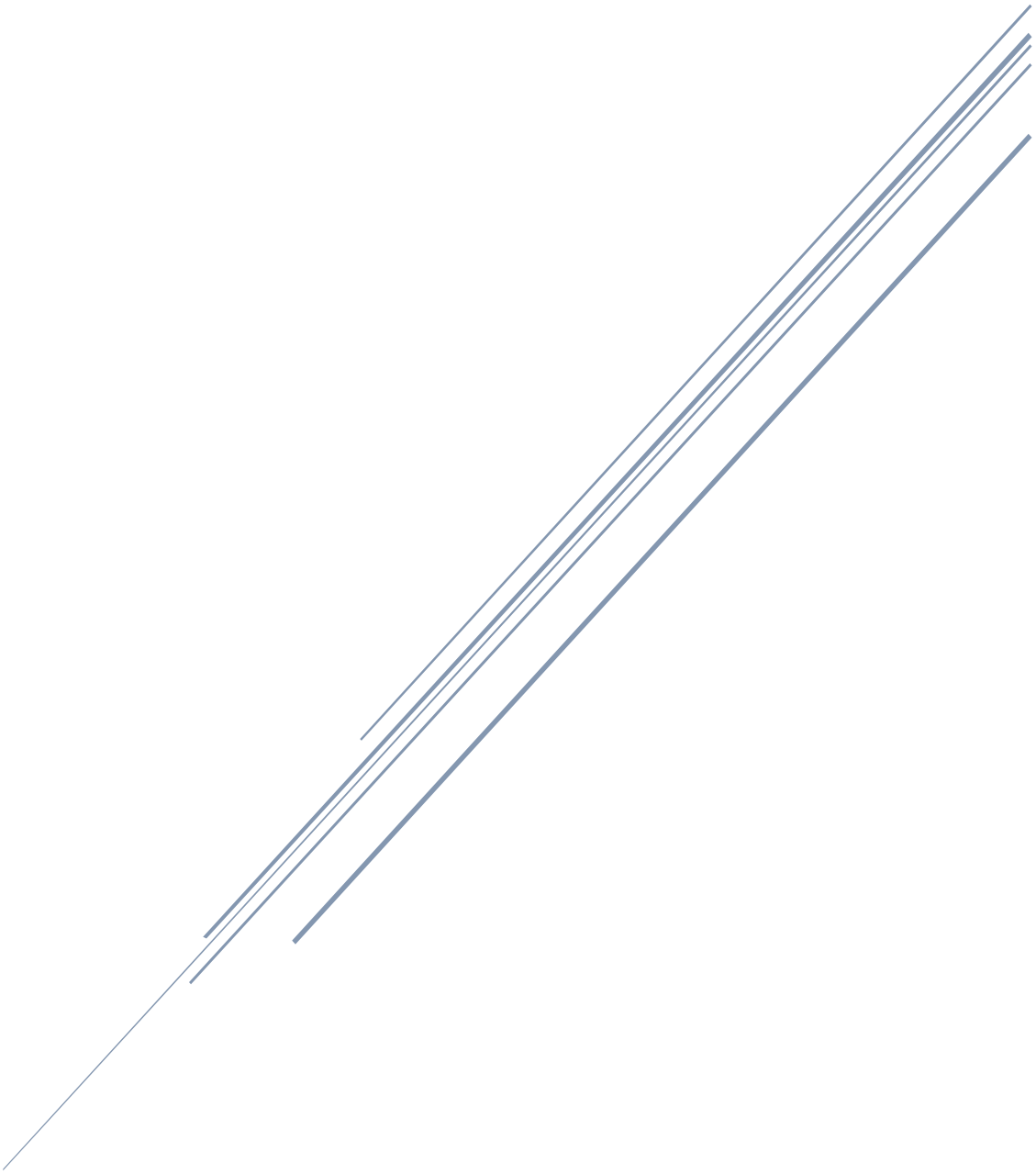
REQUIREMENTS DOCUMENT



Team: L7-C20

Table of Contents

1 Division of Labour

Members	Student Identification	Cohort	Sections	Overall %
Jordan Bertasso	45391351	ENGG200	Introduction, Scope, Terms, Testing	16.67%
Rajiv Mehta	45433062	ENGG200	Problem Definition, Detail Design, Design Review, Conclusion	16.67%
Sahithi Kodali	45712050	ENGG300	Functional Requirements, Design Traceability	16.67%
Phan Hoai Nam	44183534	ENGG300	Interface Requirements, Design Traceability, Maintenance procedure	16.67%
Ye Xu	44480814	ENG300	Performance Requirements, Constraints, Detail Design	16.67%
Abdul Wasay	45823758	ENG600	Design, Conceptual Design	16.67%

2 Introduction

2.1 Terms

Abbreviation	Definition
Comms	Communication
FR	Functional requirement
IP/OP	Input or Output
IDE	Integrated development environment
IR	Interface requirement
LED	Light-emitting diode
MVP	Minimum viable product
UI	User-interface

2.2 Scope

This semester ENGG200/300/600 students have been tasked with designing and building a functional model elevator. The final product is required to meet the specifications provided by the unit staff. L7-C20, User Interface and Communications, is responsible for providing a software user interface solution that meets the provided requirements.

This document establishes the essential requirements of the "User Interface and Communications" solution for the Elevator Challenge. It is directly applicable to our MVP and may be updated in the future in order to include the requirements for additional features. Additionally, this document provides the details of the customer's problem statement and each requirement, as well as any potential error conditions and limitations.

This document is essential to the successful delivery of a solution. It enables our team and other teams to communicate effectively and provides clear objectives that need to be accomplished in order to provide a quality solution, by definition. This is a living document and will be updated throughout the project to reflect the successes, changes in scope, and general progress. This document can only be changed by mutual consent of customer and solution provider.

Due to the shortage of Communications students in our practical class, L7-C20 team is responsible for providing this solution to three different elevators, 7, 8 and 11 and therefore, this document will be interfacing with three different elevator shafts.

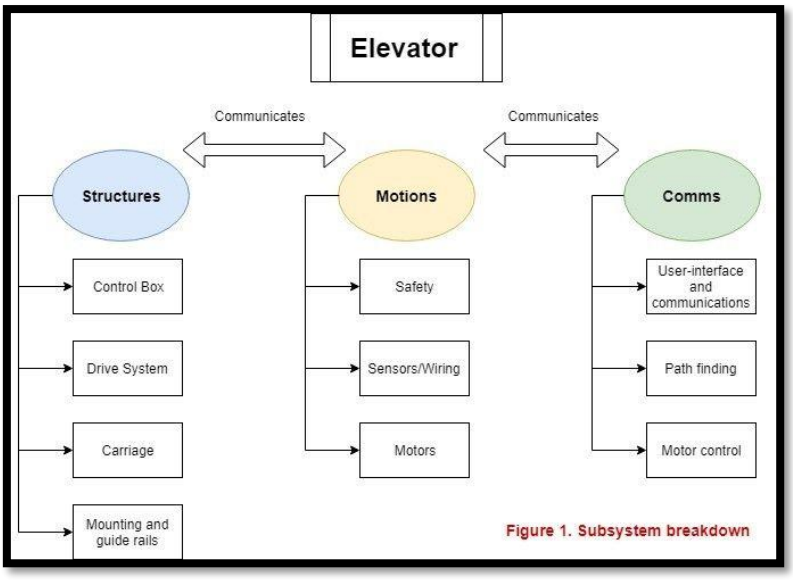
3 Problem Definition

The client has tasked the class to create three working model elevators cars. The elevator cars have to work within certain restrictions, for example, having the carriage fit inside the dimensions of the provided shaft and having the carriage having to be able to lift between 5-10 kilograms. This means that the class will be split up into smaller teams to work on smaller subsystems for each elevator such as a team working on constructing the carriage and a team working on the pulley system.

L7-C20 has been tasked to design and construct code for the control box for all three elevator shafts. Specifically, the team is required to make an output to the state machine for when a specific button is pressed for example when the first-floor button is pressed the code should output a character to the state machine.

3.1 Subsystems

The subsystems for the elevator challenge are split into three main categories as seen in figure 1. The structural teams are tasked to design and build aspects such as the control box and carriage as well ensuring that the elevator carriage moves smoothly up and down the shaft through its drive system and its mounting and guide rails.



The motions team are tasked to handle aspects such as the safety of the elevator by implementing safety features such as a safety stop. The teams also handle wiring and applying sensors to the elevators so that the carriage moves to the correct location when a certain button is pushed.

The comms teams are in charge of designing and implementing code to make the elevator function. The pathfinding team takes inputs from the control box and in turn, sends a signal to the motor control team to tell the motors where the carriage should go next, while the motor control team programs how much the carriage should move to each location.

As L7-C20 is tasked with the software for the control box, they have to design and implement code so that when a button is pressed by the user, a signal is outputted from the Arduino Uno to the Arduino Mega that corresponds to the button pressed. This means that L7-C20 will have to interface directly with all the other Comms teams as state-machine/pathfinding teams need to know what kind of signal they are receiving and what button it represents and also with the motor control team as they need to know which pins on the Arduino Uno are outputting to the Arduino Mega. L7-C20 will also have to interface with the motions teams who are designing and wiring up the physical control box so that they are informed on aspects such as the number of buttons required and which pins are representing which buttons.

3.2 Assumptions

- All buttons on the control box exist for the MVP - Even though, the MVP requires two buttons to be working on the control box (one to move the elevator one floor up/down) it's assumed that we have all buttons in the controller to avoid having to create another controller for the post MVP.
- Output through the serial pin in the Arduino - The output of which button will be pressed will be a single character sent through the serial pin.

4 Requirements

4.1 Functional Requirements

4.1.1 Description

A user interface (with the help of Arduino), which is required to be programmed in such a way that each button specified in the control box should perform its designated operation only.

There are 7 buttons in total:

1.) 5 Floor buttons

2.) Open/Close door button(Toggle button)

3.) An Emergency button

REQ_ID	Function	Requirement	Description
FR_01_IP	Digital input received from button presses. (7 buttons)	A Control Box button shall be pressed and a digital signal from the corresponding pin shall be received indicating the button was pressed.	When a button on the control box is pressed, the digital pin will be set to LOW, therefore signalling the button was pressed. The Arduino Uno will acknowledge this change internally and note the time that the button was pressed.
FR_01_OP	Serial output sent due to button presses. (7 buttons)	If a button is pressed, an integer ranging from '1' through to '7', representing the button pressed, shall be sent over the serial connection.	The Integer will be sent over a wired serial connection which corresponds to the button pressed. The mapping between integers and functions is communicated with the team that is receiving the output. The algorithm waits for 5 seconds between each particular button press before sending out the same integer again.

Table 1: Digital input and Serial output functional requirements

4.2 Performance Requirements

The user interface will conform to all moving and safety requests, which also covers integration and compatibility with pathfinding, sensor and wiring systems.

REQ_ID	Function	Description
PR_01	Data transmission	The conversion and response time of data inputs sourced from button press shall be achieved no more than 3 - 5 seconds. This ensures timely communication and smooth operation between the user interface and the elevator.
PR_02	LED	After pressing a button, the relevant LED shall be lightened within 3 seconds to signify the customer request has been received.
PR_03	Emergency stop response	The elevator shall be stopped within 3 seconds when emergency stop button is pressed.
PR_04	Door toggle response	The door toggle function shall trigger within 5 seconds and open/close within 10 seconds.
PR_05	Floor response	After pressing a floor button, the corresponding operation shall trigger within 5 seconds and finish within 15 seconds.

Table 2: Detailed Performance Description

5 Interface Requirements

5.1 Overview

The User Interface and Communications team are in charge of receiving and processing user inputs as well as transforming to an appropriate data type and transmitting the data via a secure and stable communication channel. As our team is responsible for creating the prototype for all the three lifts namely lift 7, lift 8 and lift 10, we have come to an agreement that it is the best for all three teams to build the control box and establish a communication channel with the same structure in order to reduce the completion time, reduce the complexity when creating multiple models and improve the overall quality of the system. Briefly, all the control box will have the same number of buttons with the same purpose, each button will be connected to the same serial pin on all board and the communication will be wired connection.

To reduce the complexity and simplify the integration phase between multiple departments, the best method to finish the MVP prototype in time with the schedule is: 1 structure team is assigned to build 3 copies of the physical control box for all 3 lifts and 1 motion team is responsible for designing the wiring diagram and process all the necessary wires appropriately to the data mapping agreement. Hence, the UI/Communications team interface will closely cooperate with these following teams: 1 structure team, 1 motion team and 3 different communication teams from 3 different lifts.

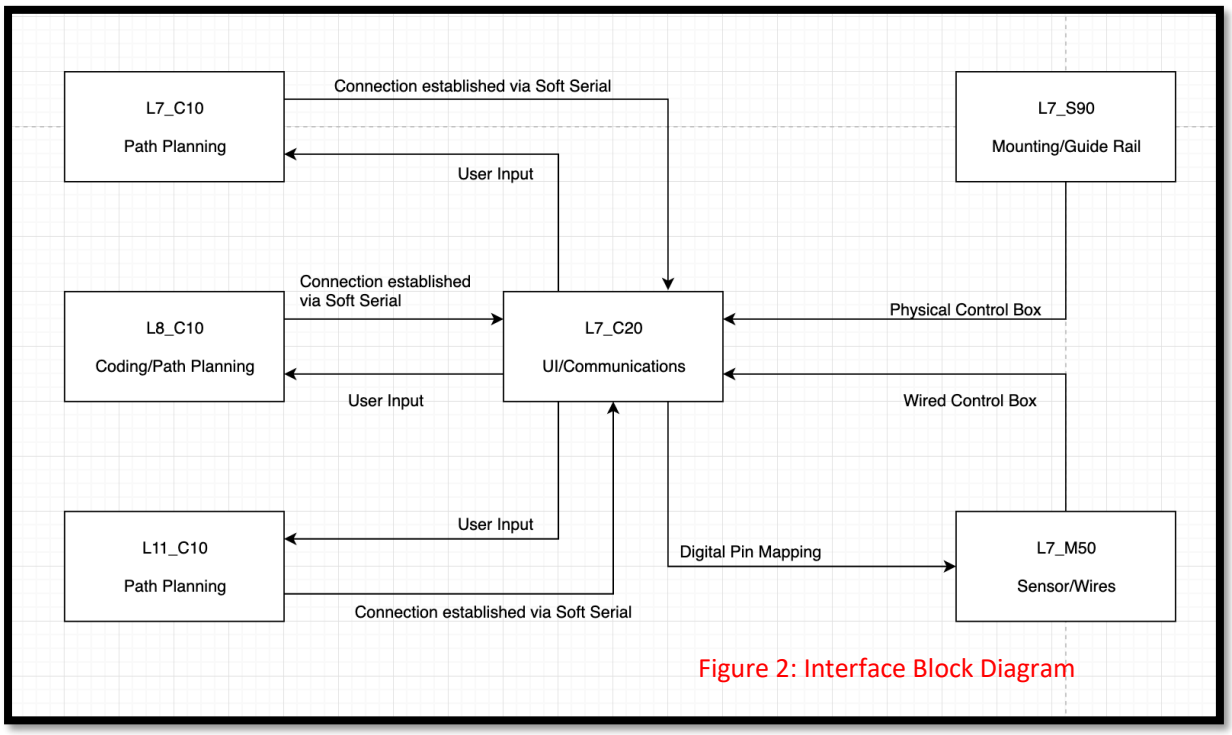


Figure 2: Interface Block Diagram

ID	Subsystem	Requirements	Description
IR_01	Path Planning (L7_C10) Coding/Path Planning (L8_C10) Path Planning (L11_C10)	UI/Communications (L7_C0) shall establish an Arduino connection with SoftSerial - pin 12 and 13 (RX and TX)	**Arduino Communication**: The interface shall provide a connection between Arduino Uno and Arduino Mega. This will be achieved by connecting pin 12 on the Uno to pin RX on the Mega
IR_02	Mounting/Guide Rail (L7_S90)	Mounting/Guide Rail (L7_S90) shall provide the physical control box	**Physical Control Box**: The interface shall provide a physical control box with 7 buttons
IR_03	Sensor/Wires (L7_M50)	Sensor/Wires (L7_M50) shall wire the Floor 1 Button - Pin: Digital 2 - Active Low	**Floor 1 Wiring:** The interface shall indicate the location of the elevator (floor number). This will be achieved by wiring the digital pin 2 to the Active Low on the Breadboard. This will set the button_state[0] to 0
IR_04	Sensor/Wires (L7_M50)	Sensor/Wires (L7_M50) shall wire the Floor 2 Button - Pin: Digital 3 - Active Low	**Floor 2 Wiring:** The interface shall indicate the location of the elevator (floor number). This will be achieved by wiring the digital pin 3 to the Active Low on the Breadboard. This will set the button_state[1] to 0
IR_05	Sensor/Wires (L7_M50)	Sensor/Wires (L7_M50) shall wire the Floor 3 Button - Pin: Digital 4 - Active Low	**Floor 3 Wiring:** The interface shall indicate the location of the elevator (floor number). This will be achieved by wiring the digital pin 4 to the Active Low on the Breadboard. This will set the button_state[2] to 0
IR_06	Sensor/Wires (L7_M50)	Sensor/Wires (L7_M50) shall wire the Floor 4 Button - Pin: Digital 5 - Active Low	**Floor 4 Wiring:** The interface shall indicate the location of the elevator (floor number). This will be achieved by wiring the digital pin 5 to the Active Low on the Breadboard. This will set the button_state[3] to 0
IR_07	Sensor/Wires (L7_M50)	Sensor/Wires (L7_M50) shall wire the Floor 5 Button - Pin: Digital 6 - Active Low	**Floor 5 Wiring:** The interface shall indicate the location of the elevator (floor number). This will be achieved by wiring the digital pin 6 to the Active Low on the Breadboard. This will set the button_state[4] to 0
IR_08	Sensor/Wires (L7_M50)	Sensor/Wires (L7_M50) shall wire the Emergency Stop Button - Pin: Digital 8 - Active Low	**Emergency Stop Wiring:** The interface shall indicate the state of the elevator (whether it is in an emergency or not). This will be achieved by wiring the digital pin 8 to the Active Low on the Breadboard. This will set the button_state[6]to 0
IR_09	Sensor/Wires (L7_M50)	Sensor/Wires (L7_M50) shall wire the Door Toggle Button - Pin: Digital 7 - Active Low	**Door Wiring:** The interface shall toggle the state of the elevator door from open to close and vice versa. This will be achieved by wiring the digital pin 7 to the Active Low on the Breadboard. This will set the button_state[5] to 0
IR_10	Sensor/Wires (L7_M50)	Sensor/Wires (L7_M50) shall provide the control box with the output (digital pin) that L7_C20 shall take that output as the input	**Control Box User Input:** The interface shall provide the indicator for the user's input to indicate the expected floor for the current user. This will be achieved by wiring the buttons to the correct digital pin number. This will give the data to the communication team about the user's input upon user event.
IR_11	Path Planning (L7_C10) Coding/Path Planning (L8_C10) Path Planning (L11_C10)	UI/Communications (L7_C0) shall send integer number 2 to the Arduino Mega if the signal is Active High	**Floor 1 Data Transfer:** This interface shall provide the indicator upon the user's input event to indicate which floor button being pressed. This will be achieved by setting the digital pin 2 to Active Low, thus, changing the button_state[0] value to 1. When the button_state[0] is 1, the floor number will be sent out as an integer 1 as an

			output. This will notify the communication about the user's input and send the signal to the path planning.
IR_12	Path Planning (L7_C10) Coding/Path Planning (L8_C10) Path Planning (L11_C10)	UI/Communications (L7_C0) shall send integer number 3 to the Arduino Mega if the signal is Active High	**Floor 2 Data Transfer:** This interface shall provide the indicator upon the user's input event to indicate which floor button being pressed. This will be achieved by setting the digital pin 3 to Active Low, thus, changing the button_state[1] value to 1. When the button_state[1] is 1, the floor number will be sent out as an integer 2 as an output. This will notify the communication about the user's input and send the signal to the path planning.
IR_13	Path Planning (L7_C10) Coding/Path Planning (L8_C10) Path Planning (L11_C10)	UI/Communications (L7_C0) shall send integer number 4 to the Arduino Mega if the signal is Active High	**Floor 3 Data Transfer:** This interface shall provide the indicator upon the user's input event to indicate which floor button being pressed. This will be achieved by setting the digital pin 4 to Active Low, thus, changing the button_state[2] value to 1. When the button_state[2] is 1, the floor number will be sent out as an integer 3 as an output. This will notify the communication about the user's input and send the signal to the path planning.
IR_14	Path Planning (L7_C10) Coding/Path Planning (L8_C10) Path Planning (L11_C10)	UI/Communications (L7_C0) shall send integer number 5 to the Arduino Mega if the signal is Active High	**Floor 4 Data Transfer:** This interface shall provide the indicator upon the user's input event to indicate which floor button being pressed. This will be achieved by setting the digital pin 5 to Active Low, thus, changing the button_state[3] value to 1. When the button_state[3] is 1, the floor number will be sent out as an integer 4 as an output. This will notify the communication about the user's input and send the signal to the path planning.
IR_15	Path Planning (L7_C10) Coding/Path Planning (L8_C10) Path Planning (L11_C10)	UI/Communications (L7_C0) shall send integer number 6 to the Arduino Mega if the signal is Active High	**Floor 5 Data Transfer:** This interface shall provide the indicator upon the user's input event to indicate which floor button being pressed. This will be achieved by setting the digital pin 6 to Active Low, thus, changing the button_state[4] value to 1. When the button_state[4] is 1, the floor number will be sent out as an integer 5 as an output. This will notify the communication about the user's input and send the signal to the path planning.
IR_16	Path Planning (L7_C10) Coding/Path Planning (L8_C10) Path Planning (L11_C10)	UI/Communications (L7_C0) shall send integer number 8 to the Arduino Mega if the signal is Active High	**Emergency Stop Data Transfer:** This interface shall provide the indicator upon the user's input event to indicate which floor button being pressed. This will be achieved by setting the digital pin 8 to Active Low, thus, changing the button_state[6] value to 1. When the button_state[6] is 1, the floor number will be sent out as an integer 7 as an output. This will notify the communication about the user's input and send the signal to the path planning.
IR_17	Path Planning (L7_C10) Coding/Path Planning (L8_C10) Path Planning (L11_C10)	UI/Communications (L7_C0) shall send integer number 7 to the Arduino Mega if the signal is Active High	**Door Data Transfer:** This interface shall provide the indicator upon the user's input event to indicate which floor button being pressed. This will be achieved by setting the digital pin 7 to Active Low, thus, changing the button_state[5] value to 1. When the button_state[5] is 1, the floor number will be sent out as an integer 6 as an output. This will notify the communication about the user's input and send the signal to the path planning.

Table 3: Interface Requirements

Group Number	Member Name	Student Identification	L7-C20 Group Member	Student Identification
L07_C10	Weisong Zeng	44481799	Rajiv Mehta	45433062
L08_C10	Matthew Smith	44950764	Jordan Bertasso	45391351
L11_C10	Adrian Kane	42117232	Ye Xu	44480814
L07_M50	Luaba Mashiat Ali	45681287	Jordan Bertasso	45391351
L07_S90	Daniel Giezekamp	44913478	Sahithi Kodali	45712050

Table 4: Interface Signoffs

5.2 Constraints

1. Coding: Since the user interface uses Arduino based technologies, the process-oriented programming language, C language, must be used in all of the designs to transmit data from the user interface.
2. Material: To transmit user commands to data, Arduino breadboards, primarily that of the UNO and MEGA boards, should be used to realize the algorithm for each button function.
3. Time restriction: The minimum variable product must be completed by 09/09/2019, the final product must be completed by 04/11/2019.
4. Function: There are 3 specific functions we must realize by designed buttons: the floor function for reaching certain floor, the door switch function for controlling the opening and closing of the door, the emergency stop function for mandatory elevator suspension.

6 Design

6.1 Conceptual Design

L7-C20 is responsible for designing and implementing software for the operation of elevator numbers seven, eight and eleven. By pressing a button on the control box, L7-C20 gives the path planning team a signal to show which button has been pressed. There are seven buttons with five buttons to reach each floor, one button for emergency and one button for the door toggling. Buttons are attached to the Arduino Uno board's digital pins. The digital pins two, three, four, five and six are connected to the floor buttons one, two, three, four and five. The Arduino board's digital pin eight is assigned to the emergency stop button and the door toggle button is assigned to digital pin seven. Because the user interface uses Arduino based technology, C language shall be used to model the software and to transfer user commands to information. As shown in figure 3, a control box is attached to the Arduino Uno board. The control box system shall transmit instructions to Arduino Uno when a button is pressed. Using serial communication, the code shall be used by Arduino Uno to perform the necessary task and output the corresponding number via serial pins, and the output will be sent to the Arduino Mega.

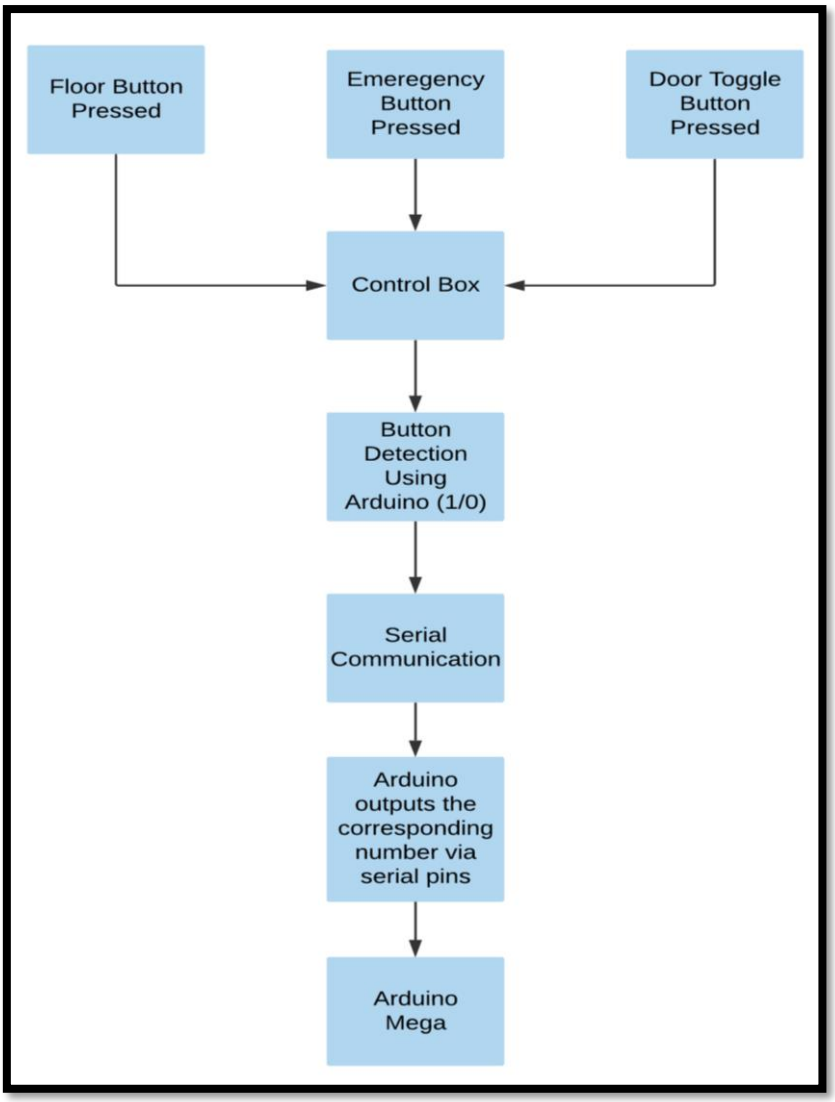


Figure 3: Conceptual design breakdown

6.1.1 Subsystems:

Each floor button being pressed:

When a floor button is pressed the input shall be provided by the program as a number. The code shall allow the elevator to go up or down to the appropriate floor when pressing a button that represents any floor. The feedback will be provided by an LED light display and the elevator will be ready to move to the appropriate level. For example, if floor one button is pressed then the number 1 will be received at the output, this is because the button is connected to digital pin number two which is by default set to zero so when the button is pressed the button state is changed from 0 to 1. The MVP for this is to have two buttons in use to push the elevator up one floor when the buttons are pushed.

Door toggle button being pressed:

The code shall be designed to open and close the elevator's doors. By pressing button number seven, the button state is changed from zero to one and the device receives the input to open/close the doors in a numerical format. A led shall signify that the input is received and that the doors are ready to open/close. This will be designed to have a ' toggle ' feature so that the door will be fully opened and closed when the button is pressed instead of a device that will be activated when the button is holding down the door.

Emergency button being pressed:

When the emergency button connected to pin number eight is pressed then the button state is changed from zero to one and the input is received by the interface to alarm everyone regarding the emergency and the elevator should stop in place. In the case of an overweight or dangerous situation, for example, pressing the emergency stop button requires another single-digit value to be transferred. As a result, the elevator goes into an unusable stop state. As this is not a part of the MVP this will be implemented later in the project.

Communication to the Arduino Mega:

This shall be achieved by connecting pin number twelve on the Uno to pin RX on the Mega. So, as shown in figure 1 the path planning team and their systems shall be able to understand which button was pressed and what action shall take place next. The MVP will start with a wired connection to the control box, but later in the project, we shall move on to wireless communications using Bluetooth.

6.2 Detailed Design

The final user interface model we selected is designed consisting of 5-floor buttons, 1 door toggle button and 1 emergency stop button since it requires the least moving components and codes to realize MVP function. The following section details relevant specifications of different module design that optimized our design patterns to the greatest extent.

6.2.1 Detailed Physical Design

ID	Object	Function
L_ID_01 - L_ID_05	5 LED Indicators for floors	Lighting indicates that the specified floor requirement has been accepted by the system.
L_ID_06	LED Indicator for door toggle	Lighting indicates that the command of opening or closing the door has been accepted by the system.
L_ID_07	LED Indicator for emergency stop	Lighting indicates that the stopping requirement has been accepted by the system.
B_ID_01 - B_ID_05	5 Floor buttons	Button press indicates the physical command has been transferred to the digital input to indicate the expected floor.
B_ID_06	Door toggle button	Button press indicates the physical command has been transferred to the digital input to indicate door open/close.
B_ID_07	Emergency stop button	Button press indicates the physical command has been transferred to the digital input to indicate emergency stop.

Table 5: Detailed physical design

6.2.1.1 Detailed Process Design

When the customer pressed a button on the control box, the physical command shall be transmitted to digital input since buttons are connected with corresponding pins of UNO board. The corresponding LED indicator lightens and command transmission will be achieved by setting digital pins to Active Low, thus, changing the button_state[?] value to 1 (Each button has an initial state value set to 0). When the button_state[?] is 1, the floor number will be sent out as an integer as an output. Pathfinding system start working, which gets elevator move. In the final stage, the elevator finishes the designated command and button state value shall be reset to 0. A general process diagram is shown in figure 4 and an example is shown in figure 5.

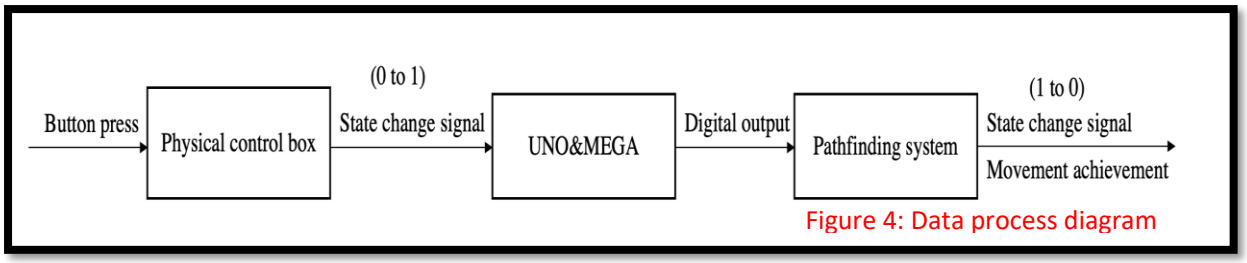


Figure 4: Data process diagram

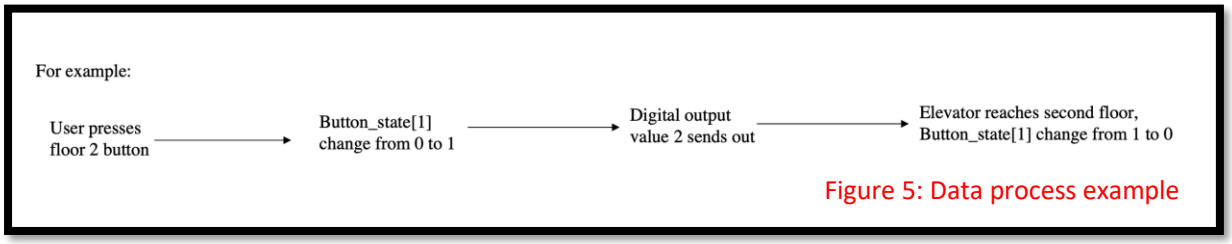


Figure 5: Data process example

Pressed button	Connected pin of UNO board	Corresponding button state	Corresponding digital output value
Floor 1 button	pin 2	button_state[0]	1
Floor 2 button	pin 3	button_state[1]	2
Floor 3 button	pin 4	button_state[2]	3
Floor 4 button	pin 5	button_state[3]	4
Floor 5 button	pin 6	button_state[4]	5
Door toggle button	pin 7	button_state[5]	6
Emergency stop button	pin 8	button_state[6]	7

Table 6: Digital values for buttons

6.2.2 Detailed software design

When the user presses a button, the code will start to execute as it is interrupt driven. The code first checks whether or not the button press is approved by checking if the button had been already previously pressed within the restricted time frame. It does this by first storing when the button was pushed and determining whether the time passed is smaller/larger than the button-ignore-time. This will then either push the state into approved or cancelled. If cancelled due to not enough time being passed, then the code will stop running and no signal will be sent to the Arduino Mega.

If this does not occur then the code will move into the approved state and begin to execute the code. It starts by running a loop to see which pin has been set too high as seen in line 64 of the code (found in the appendix). In this loop, a Boolean statement (line 68) is used to check which pin has been set too high. When this condition is met, the code moves to an output state and in turn, sends a character signal to the Arduino Mega corresponding to which button has been pressed. The code then moves to its end state and it becomes ready for another button to be pushed by the user.

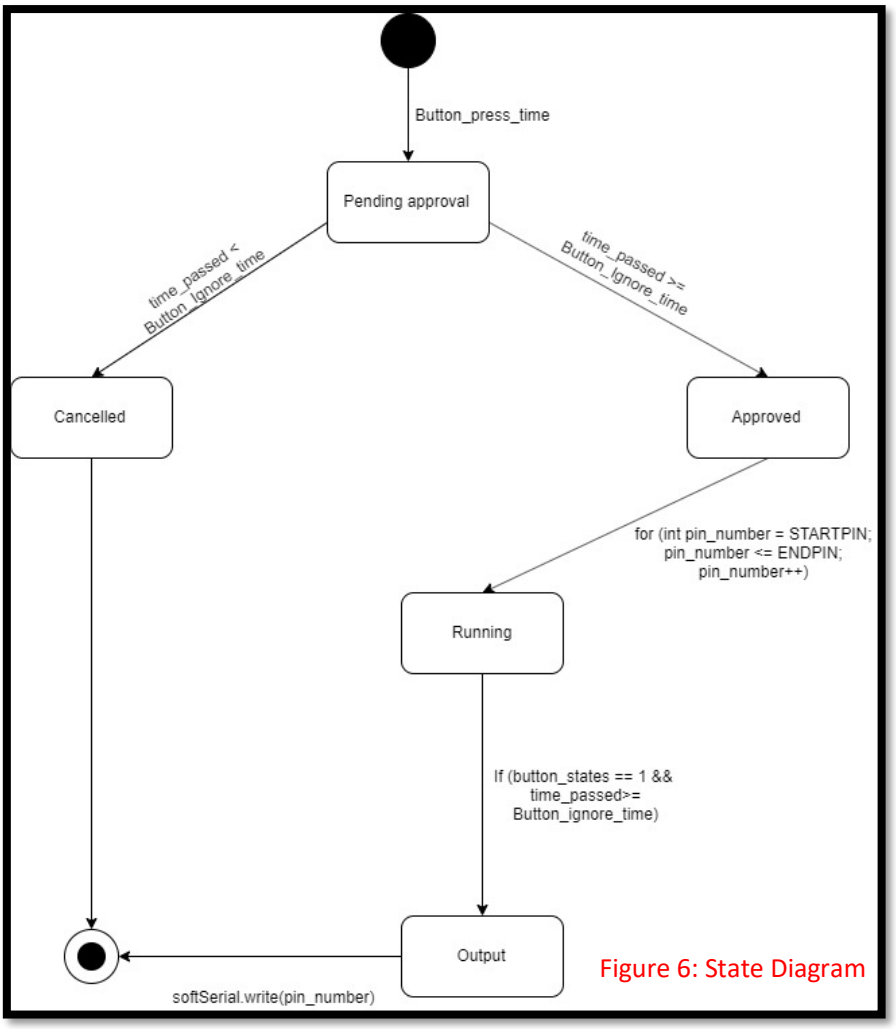


Figure 6: State Diagram

6.3 Design Traceability

Requirement ID	Status	Description
FR_01_IP - FR_01_OP	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_05	Complete	Details the performance requirements that are to be met while coding the User Interface for control box and how actually it should perform.
IR_01 - IR_15	Complete	Interface requirements detail about the User Interface requirements and how the interface integrate with other parts of elevator and what needs to be given from the interface to other parts.

Table 7: Design Traceability

7 Maintenance Procedure

7.1 Maintenance Requirement

The following categories are required to perform a complete and correct system-wide maintenance as well as to deliver precise maintenance on a specific unit part of the system:

- Hardware:
  - Arduino Uno: Perfect condition – no errors or faulty parts.
  - Breadboard: Perfect condition – no errors or faulty parts.
  - Resistor: Perfect condition
  - Connection wires: Perfect condition
  - Buttons: Perfect condition
  - Laptop: Working condition
- Software:
  - Visual Studio Code: necessary IDE to perform code maintenance
  - Arduino IDE: necessary IDE to perform Arduino maintenance
  - Bitbucket: necessary to share code between team members
- Personal:
  - C programming language: required
  - Arduino knowledge: required

7.2 Documentation procedure

1. Take note of the current system part that is under maintenance
2. Take note of the modification (if any) to the current system part
3. Comment the modification part in the Arduino code
4. Repeat from step 2 if there is any part needed for further maintenance
5. Write a thorough report of the maintenance
6. Inform other members of the existing report

7.3 System maintenance procedure

There are some conventions worth mentioning prior to the system-wide maintenance procedure:

- Terms, definitions, abbreviations are defined in the A7 requirement section
- Newly added comment is notified by the //README in the code section
- Newly added variables should be globally declared with appropriate comment
- Newly added functions should be declared at the bottom with appropriate comment
- Any modification should be correctly reported by the documentation procedure

7.3.1 Maintenance procedure

7.3.1.1 Setting up the system

1. Fork the Bitbucket for the maintenance process
2. Setting up the laptop/pc with compatible Arduino IDE version
3. Setting up the Arduino with the breadboard as seen in Appendix A
4. Connect the Arduino Uno to the laptop/PC via cable port
5. Wait for the LED on the Arduino to turn green
6. Turn on the Arduino IDE on the laptop
7. Check the correct connection port for the Arduino Uno
8. Check the correct board type for the Arduino Uno
9. Click verify to confirm the connection

7.3.1.2 To perform periodic maintenance for the core functional process:

10. Turn on Serial Monitor on the Arduino IDE
11. Press the left button on the breadboard
12. Examine the “button_state[0]” value on the monitor: it should be 1, others button_state[] should be 0
13. Press the right button on the breadboard
14. Examine the “button_state[2]” value on the monitor: it should be 1, others button_state[] should be 0
15. Perform any modification if necessary with comments
16. Write the analysis to the report
17. Press reset on the Arduino Uno

7.3.1.3 FR_01_IP and FR_01_OP:

10. Turn on Serial Monitor on the Arduino IDE
11. Wire the cable from digital pin 7 to the left button
12. Wire the cable from digital pin 8 to the right button
13. Wire the cable from digital pin 2 to the active low
14. Wire the cable from digital pin 4 to the active low
15. Press the left button on the breadboard
16. Examine the “button_state[5]” value on the monitor: it should be 1, others button_state[] should be 0
17. Press the right button on the breadboard
18. Examine the “button_state[6]” value on the monitor: it should be 1, others button_state[] should be 0
19. Perform any modification if necessary with comments
20. Write the analysis to the report
21. Press reset on the Arduino Uno

7.3.1.4 To perform maintenance for testing purpose:

10. Turn on Serial Monitor on the Arduino IDE
11. Follow the testing instructions which can be found on the A8 testing section
12. Perform any modification if necessary with comments
13. Write the analysis to the report

7.3.1.5 Finish the maintenance:

- Verify the code to check the integrity
- Flush the code to the Arduino Uno
- Unplug the cable connected the Arduino Uno to the laptop/pc
- Write report appropriate to the documentation procedure
- Include the report in the Bitbucket folder
- Commit any changes to the Bitbucket
- Push any changes to the Bitbucket
- Exit Arduino IDE

7.4 Software Unit Maintenance Procedure

Arduino Uno Board: The core software unit that contains the Arduino code for UI communications between Arduino Uno and Arduino Mega

Requirement: The Arduino Uno Board should be in perfect condition with no missing piece, no errors and no failure parts. When connected to the laptop/pc, the LED should be working fine (either showing red or

green). When the reset button is pressed, the LED should turn from green to red to indicate that the board is resetting.

Cleaning: Too much dirt or dust in the pin or the board itself may cause some signal to fail. To clean the Arduino Uno Board: make sure there is no electricity remaining inside the board, use small brushes and mini vacuum specialized for cleaning electronic equipment.

Replacing: The new Arduino Uno Board should be in the perfect condition listed above with the same version as the previously used Arduino Uno.

Bread Board: The breadboard is necessary to perform the user input testing to the Arduino Uno.

Requirement: The Breadboard should be in perfect condition with no errors. When a digital pin connected to the "-" row: it should be active low. When a digital pin connected to the "+" row: it should be active high. When correctly set up by the maintenance procedure, pressing the button should turn the digital pin from active low to active high.

Cleaning and replacing: The same as Arduino Uno Board.

8 Testing

8.1 Testing plans

Test Identification	Requirement being tested	Procedure	Pass Condition
T_01_IP	FR_01_IP	1. Set up a breadboard with 7 buttons wired as active low. 2. Attach the output of each button to digital pins 2 - 8 on the Arduino Uno. 3. Connect to and upload the code in the appendix (labeled "Code to Test Digital Inputs") to the Arduino Uno. 4. Open the serial monitor of the Arduino Uno. 5. Press one button once. 6. Clear the serial monitor of output if any is present. 7. Repeat steps 2-5 for each button up to the 7th button.	If the message "SUCCESS" is seen on the serial monitor for each button press.
T_01_OP	FR_01_OP	1. Set up a breadboard with 7 buttons wired as active low. 2. Attach the output of each button to digital pins 2 - 8 on Arduino Uno #1. 3. Connect the digital pin 12 of Arduino #1 to digital pin 13 of Arduino Uno #1 4. Connect the digital pin 13 of Arduino #1 to digital pin 12 of Arduino Uno #1 5. Connect the ground pin of Arduino #1 to Arduino Uno #2 (a second Arduino Uno) 6. Upload the code from the appending (titled - Code for Control Box) to Arduino Uno #1 7. Upload the code from the appending (titled - Arduino Serial Test Code - Client) to Arduino Uno #2 8. Open the serial monitor on Arduino Uno #2 9. Press each button connected to Arduino Uno #1 once.	If the serial monitor on Arduino #2 prints the numbers '1' through to '7' corresponding to each respective button pressed.

Table 8: Testing Plans

8.2 Results

Test ID	No. Test Performed	Description	% Passed
T_01_IP	14	A standard set of test for the core function of the control box with 7 buttons attach to the digital pin 2 to 8 ascendingly and in descending order.	100%
T_01_IP	20	A standard set of the stress test as multiple buttons are pressed simultaneously.	100%
T_01_IP	20	A standard set of the stress test as each button is pressed multiple times.	100%
T_01_OP	14	A standard set of test for the core function of the control box with 7 buttons attach to the digital pin 2 to 8 ascendingly and in descending order.	100%
T_01_OP	20	A standard set of the stress test as multiple buttons are pressed simultaneously.	100%
T_01_OP	20	A standard set of the stress test as each button is pressed multiple times.	100%

Table 9: Testing Plans

9 Design Review

In retrospect, the design of the code for the control box did fulfil the requirements that were set out by the L7-C20 and other interfaced teams for example, sending the correct output to the 'state machine' when a specific button was pressed by the user. Through testing the design through the functional requirements, it is shown that all the requirements the team set out to do as a subsystem were met before the final submission of the project.

[1]In terms of the clients design requirements it is seen that the requirements set out by the client were also fulfilled in our design, as the only limitation that affected the team was that the coding must be written in the language 'C'. This was done as control box was being wired up to an Arduino UNO therefore, meaning that the code was written in 'Arduino Ide' resulting, to the base language being written in C.

In terms of what could be added in the future to the project was functions such as a better display method to tell the user which floor the lift was travelling to and blue-tooth connectivity for the control box so the user can make the lift move from a distance instead of right next to the elevator. If more time was available, the current code for lighting up the buttons for when a floor is pressed could be further iterated on as well as there was no issues with the current design when meeting the functional requirements.

10 End of life

During the manipulation, clients should avoid pressing buttons with sharp objects such as key in case of damage of the UI panel. When there is water or oil in hand, individuals should wipe their hands before button operation to prevent from button pollution or water seeping into the control box, which could result in a break or direct electric shock to passengers. These protective measures help extend the life of the control box to some extent.

During the process of disassembly, the control box should first be thoroughly checked to make sure whether all the part can function properly. Then the next thing should be done is that all equipment must be in a power-off state. After using safety protective suit, workers start to remove the physical panel of control box by screwdriver, then carefully remove the connection between each button and hardwire cables, cables and UNO board, UNO and MEGA board in sequence. The removal of the button is basically done.

In terms of component recycling, after discarding components with function failure, reusable parts are mainly divided into buttons, wires and Arduino boards.

- Detachable buttons can be reused in various kinds of control panels like a fire alarm button in buildings.
- Wires can be reused for electrical generator and motor wiring, internal connection of small electronic instrument like automobile dashboards.
- Arduino boards can also be reused in a different area, for example, it can be applied to internal motion instruction of industrial robots, unmanned vehicle application and so on.

11 Conclusion

This document summarises the requirements for constructing the model elevators, with a focus on the user interface/control box. The document outlines aspects such as the scope of the challenge and the problem definition created through the requirements set by the customer. The report also entails an overview of the interface and performance requirements plus the constraints for all three elevators that the code for the control box was designed. The design of the UI is also showcased, indicating the process from conceptual designs of the code to the testing and procedures of the code that was implemented in the final product. The document also summarises maintenance procedures and the products end of life and review by the team.

12 References

[1] D. N. Tse, “19S2_ENGG200_300_Explained_project description_v1.docx,” unknown July 2019. [Online]. Available: https://ilearn.mq.edu.au/pluginfile.php/5841079/mod_resource/content/0/19S2_ENGG200_300_Explained_project%20description_v1.pdf.

[2] Stack Overflow, “All Questions,” Stack Overflow, 15 September 2008. [Online]. Available: https://stackoverflow.com/questions. [Accessed July-November 2019].

[3] Arduino, “Download the Arduino IDE,” Many, Unknown Unknown Unknown. [Online]. Available: https://www.arduino.cc/en/main/software. [Accessed August 2019].

[4] P. Stoffregen, “AltSoftSerial,” Atlassian, unknown unknown 2008. [Online]. Available: https://github.com/PaulStoffregen/AltSoftSerial. [Accessed August 2019].

[5] PJRC: Electronic Projects, “AltSoftSerial Library,” PJRC: Electronic Projects, Unknown Unknown Unknown. [Online]. Available: https://www.pjrc.com/teensy/td_libs_AltSoftSerial.html. [Accessed August 2019].

13 Appendices

13.1.1 Appendix A – Breadboard and Arduino Uno testing setup

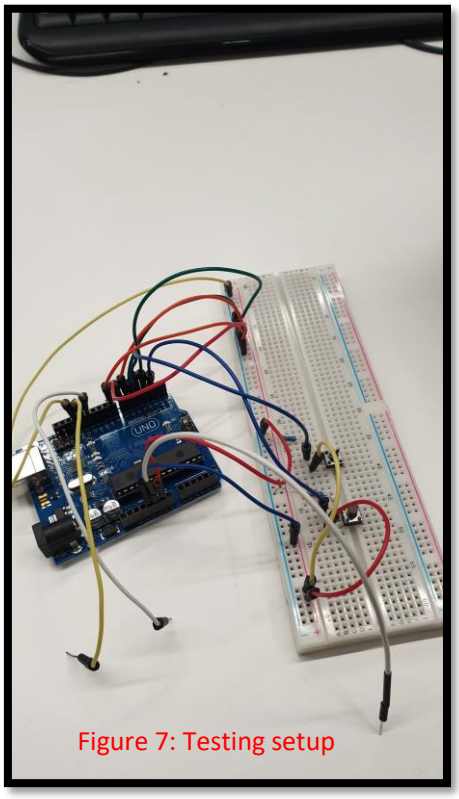


Figure 7: Testing setup

13.1.2 Appendix B – Code for the control box

```
#include <SoftwareSerial.h>

#define STARTPIN 2
#define ENDPIN 8
#define BUTTON_IGNORE_TIME 5000

SoftwareSerial softSerial(12, 13); // RX, TX

/*
  Starting from pin 2 because pins 0 and 8 are reserved for serial

  PIN LAYOUT
  Floor Buttons:  Pin:
  1               2
  2               3
  3               4
  4               5
  5               6

  Door toggle button pin: 7
  Emergency stop button pin: 8
*/

// An array to contain the state of all the buttons
int button_states[9];

// An array to store the time at which a button was pressed
```

```
unsigned long button_press_times[9];

// A long to store how much time has passed between button presses
unsigned long time_passed = 0;

void setup() {

  Serial.begin(9600);

  while (!Serial) {
    ; // wait for serial port to connect. Needed for Native USB only
  }

  softSerial.begin(9600);

  // Set all the input pins as inputs:
  for (int i = STARTPIN; i <= ENDPIN; i++) {
    pinMode(i, INPUT_PULLUP);
  }

  for (int i = 0; i < 7; i++) {
    button_states[i] = 0;
  }
}

void loop() {

  // Update the state of all the buttons, 1 if HIGH, 0 if not
  for (int pin_number = STARTPIN; pin_number <= ENDPIN; pin_number++) {
    button_states[pin_number] = (digitalRead(pin_number) == LOW) ? 1 : 0;
  }

  // Print debug to serial monitor
  print_debug(button_states);

  // Write the value of the pin to serial out if it is HIGH and not
sleeping
  for (int pin_number = STARTPIN; pin_number <= ENDPIN; pin_number++) {

    time_passed = millis() - button_press_times[pin_number];

    if (button_states[pin_number] == 1 && time_passed >=
BUTTON_IGNORE_TIME) {
      softSerial.write(pin_number);
      button_press_times[pin_number] = millis();
    }
  }
}

// Prints the status of button_states to serial monitor
void print_debug(int button_states[]) {
  for (int pin_number = STARTPIN; pin_number <= ENDPIN; pin_number++) {
    Serial.print("[");
    Serial.print(pin_number);
    Serial.print("]: ");
    Serial.print(button_states[pin_number]);
    Serial.print(" ");
  }
  Serial.println();
}
```

13.1.3 Appendix C – Code to test digital inputs

```
#define STARTPIN 2
#define ENDPIN 9

void setup() {
  Serial.begin(9600);

  // Set all the input pins as inputs:
  for (int i = STARTPIN; i <= ENDPIN; i++) {
    pinMode(i, INPUT);
  }
}

void loop() {
  for (int pin_number = STARTPIN; pin_number <= ENDPIN; pin_number++) {
    digitalRead(pin_number) == HIGH ? Serial.println("SUCCESS") : 0;
  }
}
```

13.1.4 Appendix D – Arduino serial test code

```
#include <SoftwareSerial.h>

SoftwareSerial softSerial(12, 13); // RX, TX

void setup() {

  Serial.begin(9600);

  while (!Serial) {
    ; // wait for serial port to connect. Needed for Native USB only
  }

  softSerial.begin(9600);
}

void loop() {
  if (softSerial.available()) {
    Serial.println(softSerial.read());
  }
}
```