# 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.

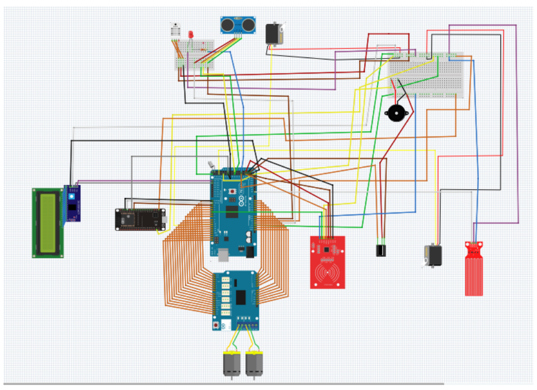
In this document short snips of the main code will be used in explanations. The main raw code can be found in the submitted documents or on my *Github* account.

## 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.

///////insert two pics of the buggy.

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



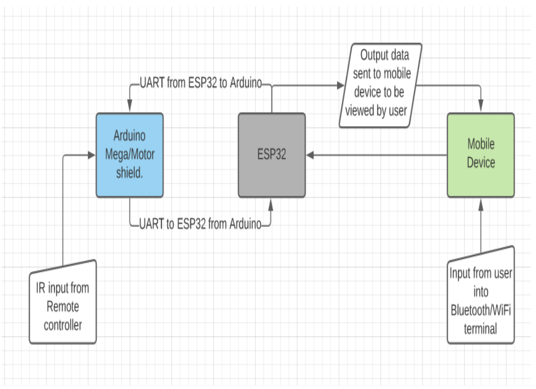

UART from ESP32 to Arduino
Output data sent to mobile device to be viewed by user
Arduino Mega/Motor shield.
ESP32
Mobile Device
UART to ESP32 from Arduino
IR input from Remote controller
Input from user into Bluetooth/WiFi terminal

Figure 1: Left: Schematic of Buggy.

Figure 2: Right: general communication graph of the Buggy

The software for the Ardunio 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

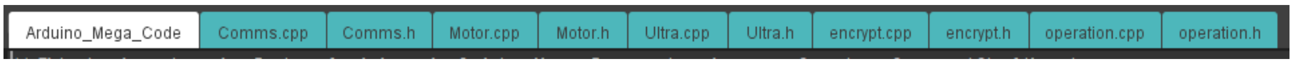


Figure 3: Tabs for Arduino mega main code.

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.

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 reciever 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.

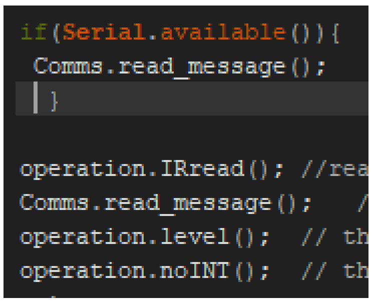
- Ultrasound sensor- This interacts with its surroundings to gain distance information. This information can then be actioned on accordingly, for example while in an autonomous mode distance information will be used to make decisions and avoid obstacles.

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, the RFID tag can only be added to the source code.

## 3 Implementation

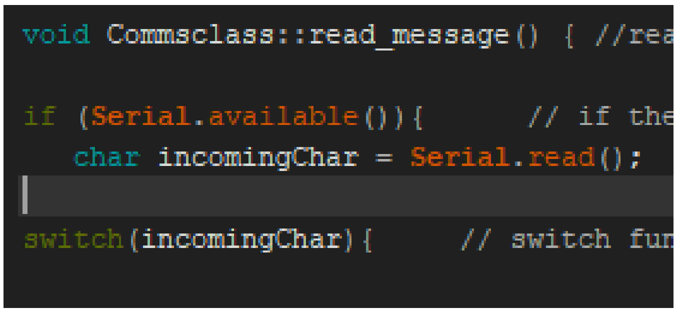
Looking at Figure 4 you can see the main loop for the Arduino mega. The commands here are reading the serial monitor when the buggy is connected to the PC as well as the IR remote control reader and the ESP32 reader (this reads messages from the mobile device). Other commands that monitor the tilt sensor and the Wifi/Bluetooth connection are at the bottom of Figure 4.

3rd Party libraries: `arduino.h`, `LiquidCrystal_I2C.h`, `IRremote.h`, `Wire.h`, `dht.h`, `SPR.h`, `MFRC522.h`



```
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 fur
```

Figure 4: Left: Arduino mega main loop

Figure 5: Right: Read incoming message

## 4 Testing

## 5 Evaluation

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

## 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