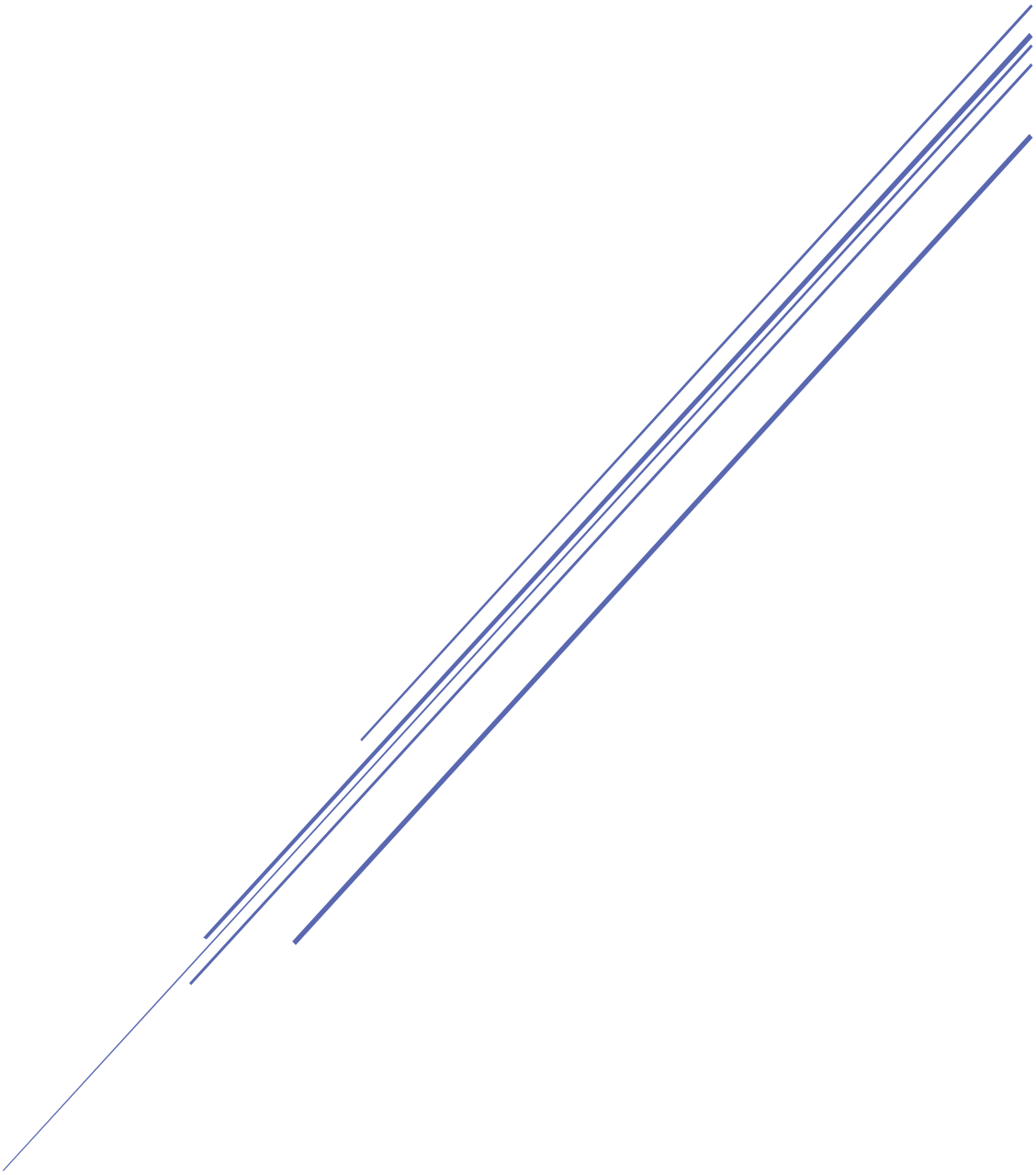
REQUIREMENTS DOCUMENT FOR MQ AUTONOMOUS TRANSIT SYSTEM

Team 4 Communications 1



Macquarie University
ENGG2000/3000

Table of Contents

1 Version History

Table 1. Version History

Version	*Date*	*Comment*	*Last Editor*
v.4	04/10/2020	Detail Design Document	Rajiv
v. 3	22/8/2020	Requirements Document	Allan
v. 2	19/8/2020	Buddy Marking	Neil
v. 1	5/8/2020	Start	Dylan

2 Division of Labour and Workload Acknowledgement

Table 2. Division of Labour

Member name	*SID*	*Cohort (ENGG2000/3000)*	*Sections Responsible*	*Overall %*
Dylan Formosa	45877300	ENGG2000	Functional Requirements, Detailed Design	18.18%
Neil Roberts	45953171	ENGG2000	Scope, Terms, Abbreviations, Detailed Design	18.18%
Rajiv Mehta	45433062	ENGG3000	Subsystems, Assumptions	9.09%
Allan Kung	44759444	ENGG3000	Conceptual Design, Detailed Design	18.18%
Jawad Ahsen Abbasi	45573301	ENGG3000	Constraints, Performance Requirements, Detailed Design	18.18%
Jack Mahon	45291896	ENGG2000	Problem Definition, Interface Requirements, Detailed Design	18.18%

3 Introduction

3.1 Scope

This document is a detailed view of the processes taken by the ENGG2000 and ENGG3000 students of Team 4 Communications 1 (T4C1). The task of designing, building, and maintaining the monorail system is explored in depth throughout the document, showcasing the processes that will be completed and have been undergone during the semester.

Throughout the document, there are details that relate to many aspects of the monorail system including the problem definition of the task, functional, interface, and performance requirements of the monorail system. Designs are utilised throughout the process of documentation in order to effectively display the basic understanding of the described systems. It allows for a view of the flow of data in an alternate format.

The documentation is a critical element of the product as it allows for an understanding of the desired product in an easy to read format. The necessity of the documentation is to allow for an overall view of the requirements from the customer to develop the monorail system. Through the process of a clear and concise document, it will allow for members of T4C1 and other teams to follow the document and understand the final goal and how it was achieved.

The document will receive many changes throughout the lifecycle development of the system in order to continually be updated with any changes made to our desired goal. This documentation is worked by members of T4C1 only.

3.2 Terms, definitions, and abbreviated terms

Table 3: List of abbreviations

Abbreviation	***Definition***
I2C	Inter-Integrated Circuits
LCD	Liquid Crystal Display
MEGA	Arduino Mega
MVP	Minimum Viable Product
T4C1	Team 4 Communications 1
T4C3	Team 4 Communications 3
T4M2	Team 4 Motions 2
T4S1	Team 4 Structures 1
UNO	Arduino Uno
RX	Receiver
TX	Transmitter
IDE	Integrated Development Environment

4 Problem Definition

The problem that has been posed by the client is to create a working monorail system. This monorails should be able to traverse the designated track and stop at each required station. Numerous restrictions have been placed on the monorails by the client including but not limited the carriage having to be able to hold 2 kg, the control box being implemented on a separate console and should be connected via Bluetooth, the software being implemented in C and a budget of $100 for any material that is not already provided. Therefore, in order to meet the demands of the client, subsystem teams have been created to meet the time requirements and produce the MVP.

T4C1 has been tasked with designing the control box for 'Monorail 1'. Specifically, the team is in charge of designing and implementing the state machine that sends the correct output from the UNO to the MEGA when a specific button is pressed. The team is also in charge of designing and implementing the LCD screen to display the current position of the monorail. The last specification that is required of T4C1 is to also implement the Bluetooth between the UNO and MEGA.

4.1 Deliverables

In the completion of the Monorail project, the client is expected to receive a working autonomous transit system that meets the client's design and constraint requirements. The system will be operated through the use of Control Box which has four buttons that define the movement and speed of the Monorail as well as the operations of the Monorail carriage doors. Through the use of sensors, these operations will be collaborating with distinct coloured lines on the track to provide an operationally smooth system.

With the completion of the Monorail project, expectations of what will be delivered include:

Table 4: List of Deliverables

Deliverables	**Expected Date**
Extensive documentation on the requirements of the project, and its conceptual and detailed design of the Control Box, state machine, and Bluetooth Connection.	Week 8 - 4/10/2020
Documentation of the T4C1's subsystem which includes: - The sections which are being tested - The testing methodology - Testing results	Week 11
A review of the design of the subsystem, highlighting any potential future improvements, limitations, and unimplemented features.	Week 11
Information on the End of Life procedure for correct disassembly and recycling.	Week 11
Bitbucket repository providing all previous of the current implemented code.	Week 11
A final presentation from T4C1 members highlighting the subsystems implementation to the overall project	Week 13

4.2 Subsystems

The subsystems of the monorail project can be divided into three macro subsystems with classifications: structures, motions, and communications. These subsystems can be further divided into micro subsystems. Each micro subsystem must interface with its macro system and the appropriate subsystems in the other macro systems. A layout of these subsystems can be seen in Figure 1.

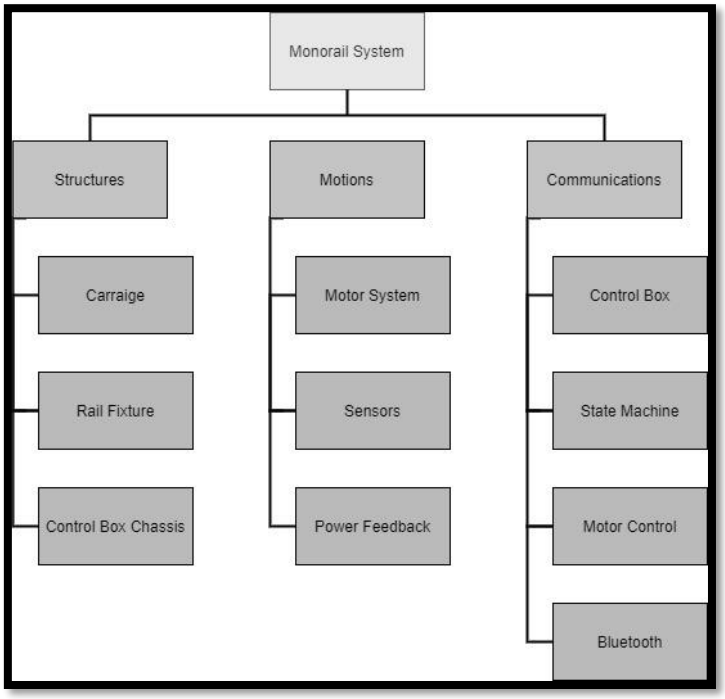


Figure 1. Layout of subsystems for monorail

4.2.1 Structures

The structure subsystems of the metro project are concerned with the physical structure of the project. This includes: the design and durability of the monorail carriage, the rail fixtures and their suitability for the project, and the physical chassis of the control box.

4.2.2 Motions

The motions subsystems of the metro project are concerned with the electrical make-up of the project. This includes: the design and efficacy of the motor system, the accuracy of the sensors, and the power supply used for the monorail.

4.2.3 Communications

The communications subsystems of the metro project are concerned with the software used to control the project. This includes: the input processing and appropriate output of the control box, the system used to control the speed of the motors, the state machine to keep track of the current states of the control box and the monorail itself, and the Bluetooth connectivity between the UNO and the MEGA.

4.2.4 Subsystems for T4C1

Control box

- The internal logic of the control box needs to be able to handle inputs from the controls and display appropriate information on the LCD screen.

State machine

- There must be an accurate account of the entire systems' movement and position. This account needs to be malleable when given appropriate input and needs to retain integrity when given bad input.

Bluetooth connection

- The connection between the two Bluetooth modules needs to be established upon activation of the monorail system and must be reliable for its entire operation.

4.3 Assumptions

The subsystems for T4C1 requires many assumptions that should be fulfilled in order to make sure that the subsystems operate as intended within the project. A list of these assumptions can be seen in Table 4.

Table 4. Assumptions for T4C1's subsystems

Assumption	***Description***
Bluetooth	For the control box to effectively control the movement of the monorail, there must be a reliable Bluetooth connection transmitting accurate information. It is assumed that the Bluetooth modules and the connection is reliable and the information transmitted between the UNO and the MEGA is accurate.
Code Development	For the code to be written in segments and successfully integrated it must be written in compatible environments. It is assumed that all code will be written in the C programming language within the Arduino IDE.
Circuit Design	For the system to operate effectively, the circuitry must be reliable and the specifications must be accurately communicated to the communications team. It is assumed that the circuitry is in working order and the connections between the pins and each piece of hardware are accurately mapped.
Structural integrity	For the system to perform correctly, it is required that the physical structure of the monorail and the control box is built to the required specifications. It is assumed that the placements of the motors and sensors are such that they can perform effectively and accurately.
Power supply	For the control box to operate effectively, a reliable power supply is necessary. It is assumed that the power supply for this project is reliable and will produce enough constant electricity for the MEGA and UNO to operate normally.
Accurate sensors	In order to properly keep track of the monorails position, there must be accurate information collected via the sensors. It is assumed that the sensors used to update the monorails position are in working order and the signals they send are accurate.

5 Requirements

The requirements refer to how the subsystems that T4C1 has been tasked, should operate within the final version of the project. The section is split into three main categories, 'Functional', 'Performance' and 'Interface' requirements. The functional requirements refer to what the subsystems should do within the project i.e. the LCD should output the correct text at different states. The performance requirements refer to the level that the functional requirements should operate at and in turn, provide the base standard for testing. The last requirement is the interface requirements. This refers to how T4C1's subsystems will operate with the other subsystems designed and implemented by other teams within the project, for example, the UNO should send an Integer signal to the MEGA. A list of full requirements can be seen under each subsection of requirements.

5.1 Functional Requirements

Table 5. Functional Requirements for T4C1

ID	*Requirement*	*Description*
FR_1	Digital Input from Buttons	A control button shall be pressed which sends a digital signal to the MEGA from the UNO via Bluetooth showing the status of the system on a display. When a button is initially pressed the UNO shall read a HIGH and stay HIGH when the button is released. When pressing the button once again the UNO shall read a LOW and stay LOW when the button is released.
FR_2	Digital Output of Buttons	The digital output shall be sent from the UNO to the MEGA when a button has been pressed.
FR_3	Input I2C LCD Screen	The input signal from the MEGA shall send a signal to the UNO which will change the state machine in the UNO to the current state of the monorail system.
FR_4	Output of I2C LCD Screen	The output of the LCD shall read the current state of the monorail system when the input signal is received from the MEGA.
FR_5	Communication of Bluetooth Module between UNO and MEGA	The UNO shall send serial data to the Bluetooth module when a control button is pressed. The Bluetooth module shall receive serial data from UNO and sends it to the MEGA through the TX pin of the Bluetooth Module (RX pin of MEGA).
FR_6	Bluetooth Module Pairing	The Bluetooth module shall connect to the UNO and successfully communicate with the Bluetooth module connected to the MEGA.

5.2 Performance Requirements

Table 6. Performance Requirements

ID	*Requirement*	*Description*
PR_1	Bluetooth Signal	The Bluetooth modules shall communicate for a minimum distance of 10 meters.
PR_2	LCD Screen Information	The LCD shall provide unambiguous information to the user. The LCD will not display character strings greater than 32. The LCD screen must display information with minimal delay to ensure accurate information is presented.
PR_3	Data Transmission	The UNO shall communicate with the MEGA within a time frame of 1-3 seconds to ensure fluid operations for the monorail system. The data being transmitted must not be corrupted or inaccurate.
PR_4	Code Style	To ensure that the data transmitted from the UNO is sent with minimal delay, the code implemented for the UNO shall be written for maximum efficiency and easy readability.

5.3 Interface Requirements

5.3.1 Overview

The subsystems that have been assigned to T4C1 requires many different forms of collaboration and communication with the other teams in the project. As T4C1 is working on the UNO, subsystems that require the functions of the UNO are directly interfaced with T4C1. For instance, T4C3 who is in charge of the MEGA is being passed information from the UNO meaning that T4C3 require to know what signals are being passed through. A list of all the interface requirements and the respective teams and sign off can be seen in Table 7.

5.3.2 Inclusion and Exclusions

Inclusions

T4C1 will provide the UNO and develop the code which will be implemented on the UNO within the ‘Control Box.’ This code will process the input from the buttons, provide output for the monorail and keep track of the monorail’s state. Additionally, T4C1 will provide the two Bluetooth modules and develop the code which will establish and maintain the connectivity between both modules. T4C1 will also provide the LCD screen and develop the code to display appropriate information on the screen. T4C1 will produce documentation detailing the subsystems T4C1 has been tasked.

Exclusions

The UNO will be housed in the ‘Control Box’ fitted with buttons and a LCD. The physical chassis for the ‘Control Box’ will be manufactured by T4S1. The wiring for the UNO, the buttons, the ‘Bluetooth module’, and the LCD will be done by T4M2. T4C3 will develop the code which will control the motors on the monorail and keep track of its state.

5.3.3 Interface Requirements and Sign-offs

Table 7. Interface Requirements and Sign-Offs

ID	*Requirement*	*Teams Involved*	*T4C1 Signoff*	*Interface Signoff*	*Date*
IR_1	T4C1 shall give T4C3 the integer values that will be sent when a button is pressed on the Control Box	T4C1 T4C3	Rajiv Mehta - 45433062	Tanner Schineller - 44339402	12/8/2020
IR_2	T4C1 shall give T4C3 the code for the Bluetooth module.	T4C1 T4C3	Rajiv Mehta - 45433062	Tanner Schineller - 44339402	12/8/2020
IR_3	T4C1 will receive character values when certain stations are reached when they are leaving to the next station from T4C3.	T4C1 T4C3	Rajiv Mehta - 45433062	Tanner Schineller - 44339402	12/8/2020
IR_4	T4C1 will receive a debugging template for the MEGA from T4C3.	T4C1 T4C3	Rajiv Mehta - 45433062	Tanner Schineller - 44339402	12/8/2020
IR_5	T4M2 will receive 15 pins on the Arduino Uno and which of those pins are being used by T4C1.	T4C1 T4M2	Rajiv Mehta - 45433062	Asaad Alshekhly- 45363579	12/8/2020

5.4 Constraints

Table 8. Constraints for the subsystem

Constraint	*Description*
Coding	In the implementation of the control box's state diagram, LCD screen, and Bluetooth connection, the process-oriented programming language, C language, must be used for all coding aspects.
Materials/Hardware	In establishing a connection between the Control Box and Monorail, Arduino based technology such as the UNO, and an Arduino compatible Bluetooth module must be used. The UNO has a memory limit of 32k bytes which can limit the functionality of the Control Box.
Time	The MVP must be completed by week 10 and the final product must be completed by week 13.
Collaboration	Due to the limitations of class hours during the week, collaboration with other teams within the overall project is highly constrained as the only time that teams can see each other face-to-face will be during that timeslot. This can cause delays in both testing and communication.

6 Conceptual Design

T4C1 has been assigned a task to build software for the 'Control Box'. The 'Control Box' is divided into three main components including:

- The Arduino UNO
- Bluetooth Module 4.0
- I2C LCD

These sub-systems work in the following order and is also showcased in Figure 2.

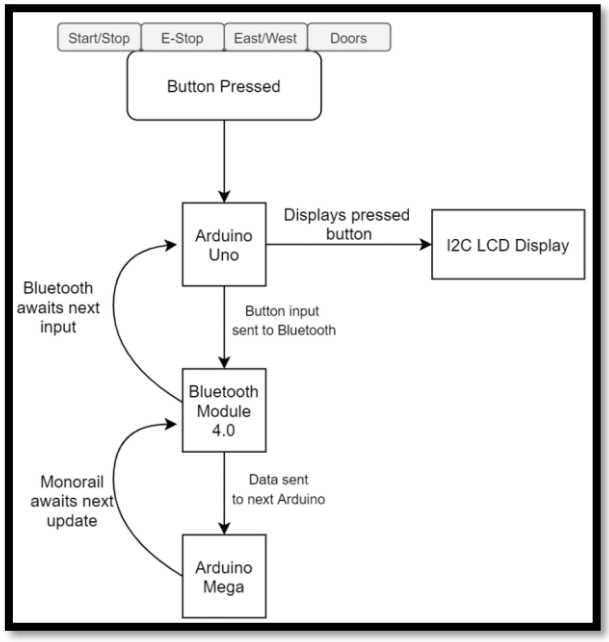


Figure 2. The conceptual design of the state machine

6.1 Conceptual Design for Arduino UNO

The UNO stores each output in an array so that when a button is pressed the code can search for that position in the array efficiently and send that output to the module. The UNO is the main component of the 'Control Box' and the three sub-systems are connected to the UNO according to their specific pins.

1. If any button is pressed, a signal corresponding the button pressed is sent to the UNO.
2. The UNO processes the information provided and sends a signal to the Bluetooth module that is further transmitted to the paired MEGA at the other end to T4C3.
3. The UNO receives the information back from the T4C3's MEGA about what state the tram is in and displays that information onto the LCD.

6.2 Conceptual for Bluetooth Module 4.0

Bluetooth 4.0 module requires no firmware or external components for functioning. Integers are used to store variables and use Boolean statements to update variables. The 'Bluetooth Module' receives the information from the UNO and sends that information through signals to the MEGA at the other end. A module consists of six pins connected to the Arduinos where Rx and Tx are responsible for receiving and transmitting the information. The main functions of the Bluetooth modules are:

- Transmitting to the corresponding Bluetooth module without error.
- Connect the corresponding pins according to the datasheet schematic.
- Develop code to transmit and receive information for T4C1.

6.3 Conceptual Design for I2C LCD

A '16x2 LCD' screen with an I2C interface is designated to be used for the project. The LCD is able to display 32 characters on its screen, showcased by white characters on a blue background. The information regarding the state of the Monorail is received from the MEGA and displayed onto the LCD. For example, if the state is going towards station A, the display should read "Approaching Station A."

7 Detailed Design

7.1 Overall Detailed Physical Design

The UNO is the main component in the Control Box which is connected to the four buttons, the LCD and the Bluetooth module. The four buttons include START/STOP, E-STOP, EAST/WEST and OPEN/CLOSE(doors). The Bluetooth module 4.0 connected to the UNO is paired with a MEGA and its connected Bluetooth module which is associated with the monorail. This connection sends and receives information in the form of characters and character strings. The Arduino displays the received information onto the LCD. The table below shows the main components of the Control Box.

Table 9. Main Physical Components in Control Box

Components	***Model***	***Quantity***
Microcontroller Board	Arduino Uno	1
Bluetooth Module	Arduino Compatible Bluetooth V4.0 BLE Module	2
Buttons	16mm Illuminated Pushbutton	4
Battery / Battery Pack	9V	1
Display	I2C 16x2 LCD	1

7.1.1 Pin Connections

Bluetooth:

- Arduino UNO D8 (RX) connects to UNO Bluetooth Module TX
- Arduino UNO D9 (TX) connects to UNO Bluetooth Module RX
- Arduino UNO GND connects to UNO Bluetooth Module GND
- Arduino UNO 5V connects to UNO Bluetooth Module VCC
- Arduino MEGA 19 (RX) connects to MEGA Bluetooth Module TX
- Arduino MEGA 18 (RX) connects to MEGA Bluetooth Module RX
- Arduino MEGA GND connects to MEGA Bluetooth Module GND
- Arduino MEGA 5V connects to MEGA Bluetooth Module VCC

LCD:

- Arduino UNO A0 connects to I2C LCD Serial Data Pin
- Arduino UNO A1 connects to I2C LCD Serial Clock Pin
- Arduino UNO GND connects to I2C LCD GND Pin
- Arduino UNO 5V connects to I2C LCD VCC Pin

Buttons:

- Arduino UNO D2 connects to E-Stop Button 1
- Arduino UNO D3 connects to East/West Button 2
- Arduino UNO D4 connects to Start/Stop Button 3
- Arduino UNO D5 connects to Doors Open/Close Button 4

7.2 Detailed Software Design

When the user presses a button on the physical Control Box, the code will start to execute due to it being interrupt-driven. When a button is pressed a low signal is sent to the Arduino UNO. The UNO first checks if the button has already been pressed within a time frame to see if the input should change the state of the monorail. If the signal is approved, then the pin number that correlates to the button being pressed is checked against its corresponding Boolean statement. The Boolean statements change the corresponding number located in the array 'stateOutput' from either 0 to 1 or 1 to 0 dependent on its current state. An example of this can be seen inline 134 – 146 within Appendix 8.1. This array is then converted into a string and is constantly outputted through the Bluetooth module to the Arduino MEGA until another button has been pressed.

For the Bluetooth implementation to work we are using the AltSoftSerial library. The AltSoftSerial library is a library that emulates an additional serial port, which allows us to communicate with another serial device. This is included as the UNO only has one serial port.

The Bluetooth Code can be split into two separate parts: the UNO code and the MEGA code.

- UNO Code: As stated earlier, the 'stateOutput' is a character array where each character refers to a Control Box button and can change from either a 0 to 1 or 1 to 0. 'stateOutput' is 7 characters long which contains a 'startMarker', 'endMarker', each button state, and a null character. An example of this can be seen inline 36 within Appendix 8.1. The UNO continuously sends the 'stateOutput' array to the connected Bluetooth Module. This will be done using the write() function. The write() function writes binary data to the serial port and the 'stateOutput' array will be sent as a series of bytes to the Bluetooth module connected to the Arduino MEGA.
- MEGA Code:First, the MEGA code has 4 global variables which are in direct relation to the Bluetooth Connection. This can be found inline 15-18 in Appendix 8.2. Within the void loop() function in Appendix 8.2, the Arduino MEGA has the function recordState().recordState() works by determining the start of a changed state transmitted by the UNO with a 'startMarker'. The function then adds the next incoming characters to the character array. recordState() determine when the state has finished sending through the use of a 'endCharacter'. The Boolean 'sentString' is then set to true. If 'sentString' is true then T4C3 is able to operate on the string we have provided to them.

The function recordState() and the corresponding variables can be found inline 57-85 in Appendix 8.2.

When the state of the Monorail changes, the MEGA changes the character that is continuously sent to its Bluetooth module to the UNO's Bluetooth module. When a character is received by the UNO, the code determines if the character is different from the current state of the monorail. The variable that stores the incoming character is 'inChar'. This can be found at line 20 in Appendix 8.3. If the received character is the same as the already stored character then no operations are made. If the character is found to be different then the code enters a switch statement. A switch statement allows the UNO to execute one block of code among many alternatives. Within the switch statement, there are 10 cases that the new character is checked against. When the character is identified with a switch case, that corresponding block of code is executed.

When the block of code is being executed the LCD will:

- Clear its screen.
- Print the top-line statement associated with that state.
- Set the LCD cursor to the bottom-line.
- Print the bottom-line statement associated with that state.
- Break out of the switch case.

This process is repeated until the monorail is turned off. This process can be seen in the state diagram below in figure 3.

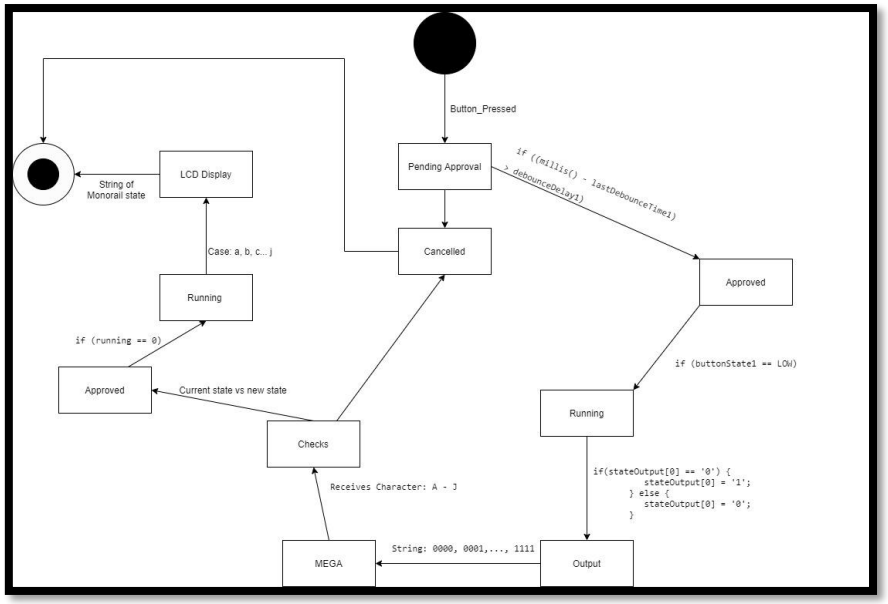


Figure 3. State Diagram of Control Box

7.3 Design Traceability

RequirementID	Status	Description
FR_01-FR_06	Complete	Functional requirements can be referred to access the information on what needs to be achieved before getting into the actual construction/coding of the interface.
PR_01-PR_04	Complete	Details the performance requirements that are to be met while coding the User Interface for the control box and how actually it should perform.
IR_1-IR_5	Complete	Interface requirements detail about the User Interface requirements and how the interface integrates with other parts of the elevator and what needs to be given from the interface to other parts.

8 Appendix

8.1 Buttons and Bluetooth Code for UNO

```
1
2   //  Pins - Need to Update
3   //  BT VCC to Arduino 5V out.
4   //  BT GND to GND
5   //  Arduino D8 (SS RX) - BT TX no need voltage divider
6   //  Arduino D9 (SS TX) - BT RX through a voltage divider (5v to 3.3v)
7   //
8   //  From the supplied ENGG2000 bluetoothUno code
9   //          -AltSoftSerial will be used instead of SoftwareSerial because
10  //           it allows for simultaneous data flows
11
12  // 2/09/20 - Changing from AltSoftSerial to SoftwareSerial to allow the RX and TX to be on
13  // 9/09/20 - Staying with AltSoftSerial for the meantime
14  /* ------------------------------------------------------------------------ */
15
16  #include <AltSoftSerial.h>
17  #include <stdio.h>
18  #include <stdlib.h>
19
20  /* Global Variables */
21
22  AltSoftSerial BTserial;
23  boolean NewLine = true;
24
25  /* Bluetooth Global Variable */
26
27  char megaResponse=' ';
28  char button1 = '0';
29  char button2 = '0';
30  char button3 = '0';
31  char button4 = '0';
32
33  char startMarker = '<';
34  char endMarker = '>';
35
36  // Need to test without null character
37  char stateOutput[7] = {startMarker, button1, button2, button3, button4, endMarker, '\0'};
38
39  /* Button Global Variables */
40
41  // declare and initialise button pins
42  int controlBut1;
43  int controlBut2;
44  int controlBut3;
45  int controlBut4;
46
47  // declare and initialise led pins
48  int ledPin1, ledPin2, ledPin3, ledPin4;
49
50  int ledState1 = LOW;
51  int ledState2 = LOW;
52  int ledState3 = LOW;
53  int ledState4 = LOW;
54
55  int buttonState1;
56  int buttonState2;
57  int buttonState3;
58  int buttonState4;
59  int lastButtonState1 = LOW;
60  int lastButtonState2 = LOW;
61  int lastButtonState3 = LOW;
```

```
int lastButtonState4 = LOW;

unsigned long lastDebounceTime1 = 0;  // the last time the output pin was toggled
unsigned long lastDebounceTime2 = 0;
unsigned long lastDebounceTime3 = 0;
unsigned long lastDebounceTime4 = 0;
unsigned long debounceDelay1 = 50;    // the debounce time; increase if the output flicker
unsigned long debounceDelay2 = 50;
unsigned long debounceDelay3= 50;
unsigned long debounceDelay4 = 50;

void setup() {

    // Arduino Uno Startup
    Serial.begin(9600);
    Serial.print("Sketch:   ");   Serial.println(__FILE__);
    Serial.print("Uploaded: ");   Serial.println(__DATE__);
    Serial.println(" ");

    // Bluetooth Module Startup
    BTserial.begin(9600);
    Serial.println("BTserial started at 9600");

    // State Machine Startup - Allen check
    pinMode(controlBut1, INPUT_PULLUP);
    pinMode(controlBut2, INPUT_PULLUP);
    pinMode(controlBut3, INPUT_PULLUP);
    pinMode(controlBut4, INPUT_PULLUP);

    //Debugging
    pinMode(ledPin1, OUTPUT);
    pinMode(ledPin2, OUTPUT);
    pinMode(ledPin3, OUTPUT);
    pinMode(ledPin4, OUTPUT);

    //Initial State which is LOW
    digitalWrite(ledPin1, ledState1);
    digitalWrite(ledPin2, ledState2);
    digitalWrite(ledPin3, ledState3);
    digitalWrite(ledPin4, ledState4);
}

void loop() {

// Read from the Bluetooth module and send to the Arduino Serial Monitor
    if (BTserial.available()) {
        megaResponse = BTserial.read();


        /* Used for Debugging - Response from Mega is shown onto plugged in computer.
           Will need to change or remove this to work with LCD */
        Serial.write(megaResponse);
    }

  int reading1 = digitalRead(controlBut1);
  int reading2 = digitalRead(controlBut2);
  int reading3 = digitalRead(controlBut3);
  int reading4 = digitalRead(controlBut4);

  if (reading1 != lastButtonState1) {
    lastDebounceTime1 = millis();
  }
  if (reading2 != lastButtonState2) {
    lastDebounceTime2 = millis();
  }
  if (reading3 != lastButtonState3) {
    lastDebounceTime3 = millis();
```

```
  }
  if (reading4 != lastButtonState4) {
    lastDebounceTime4 = millis();
  }

  if ((millis() - lastDebounceTime1) > debounceDelay1) {
    if (reading1 != buttonState1) {
      buttonState1 = reading1;

      Serial.print("Button 1 ");
      Serial.println(buttonState1);

      if (buttonState1 == LOW) {
        if(stateOutput[0] == '0') {
           stateOutput[0] = '1';
        } else {
           stateOutput[0] = '0';
        }
        // enter LCD Code

        // enter LED Code
        ledState1 = ! ledState1;
      }
    }
  }
  if ((millis() - lastDebounceTime2) > debounceDelay2) {
    if (reading2 != buttonState2){
      buttonState2 = reading2;

      Serial.print("Button 2 ");
      Serial.println(buttonState2);

      if (buttonState2 == LOW) {
        if(stateOutput[1] == '0') {
           stateOutput[1] = '1';
        } else {
           stateOutput[1] = '0';
        }

        // enter LCD Code

        // enter LED Code
        ledState2 = ! ledState2;
      }
    }
  }
  if ((millis() - lastDebounceTime3) > debounceDelay3) {
    if (reading3 != buttonState3){
      buttonState3 = reading3;

      Serial.print("Button 3 ");
      Serial.println(buttonState3);

      if (buttonState3 == LOW) {
        if(stateOutput[2] == '0') {
           stateOutput[2] = '1';
        } else {
           stateOutput[2] = '0';
        }

        // enter LCD Code

        // enter LED Code
        ledState3 = ! ledState3;
      }
    }
  }
```

```
196     if ((millis() - lastDebounceTime4) > debounceDelay4) {
197       if (reading4 != buttonState4){
198         buttonState4 = reading4;
199 
200         Serial.print("Button 4 ");
201         Serial.println(buttonState4);
202 
203         if (buttonState4 == LOW) {
204           if(stateOutput[3] == '0') {
205              stateOutput[3] = '1';
206           } else {
207              stateOutput[3] = '0';
208           }
209 
210           // enter LCD Code
211 
212           // enter LED Code
213           ledState4 = ! ledState4;
214         }
215       }
216     }
217 
218     digitalWrite(ledPin1, ledState1);
219     digitalWrite(ledPin2, ledState2);
220     digitalWrite(ledPin3, ledState3);
221     digitalWrite(ledPin4, ledState4);
222 
223     // Save the reading. Next time through the loop, it'll be the lastButtonState[i]:
224     lastButtonState1 = reading1;
225     lastButtonState2 = reading2;
226     lastButtonState3 = reading3;
227     lastButtonState4 = reading4;
228 
229     // Debugging for Bluetooth
230     if (NewLine) {
231        Serial.print("\r\n>");
232        NewLine = false;
233     }
234     Serial.write(stateOutput);
235     NewLine = true;
236 
237     // Send stateOutput 'String' to
238     BTserial.write(stateOutput);
239   }
```

8.2 Bluetooth Code for Mega

```
1   /*
2     BlueTooth Serial controlling a MEGA
3 
4    Receives from the main serial port, sends to the others.
5    Receives from serial port 1, sends to the main serial (Serial 0).
6    Serial 0 is used for debugging currently - Can change ports if need be in the future
7 
8    */
9 
10  boolean NewLine = true;  // we need to delay this one loop through the loop() routine
11 
12 
13  /* Bluetooth Global Variables */
14 
15  const byte maxDataLength = 6;
16  char receivedChars[7];
17  boolean sentString = false;
18 
```

```
void setup() {

  // Arduino Mega Startup
  Serial.begin(9600);
  Serial.print("Sketch:   ");   Serial.println(__FILE__);
  Serial.print("Uploaded: ");   Serial.println(__DATE__);
  Serial.println(" ");

  // Bluetooth Module will be connected to Serial1 PINS
  Serial1.begin(9600);
}

void loop() {

  recordState();

  //If receivedChars is a state it will be written to the Serial for debugging
  if(sentString) {
    Serial.write(receivedChars);
    sentString = false;

    //Debugging for Bluetooth
    if (NewLine) {
      Serial.print("\r\n> ");
      NewLine = false;
    }
    NewLine = true;
  }
}


// function recordState is inspired by the function recvWithStartEndMarkers by
 Robin2 of the Arduino forums
// See http://forum.arduino.cc/index.php?topic=288234.0 for further details

//Reads serial data and returns the content between a start and end marker.
void recordState() {
  static boolean recInProgress = false;
  static byte index = 0;
  char startMarker = '<';
  char endMarker = '>';
  char recordedChar;

  // Need to experiment with greater numbers, such as 4
  if(Serial1.available() > 0) {
    recordedChar = Serial1.read();

    if(recInProgress == true) {
      if(recordedChar != endMarker) {
        receivedChars[index] = recordedChar;
        index++;
        if(index > maxDataLength) {
          index = maxDataLength;
        }
      } else {
        receivedChars[index] = '\0'; // Terminate the String
        recInProgress = false;
        index = 0;
        sentString = true;
      }
    } else if (recordedChar == startMarker) {
      recInProgress = true;
    }
  }
}
```

8.3 LCD Code

```
#include <LiquidCrystal_PCF8574.h>
#include <Wire.h>
#include <AltSoftSerial.h>
AltSoftSerial BTserial;

LiquidCrystal_PCF8574 lcd(0x27);

#define GOING_A 'a'
#define ARRIVED_A 'b'
#define GOING_B 'c'
#define ARRIVED_B 'd'
#define GOING_C 'e'
#define ARRIVED_C 'f'
#define EMERGENCY_STOP 'g'
#define CHANGE_DIRECTION 'h'
#define DOORS_OPEN 'i'
#define DOORS_CLOSE 'j'

int running = -1;
char inChar;
boolean NewLine = true;



void setup() {
  int error;
  Serial.begin(9600);
  BTserial.begin(9600);
  while (!Serial);
    Wire.begin();
    Wire.beginTransmission(0x27);
    error = Wire.endTransmission();
    Serial.print("Error = ");
    Serial.print(error);

  if (error == 0) {
    Serial.println(": LCD found!");
    running = 0;
    lcd.begin(16, 2);
  } else {
    Serial.println(": LCD not found.");
  }
}


void loop() {
  lcd.setBacklight(255);
  if (BTserial.available()) {
    inChar = BTserial.read();
    Serial.write(inChar);
    }
  if (Serial.available()) {
    inChar = Serial.read();
    if (inChar!=10 & inChar!=13 ) {
        BTserial.write(inChar);
        }
    if (NewLine) {
        Serial.print("\r\n>");
        NewLine = false;
        }
        Serial.write(inChar);
        if (inChar==10) {
            NewLine = true;
        }
```

```
    }

  //  LCD OUTPUT  //
  //NEED TO ADD A CHECK FOR STATE!!!
  if (running == 0) {
    switch(inChar){
      case 'a':
        lcd.clear();
        lcd.print("Train going to ");
        lcd.setCursor(0,1);
        lcd.print("Station A!");
        break;
      case 'b':
        lcd.clear();
        lcd.print("The Train is now");
        lcd.setCursor(0,1);
        lcd.print("at station A!");
        break;
      case 'c':
        lcd.clear();
        lcd.print("Train going to ");
        lcd.setCursor(0,1);
        lcd.print("Station B!");
        break;
      case 'd':
        lcd.clear();
        lcd.print("The Train is now");
        lcd.setCursor(0,1);
        lcd.print("at station B!");
        break;
      case 'e':
        lcd.clear();
        lcd.print("Train going to ");
        lcd.setCursor(0,1);
        lcd.print("Station C!");
        break;
      case 'f':
        lcd.clear();
        lcd.print("The Train is now");
        lcd.setCursor(0,1);
        lcd.print("at station C!");
        break;
      case 'g':
        lcd.clear();
        lcd.setCursor(3, 0);
        lcd.print("EMERGENCY");
        lcd.setCursor(3, 1);
        lcd.print("STOPPING!");
        break;
      case 'h':
        lcd.clear();
        lcd.print("    Changing");
        lcd.setCursor(0,1);
        lcd.print("   Directions!");
        break;
      case 'i':
        lcd.clear();
        lcd.setCursor(0,0);
        lcd.print("Doors Opening!");
        lcd.setCursor(0,1);
        lcd.print("Have a nice day!");
        break;
      case 'j':
        lcd.clear();
        lcd.setCursor(0,0);
        lcd.print("Doors Closing!");
        lcd.setCursor(0,1);
```

```
132           lcd.print("Stand Clear!");
133           break;
134         default:
135           break;
136       }
137     }
138   }
```

9 References

[1] Wiltronics, “Bluetooth 4.0 Module HM-10 CC2541 Arduino Compatible,” Stem Warehouse, Unknown Unknown Unknown. [Online]. Available: https://www.wiltronics.com.au/product/10354/arduino-bluetooth-4-0-module-hm-10-cc2541/. [Accessed 15 August 2020].