Spatial Mapping Embedded System
Project Report

By: Sameer Shakeel

Under the guidance of:
Dr. Boursalie, Dr. Doyle, and Dr. Haddara
Department of Electrical & Computer Engineering

Device Overview:

Features:

TI MSP432E402Y Microcontroller:

- Includes the ARM Cortex-M4F Microprocessor onboard
- Operates at a Bus Speed of 60 MHz
- Utilizes the D1 Onboard LED to display status of measurement collections
- Communicates with the TOF Sensor and the PC using I2C and UART Protocols.

VL53L1X Time of Flight (ToF) Sensor:

- Sensor that sends an infrared laser through the transmitter and measures the time it takes for it to bounce back to the receiver
- Measures the distance to the nearest surface by multiplying the photon travel time by the speed of light and dividing it by two
- Has the ability to collect up to 4 meters of measurements
- Operates at a voltage of 2.6V to 3.5V
- Communicates with the microcontroller using I2C Protocols

ULN2003 Stepper Motor:

- Unidirectional Stepper Motor that allows for 360 degrees of rotation in both the clockwise and counterclockwise direction in 512 steps
- Utilizes four LEDs to display the status of the current position
- Operates at a voltage of 5V to 12V

3D Visualization Features:

- Collects measurement data by mounting the ToF sensor to the stepper motor and gathering 360 degrees of measurements
- Stores measurement values in a text file
- Visualizes the measurements by mapping them in a three-dimensional space using the matplot library in Python 3.9.9
- Operates at a baud rate of 115200 bps
- Utilizes the COM4 port to gather measurements from the Time-of-Flight sensor

Cost-Efficient:

- The TI MSP432E402Y Microcontroller sells for about $65
- The VL53L1X Time of Flight (ToF) Sensor sells for about $4
- The ULN2003 Stepper Motor sells for about $10
- This brings the total cost to around $79

General Description:

This is a three-dimensional spatial mapping embedded system that utilizes three main components for data collection, which then visualizes the measured data by graphically reconstructing it in Python 3.9.9. The three main components include the MSP432E402Y Microcontroller, the VL53L1X Time of Flight (ToF) Sensor, and the ULN2003 Stepper Motor. The TI MSP Microcontroller is the heart of the system, as it is in charge of most of the important operations such as controlling the stepper motor, gathering measurement data through I2C

protocol from the time-of-flight sensor, and transmitting that data to a computer through UART communication. The VL53L1X Time of Flight (ToF) Sensor is the main component that gathers the measurements as it uses a built in ADC process to convert a non-electrical signal of its time measurement, and serially transmits the distance in millimeters to the microcontroller. The technique used to get a full 360-degree measurement of a space includes mounting this sensor to the ULN2003 stepper motor. This motor stops every 8 steps to get 64 measurements in total of the space. Once the distance values are communicated to the computer through UART, it is then stored in a text file and is able to perform a full three-dimensional visualization by using Python 3.9.9 after raw data processing occurs.

Block Diagram, and Physical Setup:

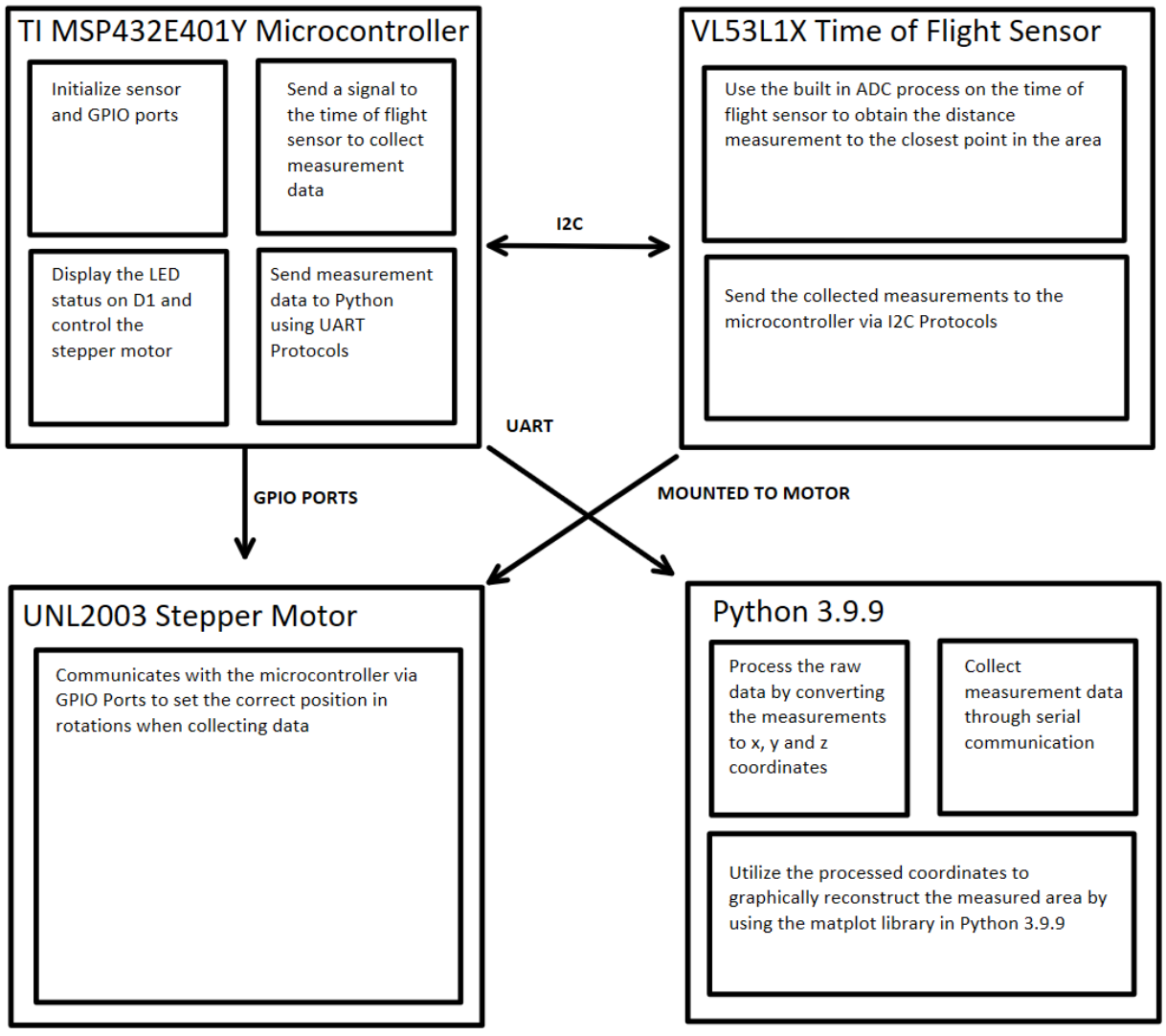


Figure 1: Block diagram of the entire spatial mapping system, displaying how the components interact with each other and their responsibilities

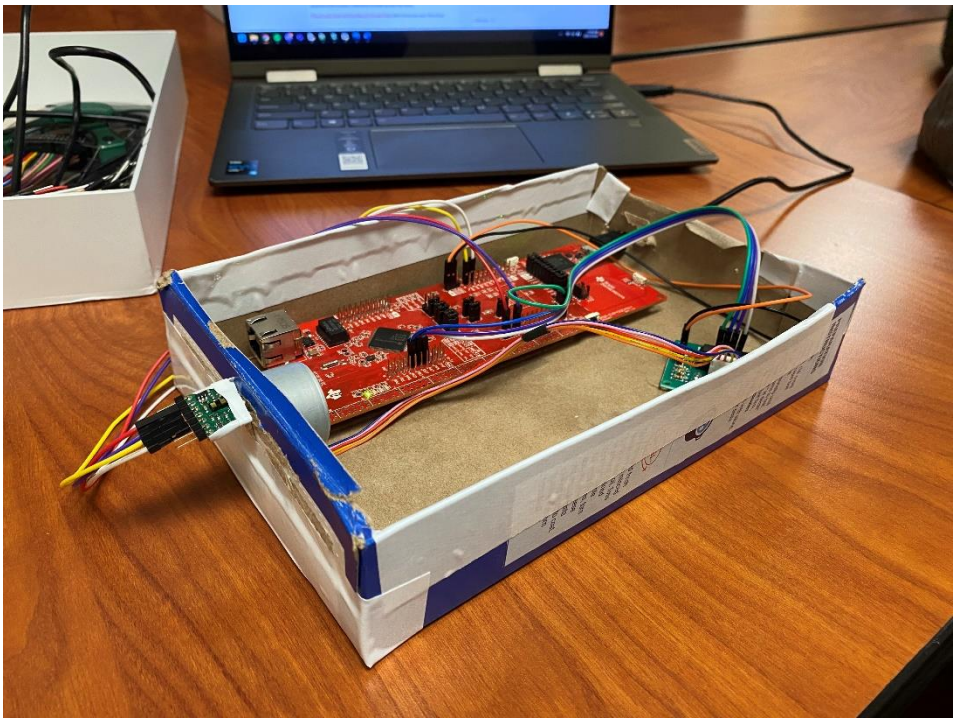


Figure 2: Setup of how the ToF sensor, stepper motor, microcontroller, and personal computer are connected to each other

Device Characteristics Table:

TI MSP432E401Y Microcontroller	
Specification	**Characteristic**
Bus Speed	60 MHz
Serial Port	COM4
Status LED	PN1 (D1)
VL53L1X Time of Flight (ToF) Sensor	
Stepper Motor Pin	**MSP432 Pin**
VIN	3.3V
GND	GND
SCL	PB3
SDA	PB2
ULN2003 Stepper Motor	
Time of Flight Pin	**MSP432 Pin**
VDD	5V
GND	GND
IN1	PM0
IN2	PM1
IN3	PM2
IN4	PM3
Python 3.9.9	
Specification	**Characteristic**
Version	3.9.9
Baud Rate	115200 bps
Libraries Used	Matplot, numpy

Detailed Description:

Distance Measurement:

To control the distance measurement, the MSP432E401Y microcontroller works in unison with the VL53L1X Time of Flight (ToF) sensor and the ULN2003 stepper motor to gather 360 degrees of distance measurements. The microcontroller's job is to first take user input from its onboard push button, then spin the stepper motor in a clockwise direction, collect the measurements from the time-of-flight sensor, transmit it to the computer through UART, and do this all while displaying the status of data collection through the D1 (PN1) LED. As for the VL53L1X, its job is to collect a set number of measurements and to transfer these measurements to the microcontroller by using I2C protocols.

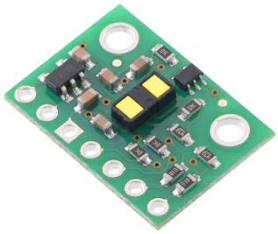


Figure 3: Time of Flight Sensor

Looking more closely at the VL53L1X Time of Flight sensor, this is a device that is able to emit an infrared laser light through its transmitter, then measures the time it takes for it to bounce to the closest surface and back to the receiver. Figure 3 to the right shows its sensor IC represented by the two golden squares on the device; one of them is the transmitter, and the other is the receiver. To calculate the distance, the sensor multiplies the time it takes for the photon to bounce to the surface and back by the speed of light, then divides it by two to only get the distance to the surface (instead of the distance to the surface and back). By using the built-in ADC process, this sensor takes in a non-electric signal of time, converts it to a digital measurement in millimeters, then sends it to the microcontroller by using API function calls with the I2C protocol as shown in Figure 4 below. There are three distance settings, short, medium, and long; in this project, the long setting was set in KEIL allowing for up to 4m to be measured.

```
//API Function call to get the distane measurement via I2C protocol
status = VL53L1X_GetDistance(dev, &Distance);
```

Figure 4: API function call using I2C protocols

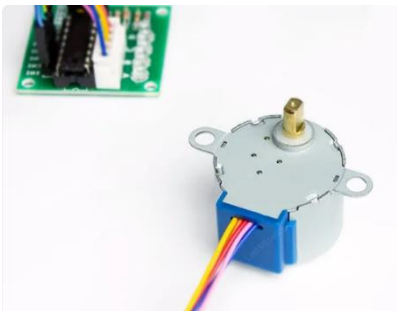


Figure 5: Stepper Motor

Another component that played a large part in the distance measurements was the ULN2003 Stepper Motor. This device was controlled by the microcontroller by giving it a 5V power supply and passing instructions through the GPIO ports M0-M3 to set the speed, direction, and angle. To control the speed, a 10ms delay was put in place between every step. For the direction, the spin function accepted a 0 to turn clockwise, and a 1 to turn counterclockwise. As for the angle, the total number of steps for a full rotation is 512, however, the goal was to gather 64 distance measurements; to do so, the stepper motor had to stop every 5.625 degrees (8 steps). This meant that when the spin function was called, the stepper motor would travel 8 steps (512/64 = 8 steps), then return to the main function. In the main function however, another for loop is implemented that calls the spin function 64 times so that it can collect 64 measurements. On each iteration of the 64-step main function loop, the stepper motor will travel 8 steps, serially transmit the measurement data to Python, and flash the onboard D1 LED as a status indicator. This is performed 64 times to get the correct amount of data in a 360-degree

setting. Furthermore, there was also a second for loop that spins the stepper 512 steps in the counterclockwise direction so that the wires from the time-of-flight sensor would get untangled.

A crucial component for gathering the data was determining when to start and stop collecting measurements. This was done by gathering user input through the Port J1 onboard push button. To start the measurement process, an if statement uses bit masking, and compares PJ1 to a zero (due to the active low setup of the push button); if the button is pressed, then the main function for loop is executed and the scanning process begins. The implementation will automatically stop once 64 measurements are collected. After this, the user can manually move the system and start it again to collect the next measurements, 0.1cm away from the previous set. A stop button, however, was also implemented as the user may need to manually stop the data collection if an error or accidental shifting of the physical system occurs. To do this, a polling method was used that checks if the same button PJ1 is pressed during every iteration of the 64-step main function for loop. If the button is pressed, then the stepper motor will spin the exact number of steps it has spun up to this point, except in the clockwise direction so that it resets the system at the same starting point. This implementation using the polling method may not be as efficient as interrupts, however, it is still a good way to see if a certain event has occurred. Since the measurement collection is happening at high speeds, the system will almost always be able to detect when PJ1 is pressed and go to the right state.

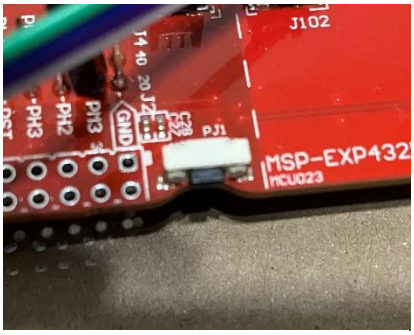


Figure 6: PJ1 onboard push button

UART communication is the most important part of the system in allowing data measurements to be observed by the Python program. In every iteration of the 64-step main function for loop, the time-of-flight sensor collects a measurement, stores it as a double, converts it to meters, then serially transmits it to Python by using UART communication through the COM4 port at a baud rate of 115200 bps. This allows the Python program to collect the data measurements, store them in a text file, then finally graphically reconstruct the measured area.

Visualization:

In order to graphically reconstruct the measured area, Python 3.9.9 was used to create a three-dimensional model that can be rotated and looked at from many angles. To achieve this, the raw measurements that were stored in the text file after the UART transmission, are converted to cartesian coordinates using trigonometric equations. These coordinates are then plotted using the matplot and numpy libraries in Python to create the 3D model.

To program the spatial mapping algorithm, a personal computer running Windows 11, that is capable of running Python 3.9.9 64 bit was used. By installing Python 3.9.9, this allowed for many other libraries such as matplot and numpy to aid in creating the 3D model.

After the user runs the program, they are prompted with the first step which asks them if they want to clear the data that is already stored in the text file. If they answer yes, the program opens the file in write mode, deletes all the data, then moves on to the next step. If they answer

no, the program will move on to the next step without editing the file. This step allows users to map multiple data sets along the x-axis, thus allowing for that three-dimensional view.

The next aspect of the visualization is the steps to acquire the measurement data from the time-of-flight sensor. To do this, the serial library is implemented, that gathers the data using UART protocols from the COM4 port at the specified baud rate of 115200 bps. When this occurs, a flag is sent to the microcontroller which then initializes the while loop that allows the program to collect measurement data using the readline() function, and append them to the text file. While the program is collecting data, the measurements are also displayed on the console to allow users to detect if any impurities or errors arise.

After all the data is collected, the next step is to ask the user if they want to graph the current data. If they answer yes, the program divides all measurements stored in the text file into sets of 64 measurements. The assigned y and z coordinate are generated by multiplying the distance by the sine and cosine of the angle respectively. Since there are 64 measurements, each measurement has an angle attached to it 0 through 360 degrees with the next measurement being 5.625 degrees greater than the previous. Python, however, works in the radians system, so while multiplying the distance by the sine and cosine of the angle, the angle is also multiplied by $\frac{\pi}{180}$. The x-coordinate is then generated based on the set of data. For example, the first set of 64 measurements will all have an x-coordinate of 0, then the next 64 will have an x-coordinate of 0.1, and so on until all sets are accounted for. Once the x, y, and z values for each measurement are determined, the points are then plotted by using the ax.scatter() function in Python.

Once all the points are plotted, a line segment connecting the points is created to better visualize the spatial mapping. This is done twice, one to connect the points on the same x-coordinate, and one to connect between every x-coordinate. Finally, after all of the raw data has been processed, the user is presented with the three-dimensional model of the area they measured.

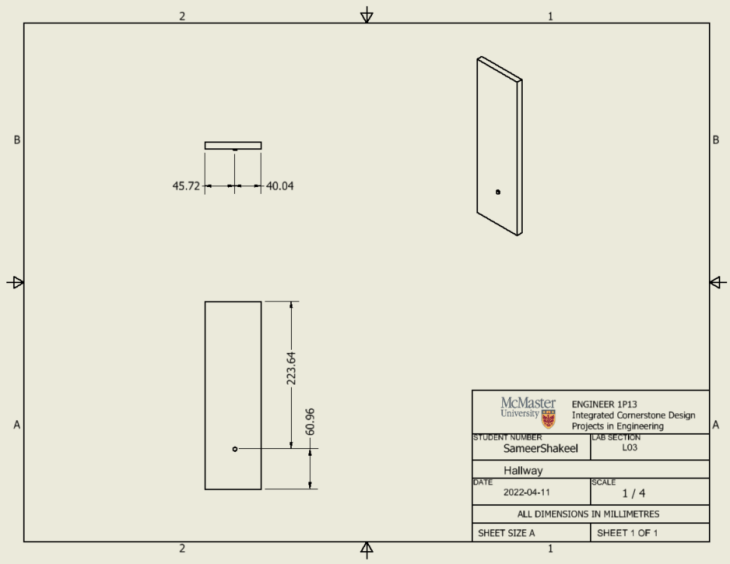


Figure 7: Isometric and Multiview sketch of the hallway used.

Application Example:

1. Firstly, the microcontroller needs to be connected to the personal computer via a micro-USB to USB-A connection. The micro-USB plugs into the microcontroller and the USB-A plugs into the only USB port on the personal computer (in this case). After the connection is made, the onboard LEDS D1-D4 should all turn on to indicate the connection.
   Note: The flashing LEDs only happen if the program is already loaded onto the microcontroller

2. Second, the KEIL project file needs to be opened in which all functions, code, and relevant files are stored. The microcontroller needs these files when the personal computer flashes the program to the device.

3. The next step is that the user needs to load the program onto the microcontroller. To accomplish this, the translate button needs to be pressed, followed by build, followed by load. These buttons are shown highlighted in the right image. Once these buttons are pressed, KEIL should display a message in the build output indicating that the program has been loaded correctly. After the program has been successfully flashed to the microcontroller, the reset button needs to be pressed in order to start the program.

Figure 8: Buttons on KEIL to be pressed

4. After the previous step, all steps relating to KEIL have been completed, now it is time to work with Python 3.9.9 (64 bit) to gather the measurement data and convert them to a three-dimensional model. To do so, the first step is to open IDLE (Python 3.9.9) and open the corresponding Python file included in the project folder.

5. Once the Python program is loaded onto IDLE, the user needs to run the program from IDLE to start. To do so, click Run, and Run Module to start.

6. After the program has started, the console will display a question to the user asking if they want to erase previous data stored in the text file. If this is the first set of measurements being taken, it is in the user's best interest to enter in "Y" as it will delete previous data that may be stored in the text file. When collecting more and more sets of measurements, however, the user does not want to erase previous data so they should enter in "N" for no.

7. The next step after entering in the answer to the console is to finally start collecting data. To do so, the microcontroller and entire system should be placed in the area in which measurements are desired to be taken. Once ready, the user may now click the onboard push button labeled "PJ1" to start. The system will now start spinning the stepper motor, stopping every 5.625 degrees to gather measurement data and flash the onboard LED D1 to signify the status of rotation. The measurements should then be outputted to the

console on IDLE, yielding 64 measurements.

8. Following the 64 measurements being collected, the user will then be prompted with a question from the console asking if they want to plot the current data they hold. If they answer yes, the program will present them with a mapping of all the data stored in the text file. If they answer no, the program will close the COM4 port and end the program.

9. To acquire multiple sets of measurements, after the program ends, the user can follow steps 4-8 as many times as they would like. When working with these steps however, it should be noted that the user should answer "N" (no) to the question asking if they want to erase previous data. This allows for multiple sets of data to be collected and plotted across the x-axis.

10. Once the desired number of measurements have been acquired, the user can answer "Y" (yes) to the question asking if they want to graph the data. This will then present them with a full three-dimensional spatial mapping of the area recorded.

The expected mapping of a hallway is shown below in Figure 9. This is an output that was achieved by taking 10 sets of 64 measurements while moving the whole system forward after each set was collected. What this did is it allowed for a range of measurements to be collected from a hallway and then mapped by using Python. One thing to keep in mind, however, is that the y and z values are the values that are being calculated by the program. The x value of each set of measurements is then set by the program and changes for every new set of measurements. The orientation may look different, however that is simply because the 3D model is rotated and seen from a different point of view.

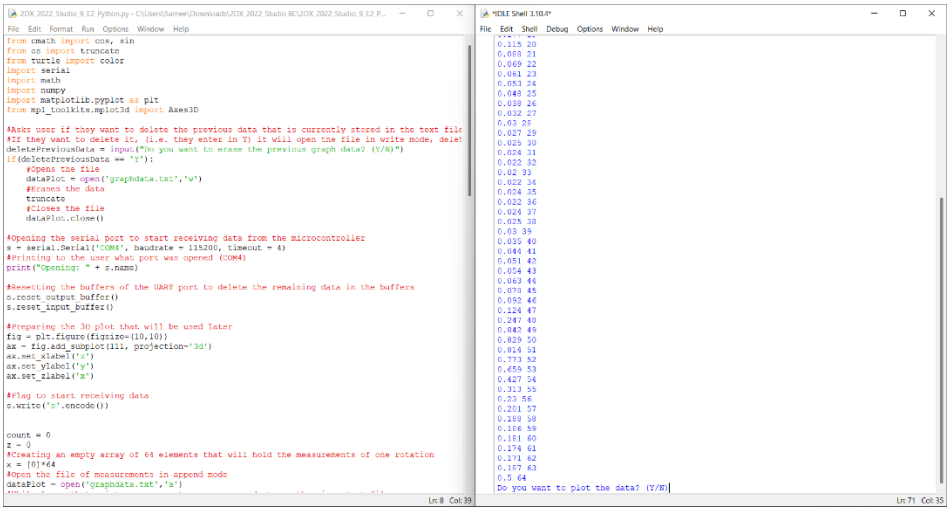


Figure 9: Expected live output of the data collection on IDLE Python

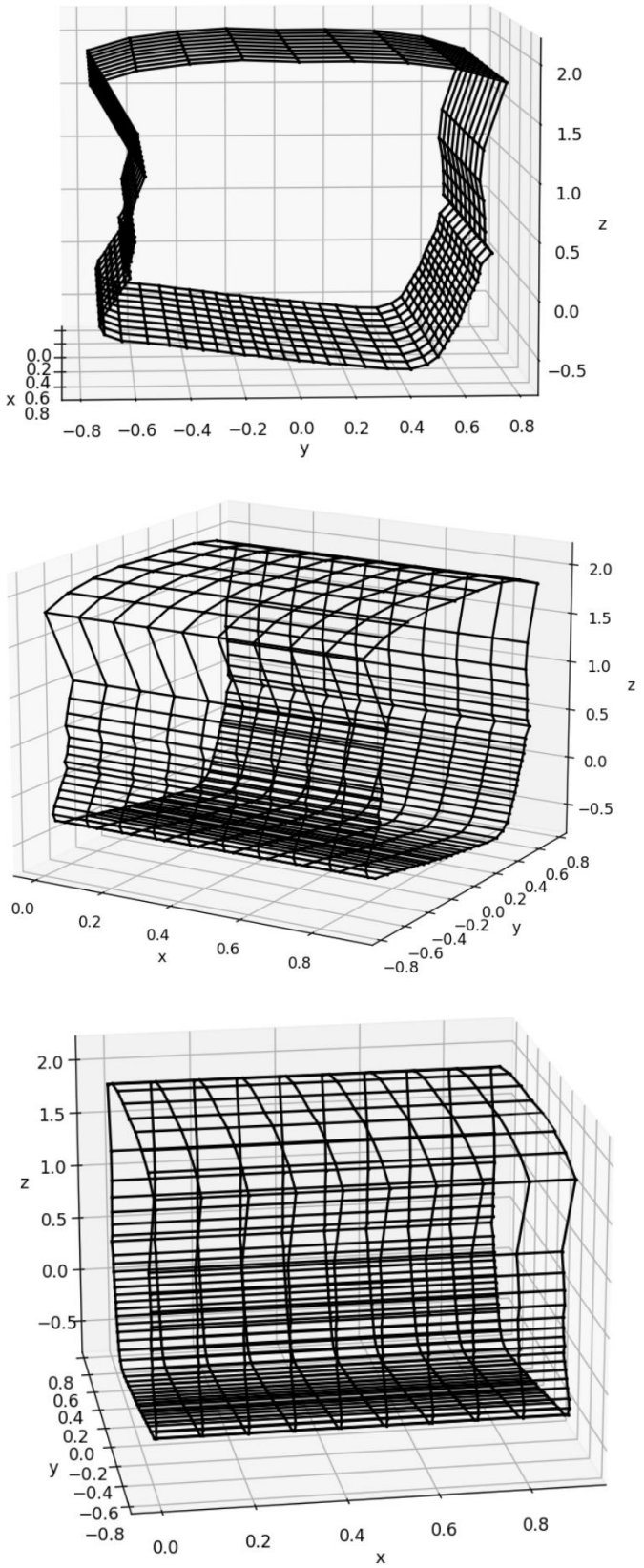


Figure 10: Expected 3D mapping of a hallway

Brief Users Guide:

1. Install KEIL
    a. KEIL MDK Version 5 is the software used to give instructions and flash the program onto the MSP432E401Y microcontroller. To install it, click here to go to MDK Version 5 installation page and once there, click on the "Download MDK" button to download the installer.
    b. After it has been downloaded, double click on the installer to open the installation wizard. Follow all the steps and prompts that appear on screen and make sure to read and accept any licensing or terms of service agreements.
    c. Once the wizard has finished the installation, the Pack Installer will open and prompt the user to choose the device. Ensure that the MSP432E401Y (under "Devices") is selected as this is the microcontroller that is used. Then, proceed to the "Device Specific" section to download and install the family pack for the microcontroller.
    d. The installation is now complete and KEIL is ready to be used.

2. Install Python 3.9.9
    a. Python 3.9.9 is the installation that is used to collect measurements and graphically reconstruct the data. To install it, click here to go to the Python installation page and scroll down until the Python 3.9.9 download button appears. Click download and wait for it to complete.
    b. After it has been downloaded, double click on the installer to open the installation wizard. Follow all the steps and prompts that appear on screen and make sure to read and accept any licensing or terms of service agreements. When going through the wizard, ensure that the "add to path" button is selected for optimal usage.
    c. Once the wizard has finished the installation, it is now time to install the correct libraries that will be used to collect measurements and graphically reconstruct the data. To do so, first open command prompt by searching for it in Windows search. Next, run the commands shown below one by one, in order to install the libraries needed.
        i. pip install pyserial
        ii. python -m pip install -U numpy
        iii. python -m pip install matplotlib

3. UART Communication
    a. UART communication is completed through a USB port on the personal computer. To ensure that the correct port is used, it needs to be identified by the user in Windows Device Manager.
        i. To do so, search for "Device Manager" in the Windows search and open up the first result.

ii. Then, find the name of the USB port that is currently connecting the microcontroller to the personal computer by opening the "Ports" tab and locating the port number that matches with XDS110 USER UART. In the application example, it was named COM4, however, it may differ user to user.

b. Once the correct port has been identified, the name needs to be set in the given Python code. To check if the correct port has been identified, open the Python file in IDLE and locate the line reading:
"`s = serial.Serial('COM4', baudrate = 115200, timeout = 4)`".
This is located on line 23 of the Python code and if value named "COM4" does not match your USB port, then change it to your respective identifier.

Limitations:

1. Maximum Quantization Error in Distance Measurement
   The maximum error in the measurement is seen by the full-scale measurement distance/2^n where n is the number of bits. In this case, n is equal to 16 and the full-scale measurement distance is 4 meters when set to the long mode. Thus, 4/2^16 yields a maximum quantization distance measurement error of 6.1 x 10^-5 m. This is a limitation on the design as the error can always go to a lower value with more bits or a smaller full-scale per bits ratio.
2. Constrained Serial Communication Rate
   The laptop that was used to complete this project only allowed for a maximum serial communication rate of 128 Kbps. If this value were higher, the system would be able to operate at a faster rate, thus lowering the wait times when collecting data measurements.
3. Limitations on the I2C Protocol
   To transfer the measurement data taken by the time-of-flight sensor to the microcontroller, I2C protocols had to be used. The problem with this is that I2C protocols for this device were limited to 100,000 bps because the ports used from the personal computer were capped at 100 KHz. Thus, a limitation was the serial data transfer speed from the ToF sensor to the microcontroller.
4. FPU
   A 32-bit floating point unit was used as per the specifications of the MSP432 microcontroller. This puts a limitation on the microcontroller because when comparing it to the floating-point units employed by the Python code on the computer, that shows a floating-point unit of 64 bits because of the double point format. Overall, the 32-bit floating point unit on the microcontroller acted as a limitation for the whole project.
5. X Coordinate Measurements
   Since the time-of-flight sensor and microcontroller are not capable of measuring when the user moves the entire system, this puts a limitation on the accuracy of the mapping as the x axis coordinates are hard coded to increase by 0.1cm for every set of measurements

Circuit Schematic:

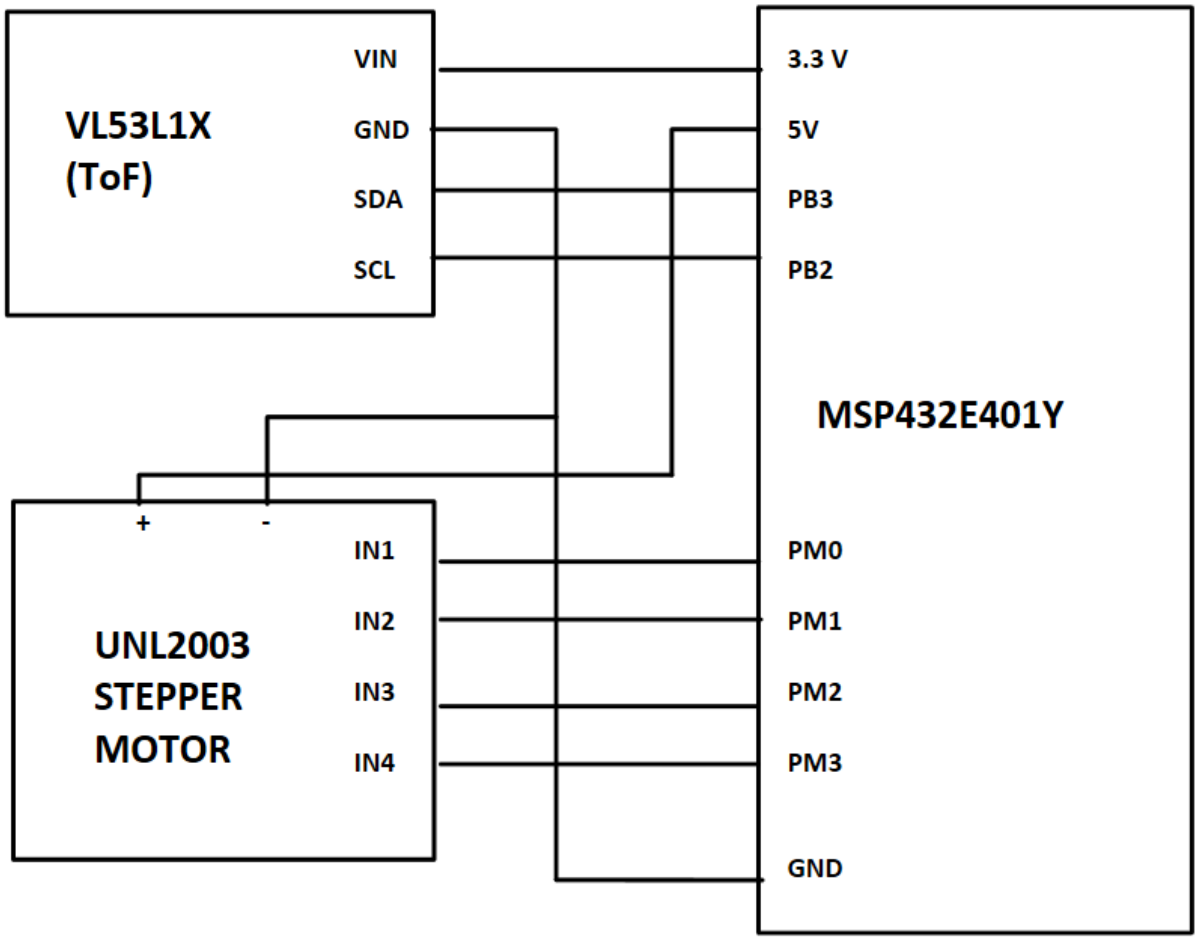
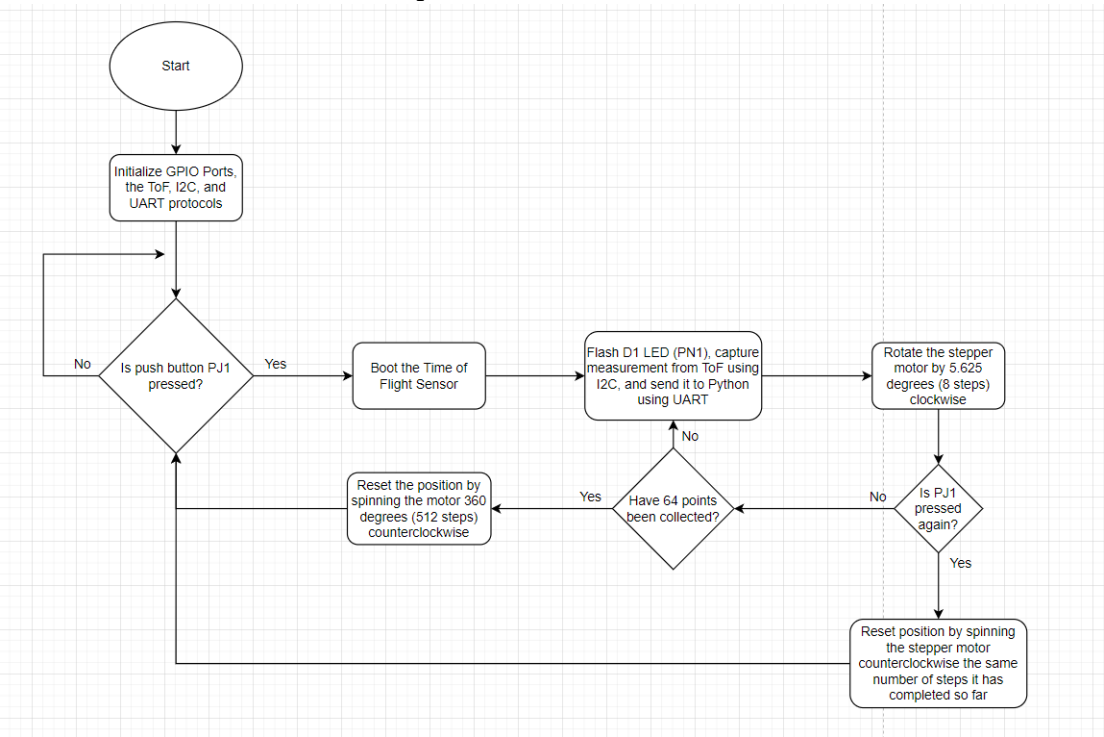


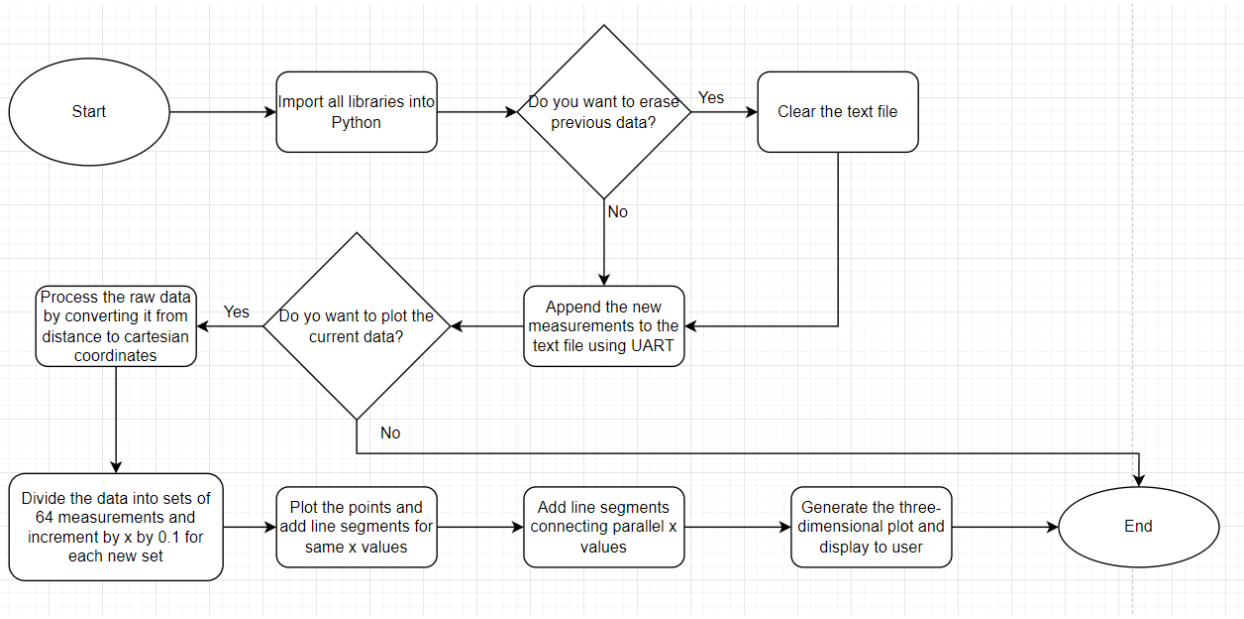
Figure 11: Circuit Schematic represented by a block diagram

Programming Logic Flowcharts:

Flowchart for Microcontroller Operations:



Flowchart for Python Code:



Next Steps:

The learning outcome of this project proved to be very beneficial in aiding with the understanding of microprocessors, digital I/O, peripherals, communication protocols, data collection, and three-dimensional graphical reconstruction. This project, however, was not perfect as there are many more improvements that could be made to improve the design and the methods of data collection. First of all, the physical design was very poor as a small cardboard box paired with double sided tape was all that was used to hold the entire system together. A more practical approach for the future would be to create a model in software, and 3D print it with the correct measurements and constraints. Building onto that, 3D printing an actual mount for the time of flight sensor is the most important next step for the project as currently, duct tape and double-sided tape are not a reliable mounting method especially on the tiny key shaft of the ULN2003 stepper motor. For now, it works well and only presents very few inconsistencies, however, it is in the projects best interest to create an actual reliable mount. Next, improving upon the method of data collection is a next step that should be taken to refine the collected data to a more accurate reading. Currently, on the reconstructed model of the measured space, there are numerous impurities on the model that are not actually present in the measured space. This can either be due to the fact that only 64 measurements are taken, or because the current time of flight sensor is not the ideal device needed to collect the data. Therefore, some next steps may include sampling even more measurements, or switching to a different system such as LiDAR. This would change many aspects of the project and be much more costly, however, the results would be a more efficient and accurate reconstructing device that could be the foundation of an even bigger project if it has a good reliability. Overall, despite all of the possible next steps this project can take, the goals that were set out in this project were met with satisfaction. A time of flight sensor was used in unison with a stepper motor to gather data, transfer it to a text file through communication protocols, and reconstruct the measurements to show a 3D mapping of the recorded space.