Penn Engineering

# ESE5190 Final Project Report

## Project Title: MALL-E

## Team No: 26


Team members:

*Erica Santos*

*Vaibhav Wanere*

*Harsh Yellai*



Report due: December 15, 2023, 11:59 PM

# Table of Contents

# 1 Abstract

*Describe your final project. What was the problem? What was your solution? How did your solution turn out? Was it successful or not?*

We intended the project to mitigate danger in disaster first-responder exploration. Our solution was to make MALL-E (Motion Actuated Lightweight Land Explorer) a small mobile robot that is controlled using a hand glove with an accelerometer on it. This allows the user to control the robot by movement of the palm about different axes, for forward, backward movements and turning. It also includes an ultrasonic obstacle avoidance for seamless navigation through challenging terrain. A standout feature is its mapping capability, allowing the robot to efficiently chart its course and provide a path map. This ensures not only thorough coverage of the disaster site but also facilitates precise navigation.

We were able to successfully build MALL-E as planned and demonstrate proper control. The parameters being measured were displayed on the LCD along with the map as intended. However, we acknowledge that there are some features that are not ideal and can still be improved.

# 2 Motivation

*What is the problem that you are trying to solve? Why is this project interesting? What is the intended purpose? This section is just here for completeness. You can copy and paste what you had written for the project proposal or make revisions if you had to make changes to your goals.*

MALL-E is meant to be the ultimate first responder in disaster scenarios. This versatile robot is engineered to explore perilous environments, aiding emergency teams like EMTs in navigating hazardous areas post-disaster. With a keen focus on safety, the robot is equipped to detect and display real-time data on heat and gas levels, crucial elements in the aftermath of a building collapse where the risk of leaks and fires is high.

The mapping and sensor feedback allow for easy control for the end users when the bot is not directly controllable.

# 3 Goals

This will be an accelerometer controlled mobile robot. Tilting will control movement forward/back and turning.

An ADXL345 Accelerometer, a MQ2 gas sensor, AHT20 humidity and temperature sensor and HC-SR04 Ultrasonic Sensor, will all provide inputs to ATMega32p, and will be displayed back out on the LCD Screen as measured numerical values.

The path displayed will be created through accelerometer values, and draw distance traveled by the robot in a given direction.

Extra: Use the ESP32 to communicate between accelerometer and ATMega, and between ATMega and LCD.

Class Topics covered:

- ADC ⟶ Data input from sensors.
- Serial Communication
    - I2C for Accelerometer and Temperature and Humidity Sensor.
    - SPI for LCD.
- (Should) WiFi Communication.

# 4 Users, Hardware, and Software Requirements Specification

*Define your hardware and software requirements specification here as outlined in the appendix of Final Project Manual.*

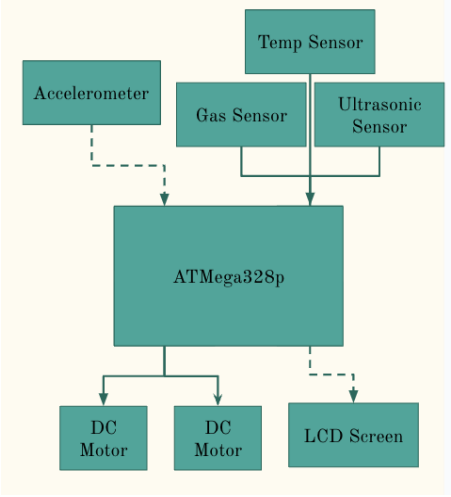
## Users

We believe emergency responders will be our end goal users.

## Hardware

Overview:

Hardware was connected according to the block diagram below:




However, due to pin usage conflicts, the final solution ended up using two microcontrollers for clock 1's input compare match, and the ultrasonic sensor was isolated from the LCD screen's wiring system to allow for both components to work. The ultrasonic sensor's outputs were then displayed on UART connections through a serial monitor as a proof of concept.

The mobile base used was the KatzBot. The pre-made robot utilized two motors.

Functionality:

HRS 01: MALL-E shall use an ATmega328P microcontroller as the main computational component.

HRS 02: MALL-E shall use an ADXL345 accelerometer to translate gestures into accelerations for motor control.

HRS 03: MALL-E shall utilize a MQ2 gas sensor for smoke detection and continuously display readings.

HRS 04: MALL-E shall use an AHT20 temperature and humidity sensor and continuously display the temperature.

HRS 05: MALL-E shall use a HC-SR04 ultrasonic sensor to read and display the distance from objects in its path.

HRS 06: MALL-E shall use a Adafruit 358 1.8" TFT LCD with ST7735R chip to display sensor readings.

HRS 07: MALL-E shall *not* use Li-Ion batteries due to safety concerns.

HRS 08:: MALL-E should use an ESP to communicate between accelerometer and ATMega, and between ATMega and LCD.

## Software

Overview:

The software works within one large while loop. As it continuously reads the accelerometer values, it will change the motor direction to the correct direction to match, and send commands to map that direction on the LCD screen. Tilting forward is forward, twisting left goes left, etc. There are buffers in place to ensure that movements will not move the robot in more than one direction at a time, and that if the accelerometer is flat there will be no movement. Within the loop, it also monitors for changes in the gas and temperature sensors. If found, it will change the print statement on the LCD screen.

Functionality:

SRS 01: Motor control: The code shall convert accelerometer values to a discrete set that will control direction of motor. Each axis of the accelerometer corresponds to a specific direction of motion. The program ensures right movement by sending data from Accelerometer -> Motor

SRS 02: Read sensors: MALL-E's code shall read the values of temperature, humidity and smoke sensors accurately and display them onto the LCD.

SRS 03 : Mapping : The code shall obtain values from the accelerometer and draw a map on the LCD using the functions we have written.

SR04 : Detect obstacles : MALL-E's code shall use the ultrasonic sensor to measure the distance of obstacles in its path and also accurately display them on the LCD screen.

# 5 Verification

*Use the tables below to test your requirements. Test method description should be short and precise. Requirement numbers shall match the requirements defined in section 4.*

## 5.1 Hardware Verification:

| Requirement No. | Test method | Result (Pass/Fail) |
|---|---|---|
| HRS 01 | Flash the motor control code on the Arduino Uno to check the output and verify if it moves | Pass |
| HRS 02 | Generated the values of Pitch and Roll of the accelerometer and read them using the arduino to convert to forward backward and turning movements | Pass |
| HRS 03 | Read the values of the MQ2 and use print statements to see if it prints out the correct values | Pass |
| HSR 04 | Sense the values of temperature and humidity using the AHT20 and print it out on the LCD correctly | Pass |
| HSR 05 | Use a PWM as the trigger for the ultrasonic and reading the echo to measure the distance from the obstacle and display on the serial monitor | Pass |
| HSR 06 | Interface the LCD and use the libraries from our previous lab assignment to draw lines and blocks based on the values from the accelerometer | Pass |

## 5.2 Software Verification:

| Requirement No. | Test method | Result (Pass/Fail) |
|---|---|---|
| SRS 01 | We run the code while keeping the accelerometer flat along all axes and change the orientation of it thereafter to see the movement in the motors. We test for all 4 directions of motion - forward, backward, left turn and right turn | Pass |
| SRS 02 | Interface the sensors and execute the code to read the values and check if they are displayed on the screen | Pass |

| Requirement No. | Test method | Result (Pass/Fail) |
|---|---|---|
| SRS 03 | While the motors are running and the robot moves, check the LCD to look at the map to verify if the direction plotted is same as that of the motion of the robot | Pass |
| SRS 04 | Display the output of the ultrasonic sensor on the LCD. Move the object placed in its path to check for change in measured distance.<br><br>Challenge: We were able to detect the object distance and also measure the change accurately but were unable to print all this on the LCD screen. Instead we connected a laptop to the arduino and used the serial communication to display this data | Fail |

# 6 Results Summary

*Summarize your verification results. What was successful? What was not successful? If verification was not successful, describe in short why it was not successful.*

**What worked:**

We could successfully implement the I2C protocol for the ADXL345 accelerometer and drive the motor based on acceleration. The robot correctly moves in the intended direction based on the accelerometer and stops when it is brought back to its original position.

We use the AHT 20 Temperature and humidity sensor which also works on I2C protocol and could successfully display respective parameters on the LCD screen. Similarly, the smoke sensor was also constantly detecting and printing the readings on the LCD. Hence, the interfacing and functionality of these sensors was complete and correct.

The mapping of the path taken by MALL-E was done successfully. We were able to precisely draw the direction the robot was moving in and also keep track of the turns correctly.

**What didn't work :**

We were not able to achieve what we intended to with the ultrasonic sensor. Although it could accurately detect the distance of obstacles in MALL-E's path, we were unable to print this out on the same LCD screen.

- Challenge: This was because the LCD used PIN8 on the arduino for its operation and so did the Ultrasonic sensor. With the given time for the scope of this project, we were unable to rewrite our own libraries to change this and come up with a solution.

- Solution: In order to test and demonstrate the functionality of the sensor we used the serial monitor on the Arduino IDE.

# 7 Conclusion

*Reflect on your project. Some questions to consider: What did you learn from it? What went well? What accomplishments are you proud of? What did you learn/gain from this experience? Did you have to change your approach? What could have been done differently? Did you encounter obstacles that you didn't anticipate? What could be a next step for this project?*

We believe the overall project was a success. We're proud of getting each individual sensor to work correctly, along with the mapping and motor control from the accelerometer. These three facets were the most important aspects of the project, and we're glad these were completed.

We found out that the integration of the sensors was the toughest part. Individually each of our components seemed to work until we tried to put them all together. As soon as we tried to put them together we encountered issues with pin re-use. Even though these issues were very simple it took us a lot of time to debug them simply because a lot of things were happening concurrently. To overcome this for the demo, we changed plans and split the functionality across two arduino boards to preserve and demonstrate the ultrasonic functionality. We believe we learned how to efficiently debug very long codes. The issue with the reused pin was only identified after diving into the AtMega specifications, re-checking our code along with the libraries and dependencies.

Next time, we believe we can plan more ahead- and confirm the requirements of each individual piece before choosing to go with it.

Good next steps would be to: find a way to condense the sensors back onto one microcontroller (and therefore get Ultrasonic outputs on the LCD screen), and attach/include the ESP32 modules to enable wireless communication between BOTH the microcontroller and the LCD screen, and the accelerometer and the microcontroller. A smaller adjustment would be to improve the mapping display- to make the lines straighter and move the map to expand further across the screen.

Another addition that can be made to MAll-E is to interface a camera. This will show the user what they are navigating through in real-time thus enabling better navigation.

# 8 References

Katzbot documentation:
https://docs.google.com/presentation/d/1ODiGQbmDfqkHQEn6y9IoyKTfTiLZS5M7/edit#slide=id.g29367b5c6c3_1_1

# 9 Links

YT Video: https://www.youtube.com/watch?v=r4ssWoydkjU

Devpost: https://devpost.com/software/mall-e

Github: https://github.com/ESE5190-UPenn-Fall2023/ese5190-project-21-22-24-26

Note: On the github, the "Final Project" folder was used specifically for testing on Erica's machine, as it included dependencies required to run Platform.io

Final Presentation:
https://docs.google.com/presentation/d/1Wg3ZOUpPl9EJqO5lcgagcXUm_G4q9dDy/edit?usp=sharing&ouid=102513877638778851928&rtpof=true&sd=true

# 10 Appendix A: Libraries

*Describe any libraries used here. Remove the section if no external libraries are used.*

**Accelerometer | ADXL345.h and I2C.h:**

Author: Michael Hennerich, Analog Devices Inc.

Source: https://wiki.analog.com/resources/tools-software/linux-drivers/input-misc/adxl345 (source code can be found at "Source Code")

1. **void i2c_init(unsigned long i2c_speed):**

   The i2c_init function serves as the initialization routine for the I2C communication protocol. It configures the parameters for communication speed, denoted by the i2c_speed parameter. It uses the two wire interface library **twi.h** to set bit rate in TWBR (TWI bit rate register) and enables the TWCR (TWI control register). It sets **SCL** and **SDA** lines as inputs for the Microcontroller to act as slave device and take input from the accelerometer.

2. **void adxl345_setDataFormat(uint8_t data):**

   This function uses the six byte data format register: ADXL_DATA_FORMAT 0x31. The DATA_FORMAT register controls the presentation of data to Register 0x32 through Register 0x37. All data, except that for the ±16 g range, must be clipped to avoid rollover.

   A setting of 1 in the SELF_TEST bit applies a self-test force to the sensor, causing a shift in the output data. A value of 0 disables the self-test force.

A value of 1 in the SPI bit sets the device to 3-wire SPI mode, and a value of 0 sets the device to 4-wire SPI mode.

A value of 0 in the INT_INVERT bit sets the interrupts to active high, and a value of 1 sets the interrupts to active low.

When the FULL_RES bit is set to a value of 1, the device is in full resolution mode, where the output resolution increases with the g range set by the range bits to maintain a 4 mg/LSB scale factor. When the FULL_RES bit is set to 0, the device is in 10-bit mode, and the range bits determine the maximum g range and scale factor.

A setting of 1 in the justify bit selects left (MSB) justified mode, and a setting of 0 selects right justified mode with sign extension.

Range Bits set the g range as described in the table below:

| D1 | D0 | g |
|---|---|---|
| 0 | 0 | +/- 2g |
| 0 | 1 | +/- 4g |
| 1 | 0 | +/- 8g |
| 1 | 1 | +/- 16g |

Data formatting needs i2c_write:

**uint8_t i2c_write(uint8_t SlvAdrs, uint8_t len, uint8_t adrs, uint8_t *buf)**

It writes the number of data bytes for the I2C slave device and follows the steps below:

1. Apply a Start condition on the bus.
2. Put the slave device address with bit-0 = 0 onto the bus.
3. Put the byte address to write to on the bus.
4. Put a data byte onto the bus.
5. Repeat step 5 for all data bytes.
6. Apply a Stop condition on the bus.

3. **void adxl345_setPowerControl(uint8_t data):**

This function configures the ADXL_POWER_CTL 0x2d Power Control Register.

Measure Bit

A setting of 0 in the measure bit places the part into standby mode, and a setting of 1 places the part into measurement mode. The ADXL345 powers up in standby mode with minimum power consumption.

4. **void adxl345_setBWRate(uint8_t data):**

   This function uses the register: ADXL_BW_RATE 0x2c which uses i2c_write.

   Rate Bits

   These bits select the device bandwidth and output data rate. The default value is 0x0A, which translates to a 100 Hz output data rate. An output data rate should be selected that is appropriate for the communication protocol and frequency selected. Selecting too high of an output data rate with a low communication speed results in samples being discarded.

5. **void adxl345_getAccelData(uint8_t buf[]):**

   This function uses following registers to output the acceleration:

   These six bytes (Register 0x32 to Register 0x37) are eight bits each and hold the output data for each axis. Register 0x32 and Register 0x33 hold the output data for the x-axis, Register 0x34 and Register 0x35 hold the output data for the y-axis, and Register 0x36 and Register 0x37 hold the output data for the z-axis. The DATA_FORMAT register (Address 0x31) controls the format of the data. It is recommended that a multiple-byte read of all registers be performed to prevent a change in data between reads of sequential registers.

   ADXL_ACCEL_DATA 0x32; ADXL_DATAX0 0x32; ADXL_DATAX1 0x33; ADXL_DATAX 0x32

   ADXL_DATAY0 0x34; ADXL_DATAY1 0x35; ADXL_DATAY 0x34; ADXL_DATAZ0 0x36

   ADXL_DATAZ1 0x37; ADXL_DATAZ 0x36

**Temperature | AHS_AHT20.h and AHS_I2C.h:**

Author: Miles Osborne, fellow 5190 classmate.

Usage:

The libraries described below use the structure:

**typedef struct AHS_AHT20 { uint16_t humidity_data_raw; uint16_t temperature_data_raw; uint8_t state; double relative_humidity; double temperature_celsius; double temperature_fahrenheit; uint8_t CRC;} AHS_AHT20_T;**

1. **void aht20_init(AHS_AHT20_T *sensor):**

   The process begins with the initialization of the sensor reading sequence using I2C communication.

   The ahs_i2c_start function initiates the communication, followed by setting the I2C address for writing and transmitting the trigger measurement command and its associated parameters (TRIGGER_MEASUREMENT_CMD,TRIGGER_MEASUREMENT_PARAM_1, TRIGGER_MEASUREMENT_PARAM_2) to the AHT20 sensor.

Subsequently, the communication is terminated with ahs_i2c_stop.

An 80-millisecond delay (_delay_ms(80)) is then introduced to allow the sensor sufficient time to complete the measurement.

Following the sensor reading sequence, default values are assigned to various attributes of the AHS_AHT20_T structure, ensuring a consistent starting point for subsequent operations. These attributes include raw data readings for humidity and temperature (humidity_data_raw, temperature_data_raw), the sensor state (state), and placeholders for calculated values such as relative humidity (relative_humidity), temperature in Celsius (temperature_celsius), temperature in Fahrenheit (temperature_fahrenheit), and a cyclic redundancy check (CRC).

2. **void aht20_update(AHS_AHT20_T *sensor)**

The aht20_update function is a low-level routine designed for updating an instance of the AHS_AHT20_T structure with fresh data retrieved from the AHT20 sensor through I2C communication.

The function begins by initiating an I2C communication sequence using the **ahs_i2c_start** function and setting the I2C address for reading data from the AHT20 sensor (0x38 with the read bit set to 1).

Subsequently, the function reads specific bytes from the sensor and processes them to update various attributes in the sensor structure.

State Update (Byte 1): The first byte read (sensor->state = ahs_i2c_read()) directly represents the state of the sensor.

Humidity Data Update (Bytes 2 and 3): The next two bytes are combined to form the raw humidity data.

The higher-order byte (sensor->humidity_data_raw = (ahs_i2c_read() << 8)) is read, shifted left by 8 bits, and stored. The lower-order byte is then read (sensor->humidity_data_raw |= ahs_i2c_read()), and the two bytes are combined using bitwise OR.

Temperature Data Update (Bytes 4 and 5): Similarly, the next two bytes are combined to form the raw temperature data. The higher-order byte (sensor->temperature_data_raw = (ahs_i2c_read() & 0x0F) << 8) is read, masked to keep only the lower nibble, shifted left by 8 bits, and stored. The lower-order byte is then read **(sensor->temperature_data_raw |= ahs_i2c_read())**, and the two bytes are combined using bitwise OR.

Finally, the temperature data is masked to ensure it fits within a 16-bit unsigned integer (0xFFFF).

**LCD Display | LCD_GFX.h and ST7735.h:**

Author: ESE 5190 Teaching Staff, J. Ye | Along with additions from Erica S.

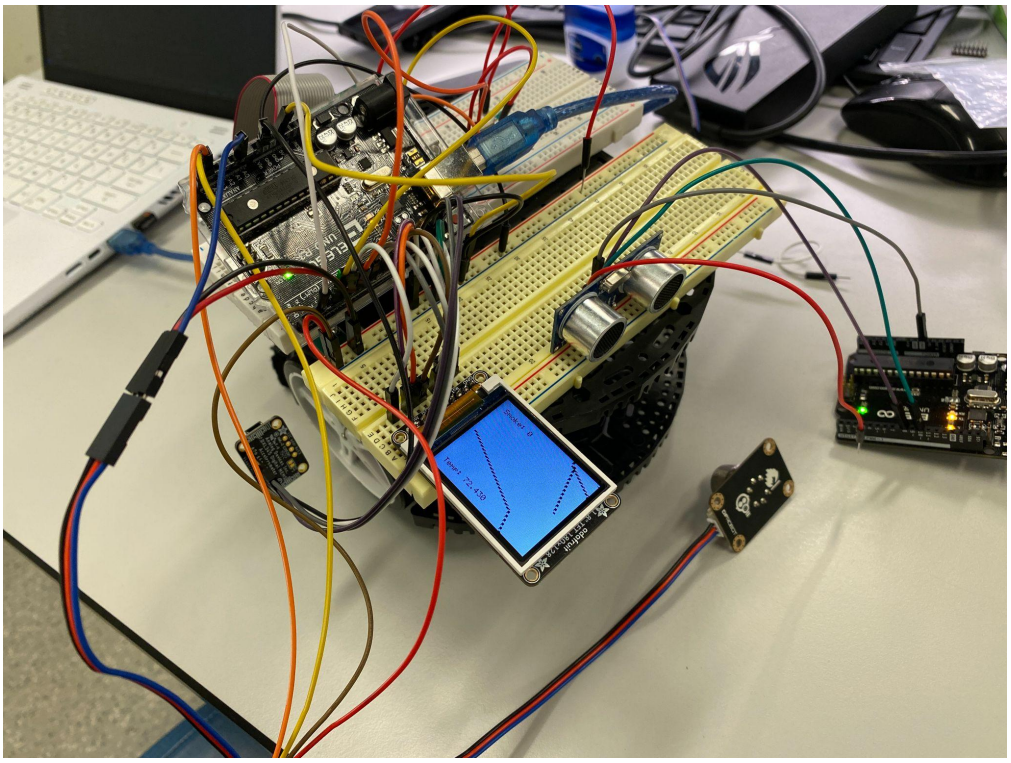
Usage:

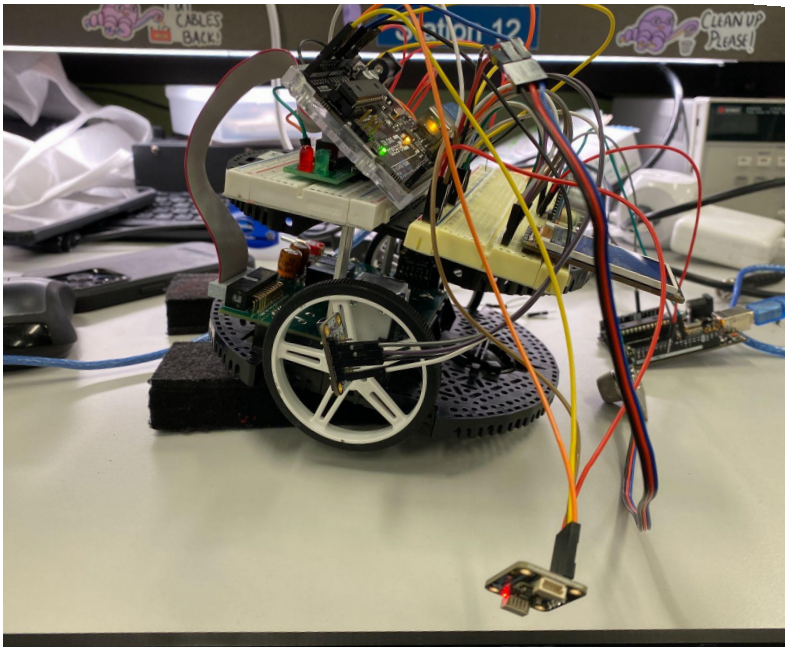
1. **void lcd_init(void):**

The lcd_init function initializes an LCD display using the ST7735 controller. It starts by initializing the necessary pins and configuring the SPI controller. A 5-millisecond delay ensures stability after power-up.
Following is the description of the registers in ST7735 display controller:
ST7735_SWRESET: Software reset command. This initiates a reset sequence for the ST7735 controller, ensuring a clean start. The delay of 150 microseconds allows time for the reset to take effect.
ST7735_SLPOUT: Exit sleep mode command. Wakes up the display from sleep mode, allowing it to operate actively. The delay of 255 microseconds provides sufficient time for the display to exit sleep mode.
ST7735_FRMCTR1, ST7735_FRMCTR2, ST7735_FRMCTR3: Frame rate control commands. These configure parameters related to the frame rate of the display. The parameters 0x01, 0x2C, and 0x2D are specific values for frame rate control.
ST7735_INVCTR: Display inversion control command. This command, with the parameter 0x07, sets the display inversion to a particular mode.
ST7735_PWCTR1, ST7735_PWCTR2, ST7735_PWCTR3, ST7735_PWCTR4, ST7735_PWCTR5: Power control commands. These adjust various power-related settings to optimize the display's performance.
ST7735_VMCTR1: Vcom control command. It configures Vcom control with the parameter 0x0E.
ST7735_INVOFF: Display inversion off command. It turns off display inversion. ST7735_MADCTL: Memory access control command. It configures parameters governing memory access, allowing customization of display orientation. The parameter 0xC8 specifies a particular configuration.
ST7735_COLMOD: Interface pixel format command. It sets the pixel format for data transmission between the microcontroller and the display. The parameter 0x05 indicates a specific pixel format.
ST7735_CASET, ST7735_RASET: Column and Page Address Set commands. These commands define the column and page addresses, specifying the active display area. Specific parameters, such as 0x7F and 0x9F, determine the address range.
ST7735_GMCTRP1, ST7735_GMCTRN1: Gamma correction commands. These configure positive and negative gamma correction values for accurate color reproduction. The provided parameters constitute specific gamma correction values.
ST7735_NORON: Normal display on command. It turns on normal display mode.
ST7735_DISPON: Set display on command. It activates the display. The delay of 100 milliseconds allows time for the display to fully turn on.
ST7735_MADCTL: Memory Access Control command. It sets the memory access control parameters, specifying the default rotation to be used. The parameter MADCTL_MX | MADCTL_MV | MADCTL_RGB indicates a combination of mirroring and rotation.

2. **void LCD_setScreen(uint16_t color):**
   Sets the color of the LCD screen using **SPI_ControllerTx_16bit_stream(color);.**
3. **void LCD_drawString(uint8_t x, uint8_t y, char* str, uint16_t fg, uint16_t bg):**
   Uses **LCD_drawChar(x, y, character, fg, bg);** to write the Temperature values on the LCD screen. The **LCD_drawChar(x, y, character, fg, bg);** uses the ASCII character table to draw the pixels corresponding to the characters using **void LCD_drawPixel(uint8_t x, uint8_t y, uint16_t color)** function.

# 11 Appendix B: Images




CABLES BACK!
CLEAN UP PLEASE!