28/10/2020: The device worked exactly how it should every command button worked as it should. The controls were very easy to understand and get to grips with which means I could use the buggy how I wanted with no issues. Only thing I would like to add is that the left and right turns be a little less aggressive. So instead of a 90 degree turn maybe a 45 instead. Also I would like the distance of the sensor to be displayed on the mobile device as well as the LCD screen on the buggy so I dont have to be stood directly over the buggy to know this information, as I think this would make it more user friendly. Other than that, it was flawless.

Test 2

This test is like the last with a few minor changes. This time around the buggy is in a combined remote Bluetooth and autonomous mode. The autonomous element will allow the buggy to roam freely while avoiding obstacles.

Acting on the previous test a feature of being able to see the reading of the ultrasound sensor has now been added and can be viewed in the mobile device terminal as well as the turns being less harsh.

Test 2.1, using the combined autonomous and Bluetooth mode, 04/11/2020: self-roaming was effective, the buggy avoided obstacles well, although had trouble once it was under a certain distance from complex obstacles. commands were easy to pick up and the buggy was easy to manoeuvre, although lag time could be reduced, or possibly a cut off after x number of commands. possibly introduce a short cut to speed one and/or options for varying stop times, so that when flustered a driver can take time to re-think.

Test 2.2, test is the same as previous (code remains the same as the last test), 05/11/2020: the buggy avoided the obstacles well however due to lag it made it difficult to manoeuvre, also would be much easier and efficient for the driver if there was a cut off command so that the user can re-access the situation and move from there.

Test 3

The buggy now can roam around autonomously while avoiding obstacles and this can then be stopped and be used remotely again. Its easy to switch between the two, using the mobile device terminal the ”roam” hotkey enters the buggy into an autonomous mode(which can be seen when the android device is tilted on a landscape view and the screen will show more potential hotkeys). when any button/command is used in the terminal the buggy will then exit the autonomous mode and reverts to a remote mode.

Test 3, testing combined autonomous and remote modes, 07/11/2020: I enjoyed the new features on the handset. The fact that the buggy now had a roam feature was very interesting. The fact it could now make decisions on what was in front of it and change the course on which it was travelling due to an object being in its way was fascinating. A cool feature would be if it could decide which way it wanted to turn. For instance, left or right depending on the amount of space it has either side, this depends on the amount ultrasound sensors it has available. All the other buttons worked well and did what I wanted when I asked it to.

Test 4

A tilt sensor and servo motor have now been added, the use of WIFI connection is also up and running. Security measures have been added so if WIFI or Bluetooth connection is lost it will stop the buggy and turn on the front red LED. The servo motor allows the buggy to look left and right and then decide on which direction to move. The tilt sensor will monitor the buggy and if it capsizes or is flipped over it will stop the buggy and turn on a buzzer as well as notifying the user if in remote mode.

tests to be completed:

- turning system for servo ultrasound sensor, just to check its working right.
- tilt sensor while in autonomous mode while moving.
- temp and humidity sensor and water sensor while moving and stationary.
- disconnect functions via Bluetooth, by trying to connect another device (already known that two devices cant connect over Bluetooth).

- disconnect from WiFi, turn off router.
- test overall usability.

Test 4, 23/11/2020, testing the new obstacle avoidance system, tilt sensor and the temp, humidity and water sensor (all in Bluetooth mode): I found the new obstacle avoidance system was very good the Buggy made very accurate adjustments when it came to an obstruction. the fact it was able to stop look around and decide for itself which way to go was very impressive. the tilt sensor that has been integrated into the buggy was a very impressive feature. especially since you had the use of this both in autonomous mode and just when controlling the buggy, yourself. also having the bonus of a water, temp and humidity sensor on it was very good. being able to see those readings was a nice feature to have the use of. I also found the use of the IR remote to be far more simplistic and easier to use than the Bluetooth device I much preferred it. there was a few points I feel could do with some improvement. for instance, when you are on the move and then ask the buggy to perform a water reading it stops does this command but then waits for another new instructions. I would like it to instantly carry on with the previous command once the reading has been carried out with no need to give it further instruction from either the Bluetooth device or IR remote. other than that, the buggy was very good.

Test 5

A RFID security tag has been added for security.

Test 5, first time using WIFI set up and looking at disconnection issues. To then compare the WIFI and Bluetooth usability, 11/12/20:

Using BLYNK: was detailed and easy to use. Very easy to read the screen. All commands worked straight away. Liked the safety feature as well the fact you had to use a key fob to be able to use the buggy. Also, the fact it tells you on the buggy with a red led that it is disconnected from the internet and you no longer have control of the buggy.

Standard WIFI: wasnt as easy to use. Didnt like the fact every time you placed a command the keyboard went away so you had to click on the box again to load a command. When I asked for a temp reading it wasnt displayed on the WIFI app. Just gave me a random number. Also, when disconnected from the internet the red led light didnt show on the buggy to indicate it was inoperable, however it did show on the WIFI app that it had been disconnected from the network.

Comparison to Bluetooth: WIFI was good but the Bluetooth was far better in my opinion as I can used the buggy anywhere at any time I like with ease. Whereas with the WIFI you must have that connected to be able to use the buggy. Also, the Bluetooth has the hot keys which made the whole experience of using the buggy more user friendly and easier.