UFG4G89F23 Lidar Sensor

Device Overview

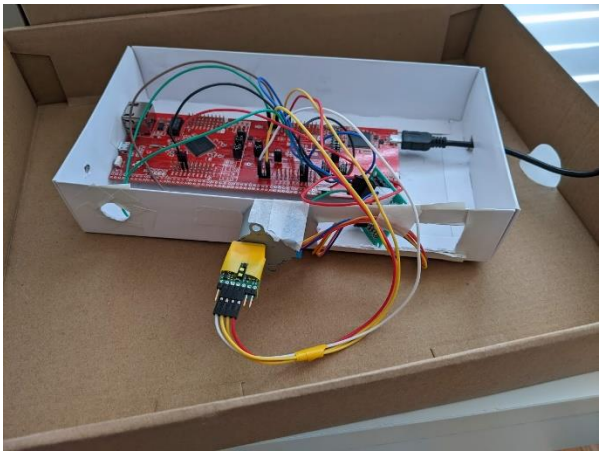

Figure 1: Image of the UFG4G89F23 Lidar Sensor

Features

- MSP-432E501Y Microcontroller
  - 40MHz Bus Speed
  - Arm Cortex-M4F Processor [1]
  - 40MHz Bus speed
  - Size:17.4cm×6.35cm×10.8 mm
  - $70.36
- VL53L1X Time-of-flight Sensor
  - Measurement range of 4000 mm [2]
  - Frequency range of 50Hz [2]
  - 2.6 to 3.5 V Operating Voltage [2]
  - Size: 4.9 x 2.5 x 1.56 mm [2]
  - $24.95
- Communication to VL53L1X Time-of-flight Sensor via I2C
- UNL2003 Stepper Motor and ULN2003 Stepper Motor Driver
  - 360 degrees rotation
  - 5-12 V power supply [3]
  - $10.53
- Communication to PC via UART (115200 baud rate)
- Lidar Sensor systems coded in C
- Data visualization via Open3D and python

Description

The UFG4G89F23 Lidar Sensor is capable of scanning data in 360 degrees to be able to create a detailed 3D visualization of the data. The design has high accuracy and real time scanning abilities. The design is easy to use and ideal for renovations of homes as the 360 degrees scan will display the room in a 3D design with dimensions.

The UFG4G89F23 Lidar Sensor is secured in a strong box with holes allowing the wires to connect to the VL53L1X Time-of-flight Sensor and the UNL2003 Stepper Motor. The VL53L1X Time-of-flight Sensor is connected to the UNL2003 Stepper Motor using a block. The UNL2003 Stepper Motor stepper motor rotates 360 degrees stopping at timed angles to take data, after rotating for 360 degrees, it rotates back to ensure the wires do not get tangled. The VL53L1X Time-of-flight Sensor is able to measure the distance by emitting infrared light and then collecting the light being reflected back. An equation is then used to determine the distance of the object that reflected the infrared light back.

The Lidar sensor system sends the data collected to the PC via UART, where the data is converted into cartesian points. These points are then used to create a 3D visualization by plotting the points.

Block Diagram

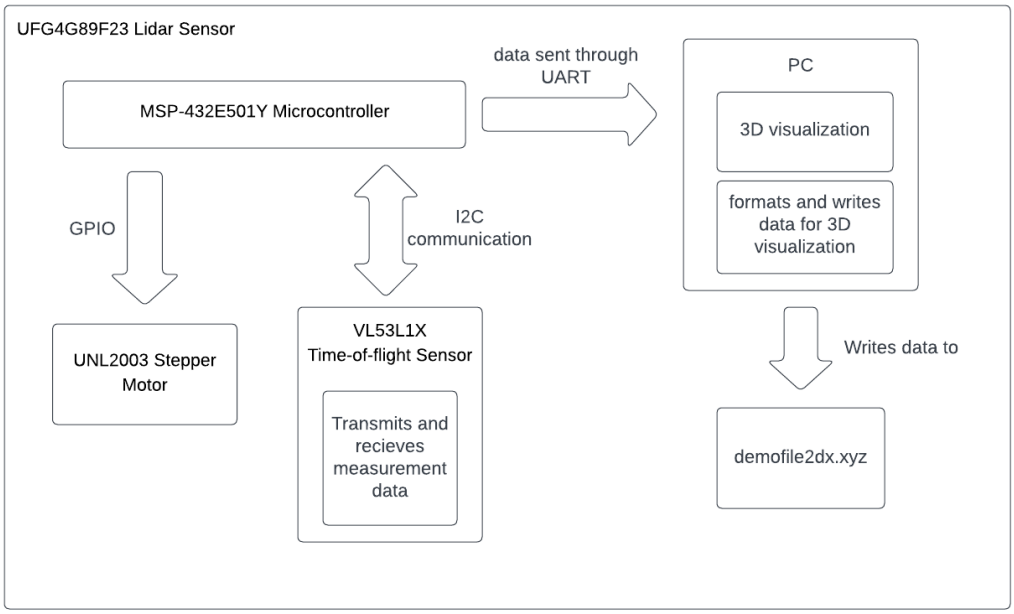

Figure 2: Block diagram of the UFG4G89F23 Lidar Sensor

Device Characteristics Table

Table 1: device characteristics of the MSP-432E501Y Microcontroller

Feature/Pin Used	**Detail**
Bus Speed	40 MHz
Size	17.4cm×6.35cm×10.8 mm
Serial Port	COM7
Baud Rate	115200
Sensor status	PN1
Scanning Status	PF4

Table 2: VL53L1X Time-of-flight Sensor

Feature/Pin Used	**Detail**
Measurement Range	4000mm
Size	4.9 x 2.5 x 1.56 mm
VIN	3.3V

GND	GND
SDA	PB3
SCL	PB2

Table 3: UNL2003 Stepper Motor

Feature/Pin Used	**Detail**
Steps	512
Power supply	5-12V
VDD	5V
GND	GND
IN1	PH0
IN2	PH1
IN3	PH2
IN4	PH3

Detailed Description

Distance Measurement

The measurement of the distance is obtained by the VL53L1X Time-of-flight Sensor. The sensor takes the distance measurement after the motor has turned a certain amount of degrees. In this case it is 5.62 degrees. The motor will rotate 360 degrees, where after 5.62 degrees it will stop to measure the distance using the ToF sensor.

The Time-of-Flight senso sensor receives and sends data to the MSP-432E501Y Microcontroller using I2C. To determine the distance, the time-of-flight sensor uses infrared light. The sensor has an emitter and a collector, where the emitter emits infrared light, and the collector collects it. This means that the time-of-flight sensor emits the laser light and collects the light that is bounced off the object. The distance is then calculated with the formula $D = \frac{e}{2t}$. D is the distance, e is the speed of light, and t is how long it took for the laser light to return. We divide the formula by 2 as we have determined the time for the light hitting an object and returning but we require the distance till the object thus we divide the equation by 2.

To transmit and receive data from the VL53L1X Time-of-flight Sensor, it must first be turned on using the PJ0 button on the microcontroller, where the LED PN1 should signify the Time-of-Flight sensor turning on. The button is enabled in the polling method. When the button is

pressed it toggles the flag called *flag2*, which is initially initialized as 0 or off. When the flag is toggled to 1, the Time-of-Flight sensor starts capturing data. The value of the flag disables or enables the Time-of-Flight sensor data acquisition using an if statement. The button at PJ0 will also reset the data acquisition variables for the lidar sensor ensuring that when the sensor is turned on again, the data acquisition is reset and ready to take new variables.

The UNL2003 Stepper Motor rotates in 360 degrees or 512 steps, where it stops after 5.65 degrees to be able to allow the Time-of-Flight sensor to collect the distance measurement. The button PJ1 enables the flag called *flag*, which enables the motor to rotate. This allows the motor to rotate 512 steps where the variable *val* is used to count the steps where it stops the motor after 512 steps (360 degrees). The values collected from the Time-of-Flight sensor using I2C are then transmitted to the PC using UART using a bandwidth 115200. The distance value collected from the sensor and the motor angle values are transmitted to the PC.

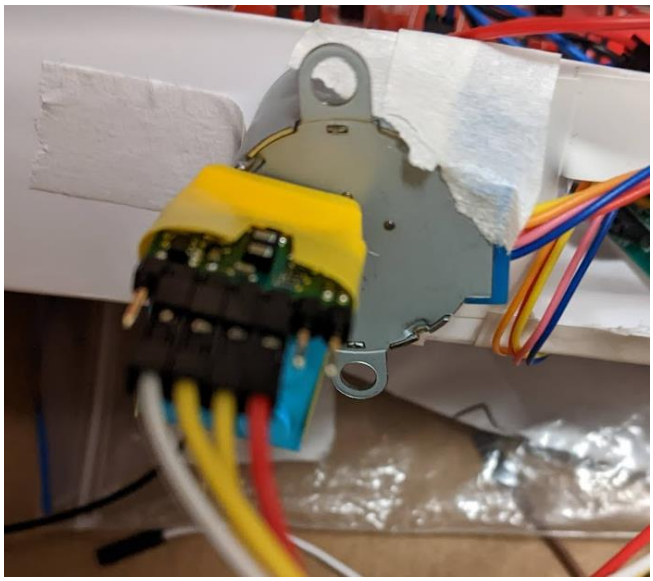

Figure 3: image showing the VL53L1X Time-of-flight Sensor connected to the UNL2003 Stepper Motor

A python script *deliverable2.py* collects this data to be able to output the data into a 3D model using Open3D. The data can be collected due to the pySerial library and the 3D model is produced using the script due to the Open3D library in python 3.9. The python script collects the distance and the angle measurement values and converts them to y and z values that can be plotted. The y value of the data is obtained using the formula $y = d \cdot \cos(2\pi \cdot angle)$ and the z value is obtained using the formula $z = d \cdot \sin(2\pi \cdot angle)$. Where d is the distance measured from the Time-of-Flight sensor and the angle is the angle of the motor during the taking of the measurement.

As the lidar sensor can take 3D data, the python script can take multiple data using a command where the press of the button starts another data acquisition by the lidar sensor. The x value, representing the displacement, is increased after each complete data acquisition. Since after each data acquisition a prompt will be asked, answering yes will allow for another data communication, where pressing PJ1 on the microcontroller will start the Lidar sensor again. The difference in this data acquisition would be the displacement as it has increased. The x,y,z values obtained are placed into a .xyz file called *demofile2dx.xyz* which is able to be opened by Open3D.

The python script can output a point cloud in Open3D using the points in the xyz file and then a model where the points are connected which better visualizes the values collected. The python script is able to individually connect the lines from the points using a loop.

Visualization

The 3D visualization of the data is performed by the Open3d software. This is done through the use of python and its Open3D library which visualizes the data using an xyz file. The python file first writes new data to the .xyz file called *demofile2dx.xyz*. This data is first represented as a point cloud using the *o3d.io.read_point_cloud ()* function. A point cloud represents all the data as points on a cartesian plane.

The python script will also connect the points of the data for a better visualization allowing the data points to be represented as a shape. The python script turns the data points into slices. The yz slices in the visualization are the cartesian points of the distance measured with the sensor. The slices at the same displacement or x value, have a line connected to the next data point except for the last one which connects to the first data point of that displacement. This outputs a 2D shape at different displacements and allows for the visualization of different displacements. This is performed using 2 nested for loops to allow the values of each displacement to connect to each other. To connect the lines from the points of different displacements, we create a separate for loop that is only performed when there are 2 different displacements of data points. When the lines are created the python uses Open 3D to visualize the shape using the *o3d.visualization.draw_geometries([line_set])* function.

The 3D visualization displacement can be as long as needed but the y and z values of the points are limited by the range of the *VL53L1X Time-of-flight Sensor*, which has a range of 4000mm. The displacement or the x value can be as long as needed as long as it does not exceed the memory space allocated to store the values. This is due to each variable and array in programming to have a specific amount of memory space allocated for it. Due to the python script having an infinite for loop that asks for the continuation for the data acquisition, continuously pressing yes, we can collect as much data as we need.

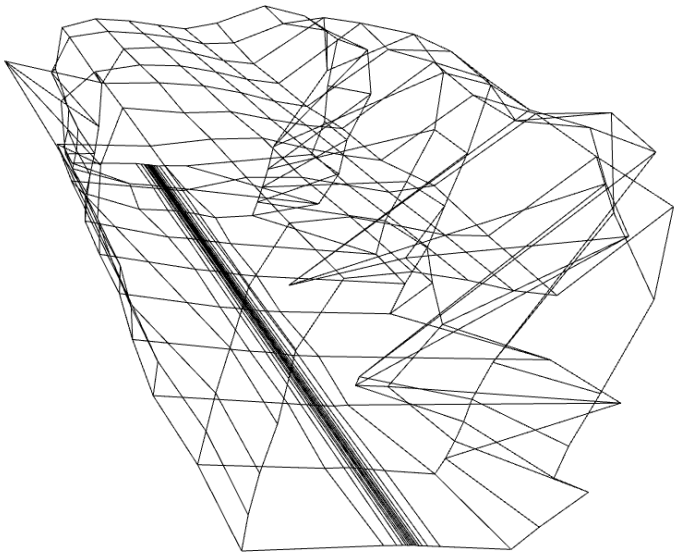

Figure 4: points visualized as a 3D model using open3D

Application Example and Instructions

Before the lidar system is able to be functional, there are a few tools to set up to ensure that the system is functional on one's device. The setup process is as follows :

- Install a release of Python from Python 3.8-3.10. The releases of python can be installed using *https://www.python.org/downloads/*. After running the setup ensure the option "Add Python 3.X to path" is selected.

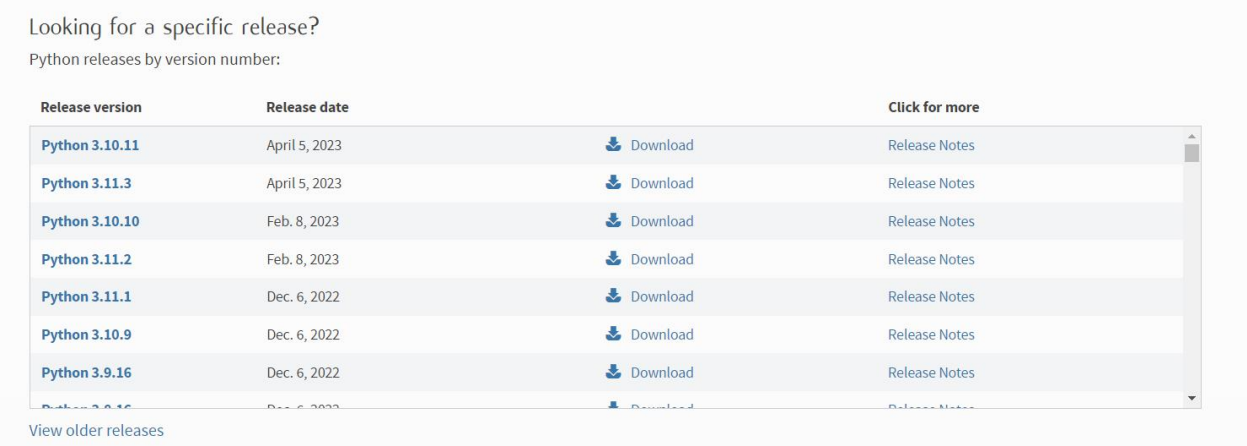

Figure 5: image of where the releases of python can be installed

- The following setup can only be performed after the installation of python. Open command prompt using the search tool on Windows 10. The library pySerial is installed by typing the following command "pip3 install pySerial". This installs the necessary pySerial files.

- The following setup can only be performed after the installation of python. Open command prompt using the search tool on Windows 10. The library for Open3D can be installed using the following command “pip3 install open3d”. This installs the necessary open3d files.
- The python file can only be run using an IDE. The latest release of the IDE *Visual Studio Code* is recommended. The latest release can be installed using *https://code.visualstudio.com/*. After installing, open *Visual Studio Code* and click “extensions” on the left side bar. Search for python and install the python extension. This allows python codes to be run on the IDE. After that the shortcut ctrl+shift+D can be used to run the python script.

After the setup we can use the LIDAR device for the collection and visualization of data. The following will be steps to be able to use the device correctly.

1. Open the *deliverable2.py* python file using an IDE.
2. Plug in the Lidar sensor to your computer.
3. The COM port can be set to your device by opening ‘Device Manager’ on your computer by searching it on the windows search. The port can be changed according to what port the device is connected in for example if it's connected in port 7, the line on the python script would be set to *s = serial.Serial('COM7', 115200, timeout=10)*.
4. The variables for the length (x-value) can be changed in the file by changing the variable width in the first While True statement. The statement is initially kept as *width += 0.*
5. Run the python script.
6. The terminal will ask to communicate with the line ‘communicate? [Y/N]’. Typing Y will allow the python script to get data variables from the sensor while pressing N will stop the python script from getting data variables. After each set of data collected, the prompt would be asked again until ‘N’ is entered. After each communication the displacement value is increased.
7. Press the button on PJ0 to enable the sensor and then press button PJ1 to start the data acquisition process.
8. After the data acquisition is completed and ‘N’ is entered as the input, the python script would output a 3D point cloud of all the data. Closing this point cloud, another 3D data is shown which represents the data points as a 3D shape. The x slice is the displacement value and the yz slices in the visualization are the point representation of the distance and angle measured using the Lidar system.

An example for the output of the data acquisition process is shown below with the location scanned using the Lidar sensor. The 3D model outputted looks very similar to the location but the reason the large corridor is not scanned as well is due to the range of the sensor not being long enough.

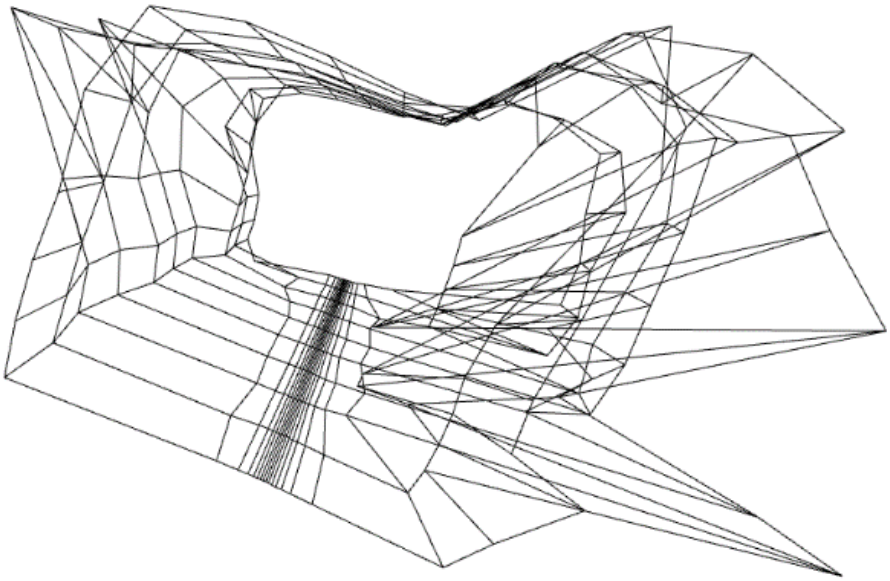

Figure 6: 3D model of a hallway

Figure 7: Image of the hallway that was scanned by the lidar sensor

Limitations

1. The MSP-432E401Y microcontroller features a cortex-M4F processor core which means it has a 32-bit single precision FPU [1]. The decimal numbers will only be accurate to 32-bits and due to

the need of using trigonometric functions, the results would be limited. But all these calculations would be performed in python which has a float of 64-bit double precision, which means that there would be no problems caused by the limitations [4].

2. The max quantization error for the ToF sensor is $quantization\ error = \frac{maxval}{2^n} = \frac{4000mm}{2^{16}} = 0.061$. Since the max range of the sensor is 4000mm and the ToF sensor I2C communication is occurring in 16 bits [2].
3. The maximum baud rate is 115200 bits per second as it uses UART serial communication. I attempted the serial communication with larger baud rates using realterm but it resulted in error.
4. The communication between the microcontroller and the ToF sensor is of I2C communication. This means that the direction of communication is bidirectional where the 2 wires are connected to the serial Clock line and the serial data line of the ToF sensor. The communication with the IMU module of the sensor is also of I2C communication. The speed of the communication with the TOF module is 400 kHz [2].
5. The speed of the motor is depending on the delay and the steps. Since the motor is limited to 512 steps and we require a delay while the motor is turning. This can bottle neck the system especially since after the motor is turned by a step, the sensor may be called. The speed of the sensor would not bottle neck the system as much as the motor but there would be some delay due to the transmissions through the communication between the sensor and the microcontroller in I2C and the communication between the microcontroller and PC in UART that occurs right after. These bottlenecks were tested using the time.h library in C and using the clock module to test the speed of the systems with different bus speed or changing a few details, which represented the bottleneck mentioned before in numbers.

Circuit Schematic

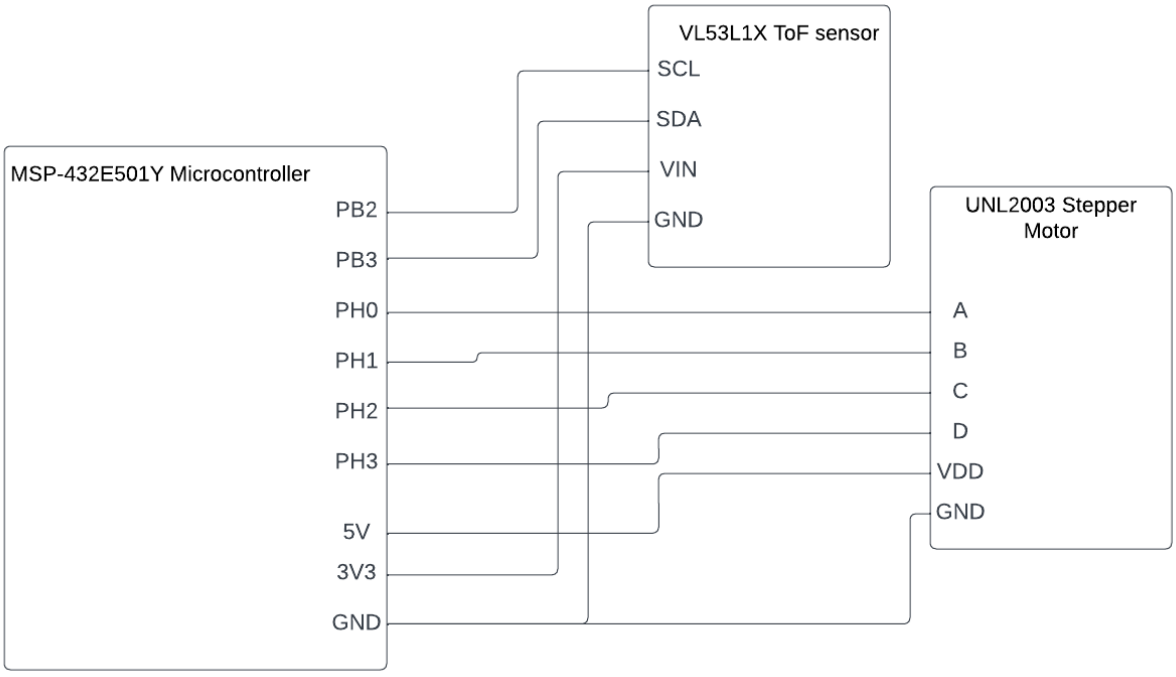

Figure 8: Schematic diagram of the UFG4G89F23 Lidar Sensor

Programming Logic Flowchart

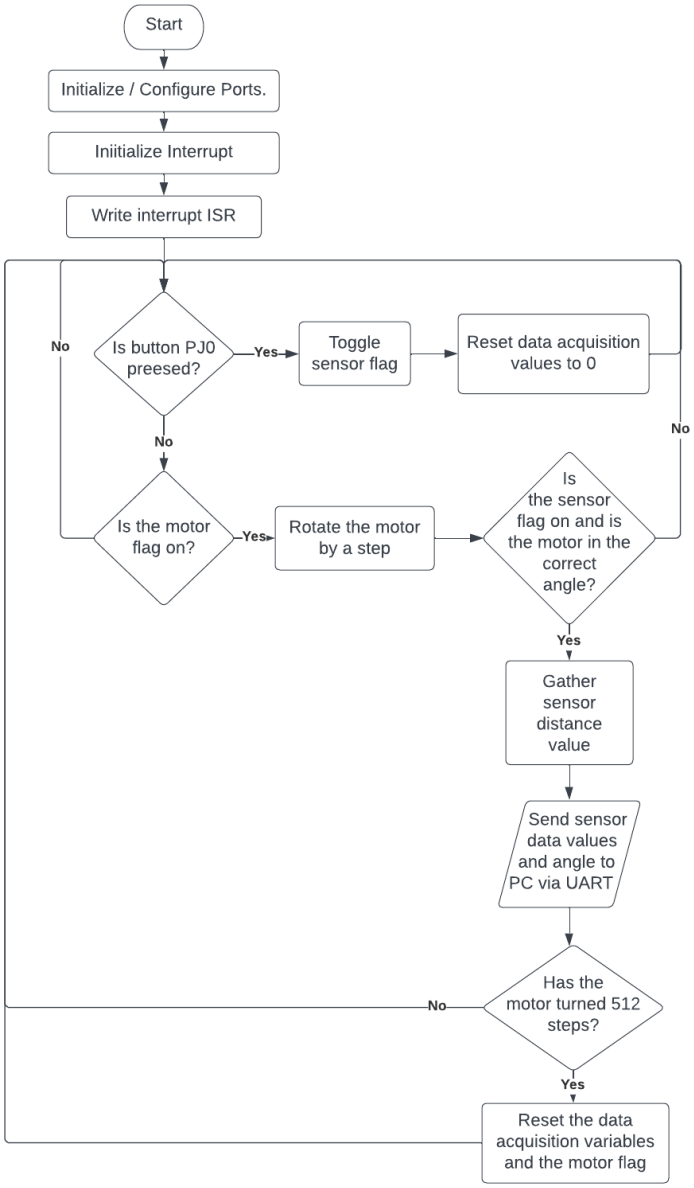

Figure 9: Flow chart of the programming logic in the microcontroller

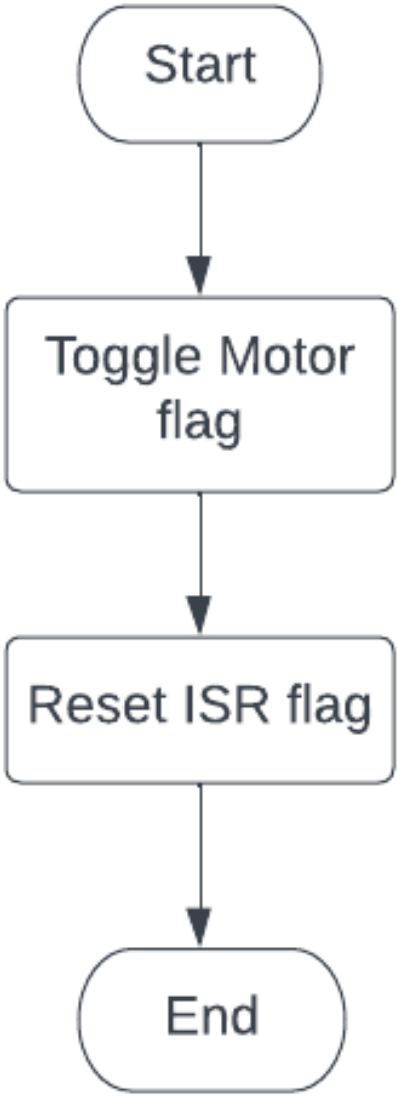

Figure 10: Flow chart of the interrupt programming logic in the microcontroller

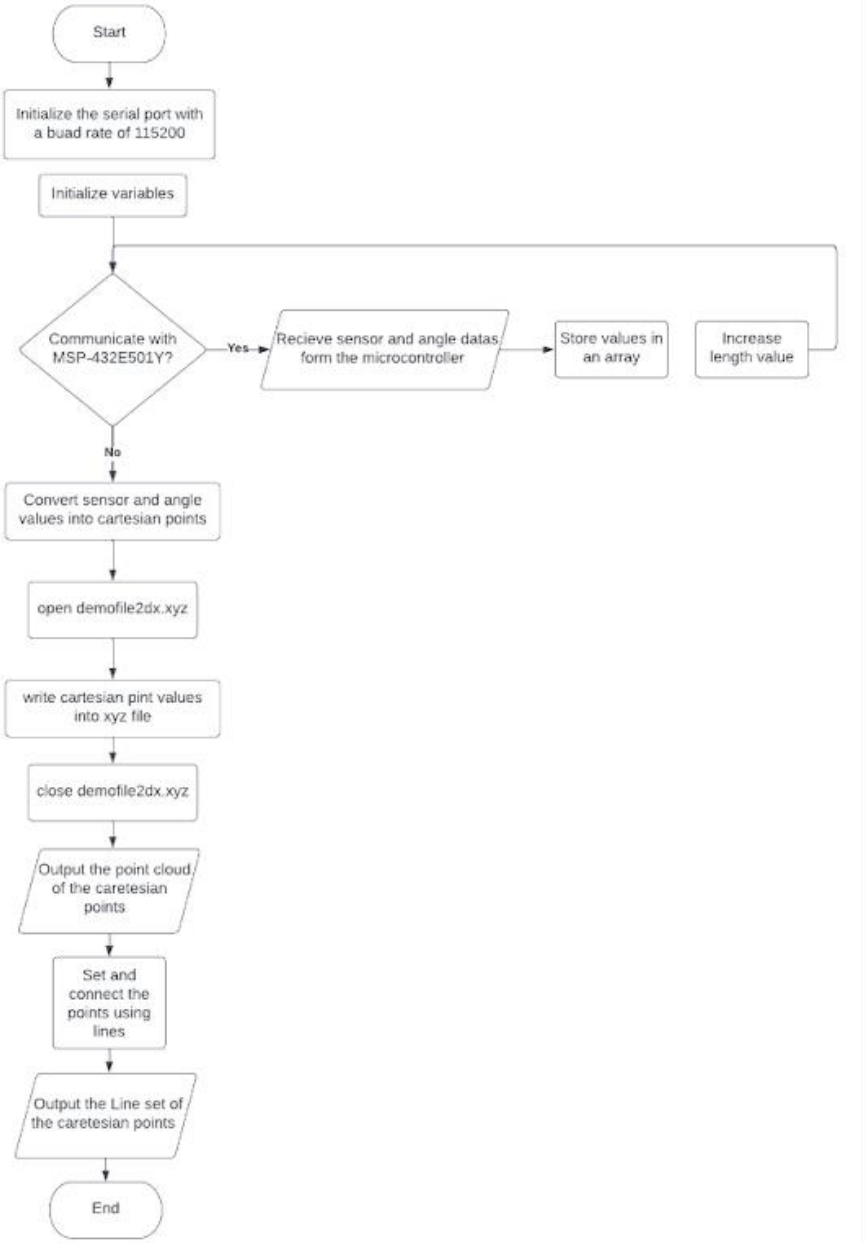

Figure 11: Flow chart of the programming logic in the Python script

Works Cited

[1] Texas Instruments, “MSP432E401Y SimpleLink™ Ethernet Microcontroller”, MSP432E401Y Datasheet, October 2017. [Online]. Available: https://www.ti.com/lit/ds/slasen5/slasen5.pdf?ts=1681656300702&ref_url=https%253A%252F%252Fwww.ti.com%252Ftool%252FMSP-EXP432E401Y.

[2] STMicroelectronics, “A new generation, long distance ranging Time-of-Flight sensor based on ST’s FlightSense technology”, VL53L1X Datasheet, November 2022. [Online]. Available: https://www.st.com/resource/en/datasheet/vl53l1x.pdf.

[3] “4 Phase ULN2003 Stepper Motor Driver PCB”, ULN2003 Datasheet. [Online] Available https://www.electronicoscaldas.com/datasheet/ULN2003A-PCB.pdf.

[4] N. Ojaokomo, “A step-by-step guide to python float(),” *HubSpot Blog*, 20-Jan-2023. [Online]. Available: https://blog.hubspot.com/website/float-python. [Accessed: 16-Apr-2023].