Design Overview

Features

- Texas Instruments MSP432E401Y Microcontroller
  - ARM Cortex-M4F Processor Core
  - Programmed in Assembly/C/C++
  - 12 MHz clock speed
  - 256KB of SRAM
- ULN2003 Stepper Motor Driver
  - 512 steps per rotation
  - 5V–12V operating voltage
  - LED indicators for the step state
- VL53L1X Time-of-Flight Sensor
  - Up to a 400cm range
  - 2.6V–3.5V operating voltage
  - Up to 50 Hz ranging frequency
  - 940nm invisible Class1 laser emitter
  - ±20 mm of ranging error
  - Size: 2.6V–3.5V operating voltage
- 3D Visualization
  - Uses PySerial, Numpy, and matplotlib modules
  - Written in Python 3.8
  - Updating in real-time as data is transferred
- Data communication
  - Baud rate of 115200 bps
  - I2C serial communication between time-of-flight sensor and microcontroller
  - UART serial communication between PC and microcontroller
- Cost
  - Analog Discovery 2 - $435
  - COMP ENG 2DX3/2DX4 Kit (Microcontroller, ToF sensor, stepper motor) - $189

General Description

This system is an embedded spatial measurement system that uses the VL53L1X time-of-flight sensor to collect data around it. This is done with the use of the ULN2003 stepper motor to allow for a 360-degree view. The information gathered by the time-of-flight sensor is internally processed then sent over to the PC via USB to construct a 3D visualization.

The system contains a time-of-flight sensor, microcontroller, and a stepper motor. The MSP432E401Y manages the entire embedded system, by initializing the entire system and distributes the power supplied from the PC to the other components. Furthermore, the microcontroller receives the processed data from the time-of-flight sensor and transfers it to the PC through a USB.

The VL53L1X time-of-flight sensor uses infrared lights to determine the distance between an object and a sensor. This is done by measuring the time it takes for the pulse of light

to travel to the object and back to the sensor. The system then calculates the distance by converting the value from analog to digital. After the data is ready, it is sent to the microcontroller through I2C communication.

The microcontroller is connected to the PC through UART communication via USB. Once the python program is run, the microcontroller will transfer the data to the PC which allows the python code to translate the data into x-y-z components that can be visualized.

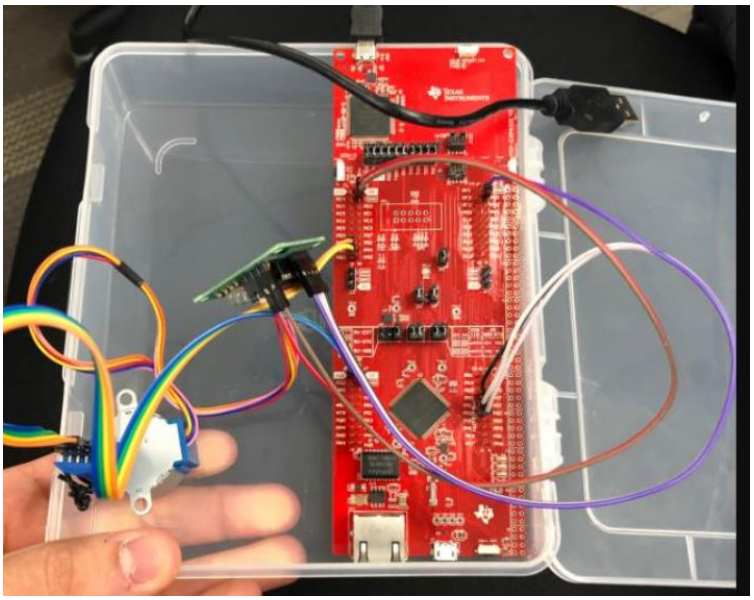

Figure 1: Embedded System. Microcontroller on the right, stepper motor with ToF sensor mounted on the left

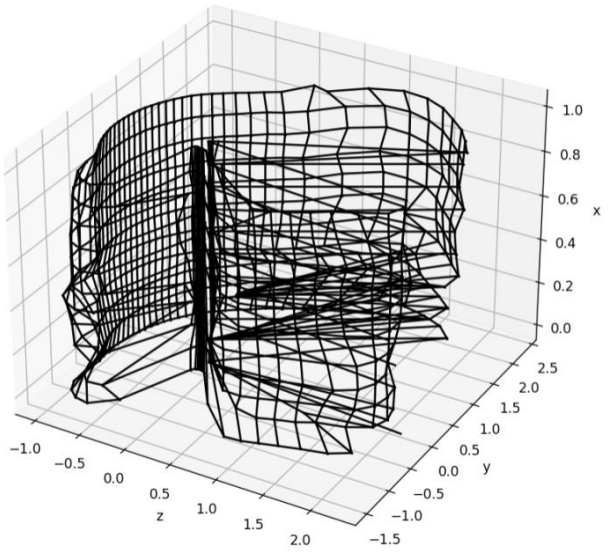

Figure 2: Screenshot of 3D visualization

Block Diagram

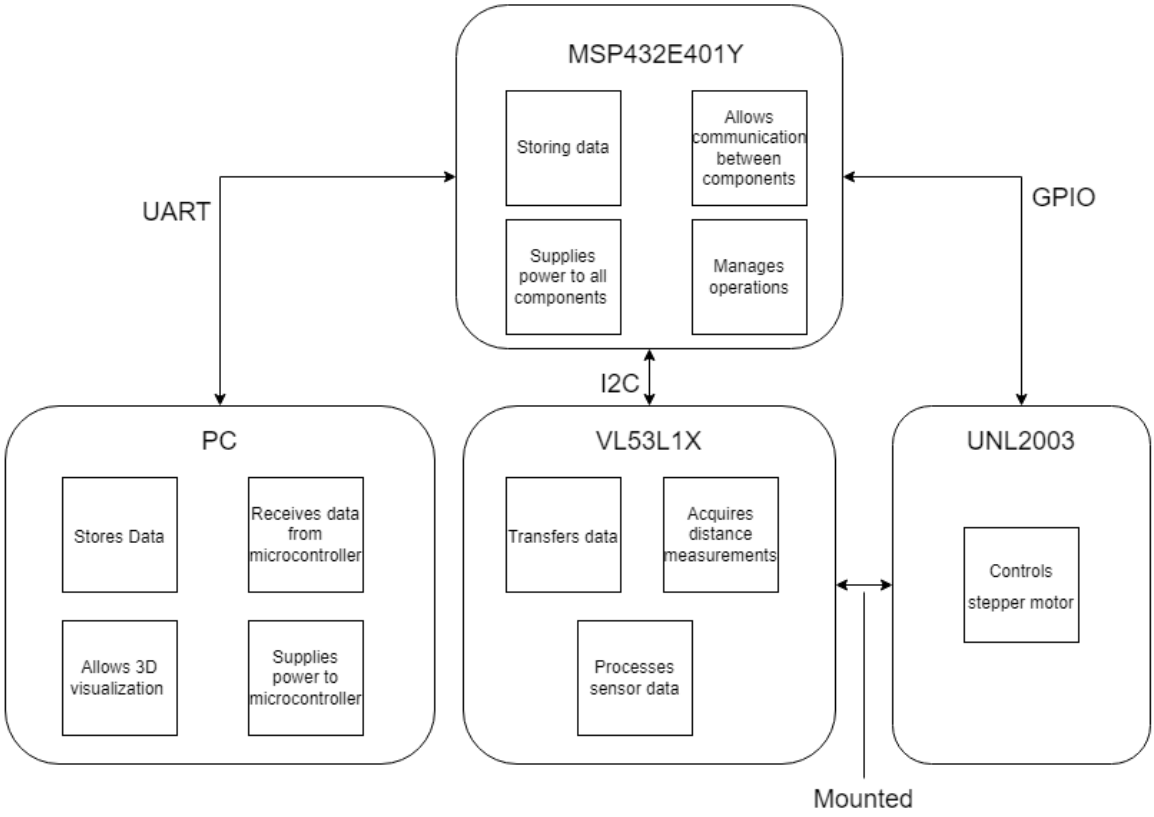

Figure 3: Block Diagram

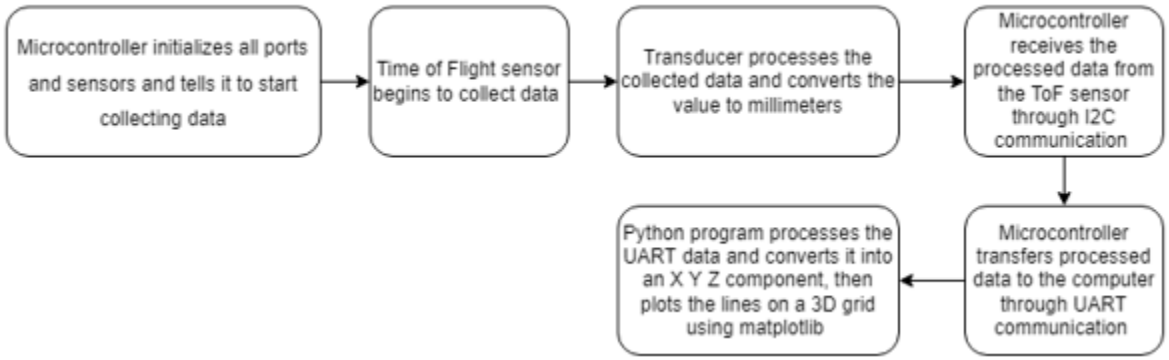

Figure 4: General Description

Device Characteristics

Features	Detail
Bus Speed	50MHz
Serial Port	COM4 (UART)
Baud Rate	115200 BPS
Python Version	Python 3.8
Libraries	Numpy, Matplotlib, PySerial
VL53L1X	
Sensor	Microcontroller
V_{DD}	
V_{IN}	3.3V
SDA	PB3
SCL	PB2
GND	GND
GPIO1	
XSHUT	
ULN2003	
Stepper motor	Microcontroller
IN1	PE3
IN2	PE2
IN3	PE1
IN4	PE0
+	5V
-	GND

Table 1: Technical Specifications

Detailed Description

Distance Measurement

The distance measurements are completed by the VL53L1X time-of-flight sensor to gather data about the area surrounding it. With the use of the ULN2003 stepper motor, the time-of-flight sensor can acquire 360-degree measurements of distance in one geometric plane.

The VL53L1X time-of-flight sensor radiates infrared light and determines the distance to an object by measuring the time it takes for the light to travel to the object and return. The general equation for time-of-flight sensor is:

$$D = \frac{t}{2} \cdot c$$

Where D is the measured distance, c is the speed of light, and t is the time of travel of light. Once the sensor gathers the data, it calculates the distance by converting the time value gained from analog to digital then it is processed to check if it is within the range. If it is a valid number, the sensor sends this data back to the microcontroller through I2C communication.

The time-of-flight settings that are used for the initialization are the default settings which are provided in the VL53L1 core API found in the user manual. Furthermore, all the time-of-flight functions that are used in this system are also provided in the VL53L1 core API provided by STMicroelectronics as well as in the code provided by the course coordinators. The implementation of the code and initialization are unmodified from the code provided.

When initialization on the microcontroller begins, the I2C functionality within the microcontroller is enables, which allows the microcontroller to communicate with the time-of-flight sensor. Furthermore, all necessary variables used to hold distance data and any time-of-flight functions are also initialized. When this is complete, the time-of-flight sensor starts up and begins ranging. However, nothing is being done with the data until the button is clicked which initiates the systems beginning.

The button located on the microcontroller GPIO PJ1 is initialized and set as a toggle to start and stop the system from capturing data via time-of-flight sensor. In neutral position, the button is set to true which means that the system is not ready to gather data from the time-of-flight sensor. When the button is clicked, the AND operate causes **GPIO_PORTJ_DATA_R** to set to 0, which allows the system to begin and gather data from the time-of-flight sensor.

When the button is clicked and the system is ready to gather data from the time-of-flight sensor, the program waits for the sensor to be ready by using one of the VL53L1 functions, then extracts the range status and distance measurements when the data is ready. When this is complete, it clears the interrupt set on the time-of-flight sensor which is also another function provided in the VL53L1 user manual. The communication between both embedded components

is possible by the 12C protocol that is enabled on both the microcontroller and time-of-flight sensor. The data gathered from the time-of-flight sensor is then stored on the microcontroller and then transmitted to the PC through UART communication and USB port. The processed data is then received by the PC running the python code. Once the for loop is completed which allows the stepper motor to travel 512 steps, another for loop is executed which changes the boolean variable **direction** to false, which causes the stepper motor to return to its neutral position. This is implemented so any wires are not at risk of being damaged due to continuous spinning.

The python code receives the data sent from the microcontroller through a USB by using the PySerial library in python. The data is received in a .txt file, where the values are separated and turned into a float. Once this is complete, the x and y components of the measurements are calculated with basic trigonometry functions. After each iteration, the z value which represents the depth of the plot is incremented by 0.1 units. As part of the KEIL code, I determined that 8 steps is equal to 5.625 degrees by using the ratio of 512 steps : 360 degrees. However, in the python code that number had to be changed to radians, so a value of 0.09817477 was used. This value was multiplied with the **step** counter variable that held the number of how many steps the motor has turned. When working with the x component of the measurement, the equation to calculate the angle was:

$$x = D \cdot cos(step \cdot \theta)$$

While the y component had an equation of:

$$y = D \cdot sin(step \cdot \theta)$$

Visualization

The 3D visualization of data is done by reading the data transferred by the microcontroller through UART communication and USB, writing it to a .txt file, and then using numpy and matplotlib library functions to visualize the data.

The visualization of the data was performed on a HP Pavilion x360 - 15-dq1003ca with the following specs:

Hardware	Specifications
Microprocessor	Intel® Core™ i7-10510U (1.8 GHz base frequency, up to 4.9 GHz base with Intel® Turbo Boost Technology, 8 MB cache, 4 cores)
Chipset	Intel® Integrated SoC
Memory	16 GB DDR4-2666 SDRAM (2 x 8 GB)
Video Graphics	Intel® UHD Graphics
Hard Drive	512 GB PCIe® NVMe™ M.2 SSD
Display	15.6" diagonal FHD IPS micro-edge WLED-backlit multitouch-enabled edge-to-edge glass (1920 x 1080)

The IDE I used to perform the 3D visualization is IDLE using python 3.8.11 The code requires the installation of PySerial, NumPy, and Matplotlib to read, write, and visualize the measurement data. It also requires the math built-in library to manage the data. Once the setup is ready, the system is ready to collect data from the microcontroller and begin visualization.

When the program begins running, an I/O file is opened and extracts the values and converts them into float values, so no error occurs when performing the calculations later in the program. After each value is extracted, a '\n' is written which places the next value that is measured in a new line. Once all data is extracted and organized, the system checks the number of data points and stores it in a variable called **measure_num**, as well as ensure that all values in the list are converted to a float. Once all points have been converted into floats, basic trigonometry functions can convert the data into x and y components, which are then plotted using matplotlib functions. The figure below shows the path the sensor takes points at, which is every 5.625 degrees (8 steps).

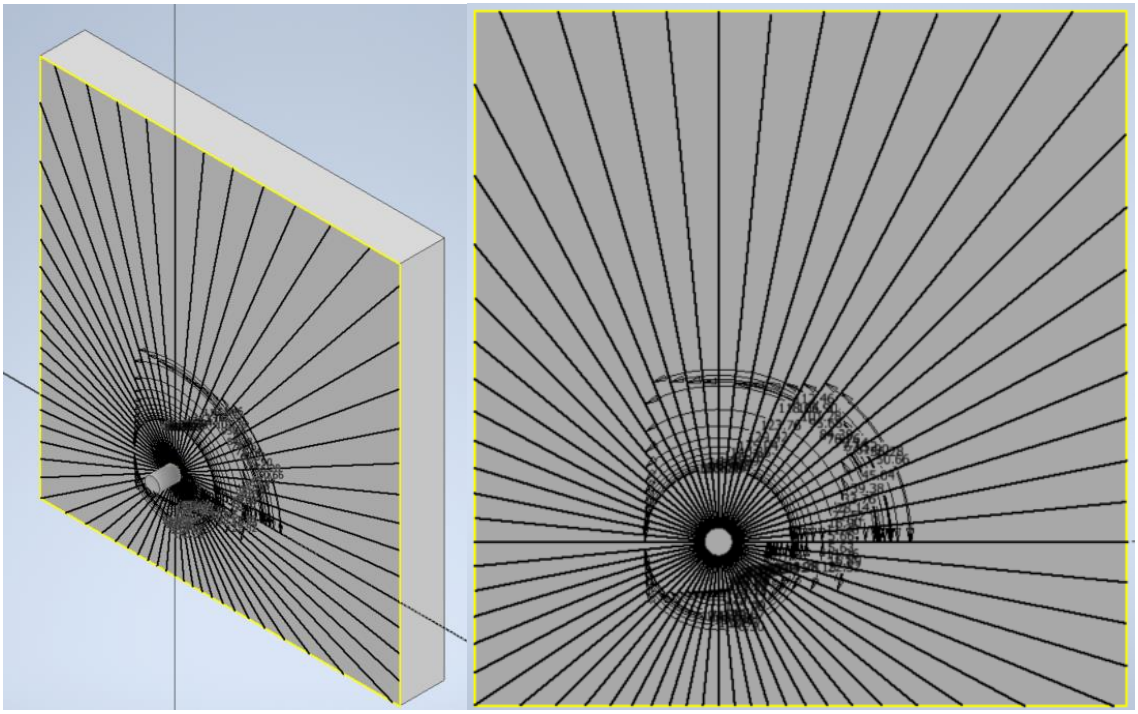

Figure 5: Demonstration of ToF sensors' path

The dimension of the hallway is shown below, in respective to the ToF sensor. The actual value of the hallway is roughly a factor of 200x greater than the sketch.

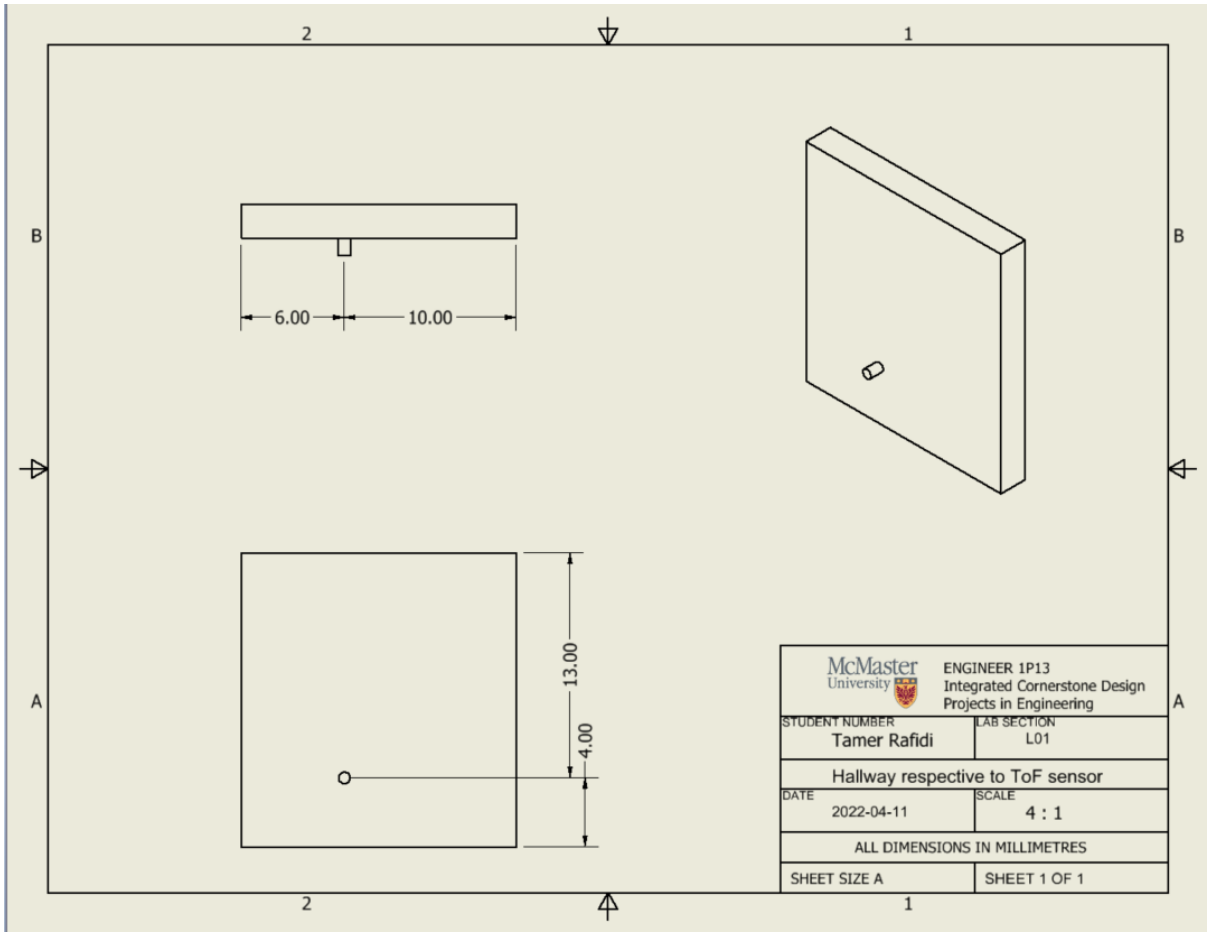

Figure 6: Dimension of the hallway in respective to ToF sensor

Application Example

To use the embedded system, there is a setup that will ensure all necessary software is installed on the user's computer. All steps are important for the user's computer to receive data from the microcontroller and to 3D visualize the data. Note: This setup process is done on Windows 10.

1. Download and install KEIL uVision5 from https://www2.keil.com/mdk5 and download MDK. Complete the entire process then open the downloaded files. When the installation is complete, the pack installer should pop up. Click "Devices" tab, and search for MSP432E401Y. Install the family pack then close the installer.

2. Download and install Python 3.8.11 from https://www.python.org/downloads/. Scroll down and look for the 3.8.11 release. Run the setup and make sure to add it to the path.

3. Once Python in installed, download an IDE that is capable for python. I used IDLE which is included in the default python installation. Once the download is complete, make sure to install it and make it your python IDE.

4. Once you are ready, open command prompt in administrator mode by right clicking "Command Prompt" and then clicking "Run as administrator." Once the command prompt is running, you will need to install PySerial, NumPy, and Matplotlib. To do this, you must enter "pip install pyserial." Wait for a confirmation message saying that all packages have been installed correctly. Similarly, complete this process for "python -m pip install -U matplotlib," and "python -m pip install -U numpy" to install NumPy and Matplotlib.

Once you have completed the setup, your computer is now capable to interact with the system. You are now ready to collect data and begin to visualize the data. The process goes as follows:

1. Download the zip folder and unzip the components. Once you reach the python file, right click the file and open it with the IDE of your choice. If you are using IDLE, right click the file and choose "Edit with IDLE."
    a. In line 24, you will need to change the COM port being used. To determine the COM port that is being used for UART communication, search for "Device Manager" and plug in your microcontroller. Once it is plugged in, drop down the menu "Ports" and the COM port can be found listed next to "XDS110 Class Application/User UART".
    b. The file where the data is being stored can be changed to your desired file name. This can be found on lines 20, 39, and 48.
    c. The increment of z displacement can be changed by adjusting the value in line 65.
    d. The baud rate can be edited in line 24

2. Connect the microcontroller to the PC via micro-USB port on the other side of the ethernet port.
3. When aligning the time-of-flight sensor, keep in mind that the data will not be affected if you are moving the system in reference to how it is oriented. For reference, the positive y-axis is the direction where the time-of-flight sensor is initially facing, the x-axis is the plane the stepper motor rotates on, and the z-axis the direction the tip of the stepper motor points towards. The z-axis is also the direction that the system moves in after every consecutive iteration.
4. Open the file in KEIL and then click the “Translate”, “Build”, and “Download” buttons in that order. Once that is complete, click the reset button which is next to the micro-USB port on the microcontroller.
5. As the microcontroller is downloading the information, open the python file and run the program. Once the LED’s on the microcontroller stop flashing, answer the prompt on the computer while simultaneously clicking GPIO PJ1 button on microcontroller which is located on the side.
6. Wait until the time-of-flight sensor has made a 360-degree turn then answer the prompt on the computer to decide whether you want to plot the graph or not. If you answer “y” for yes, wait a few seconds for the 3D visualisation to be processed and outputted.

Limitations

1. The Floating-Point Unit (FPU) in the MSP432E401Y is the component that carries out all floating-point operation partaking in the microcontroller. The unit can support 32-bit addition, subtraction, division, multiplication, multiple-and-accumulate, and square root division. Furthermore, it can also perform conversions between floating-point data types and floating-point constant instructions. It also supports trigonometric functions from the math.h library that can be applied on float variables. Since the FPU is 32-bits wide, 64-bit operations will require more time since it needs more than 1 instruction to complete.
2. Maximum quantization error for the ToF module is $\frac{4000mm}{2^{16}} = 6.104x10^{-2}mm$
3. The maximum serial communication rate that can be implemented on a PC is 128000 bits per second. This can be verified by connecting the microcontroller to the PC, then following these steps: Device Manager -> Ports -> XDS110 Claas Application/User UART (COM4) -> Port Settings -> Bits per second.
4. The microcontroller and the time-of-flight sensor module communicate with each other through I2C protocol with a clock speed of 100KHz and a transfer rate of 100kbps.
5. When analyzing the whole system, it is clear the ToF sensor is the primary limitation for speed. The ToF sensor has a sampling rate of 50Hz which means that it maxes out at 50 scans per minute.

Circuit Schematic

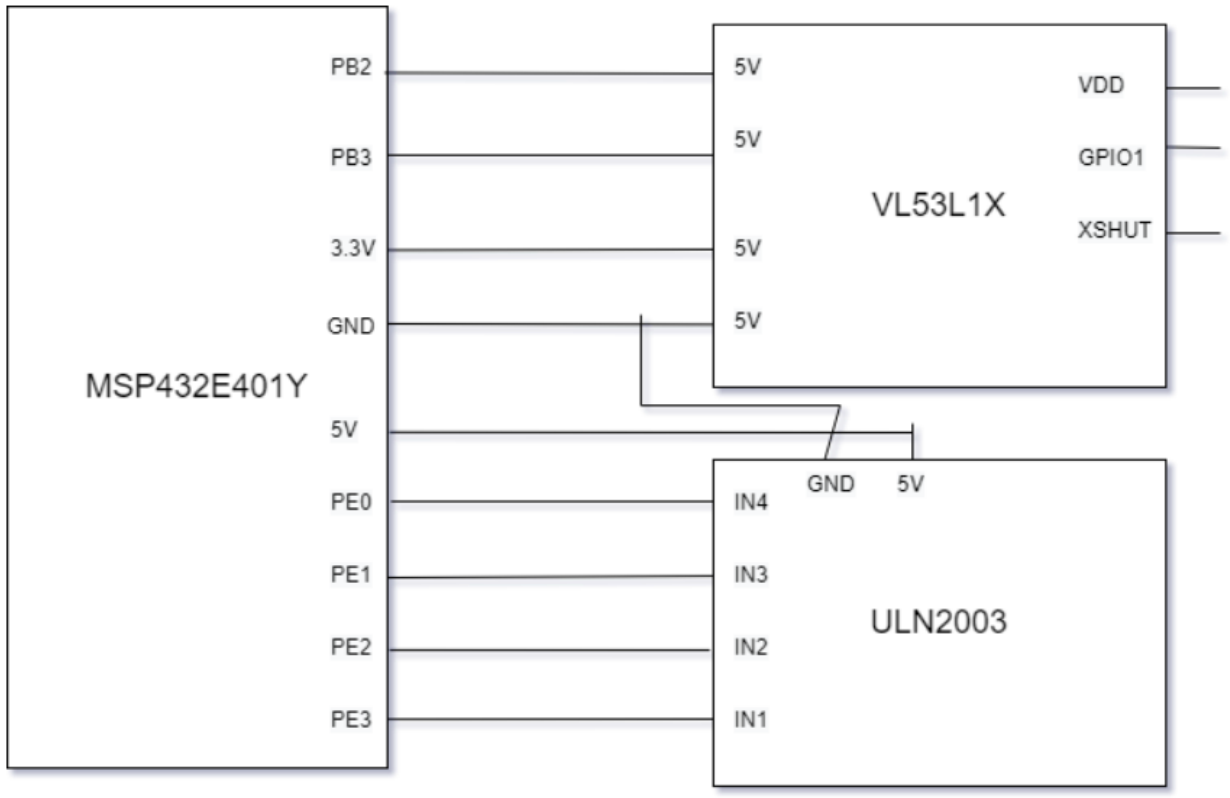

Figure 7: Circuit to Schematic

Flowcharts

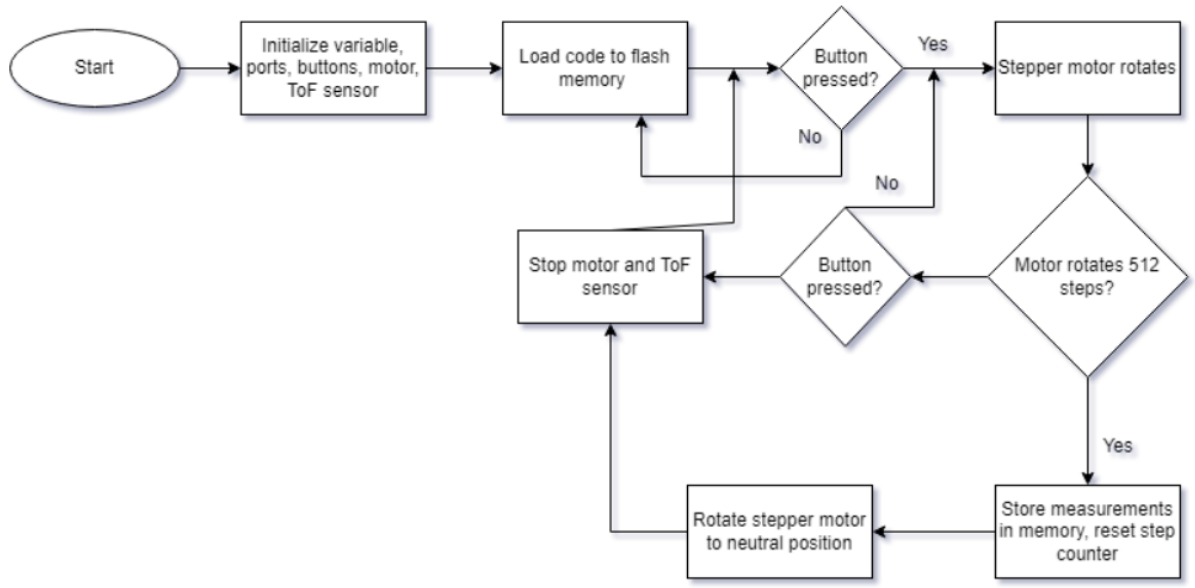

Figure 8: Microcontroller flowchart

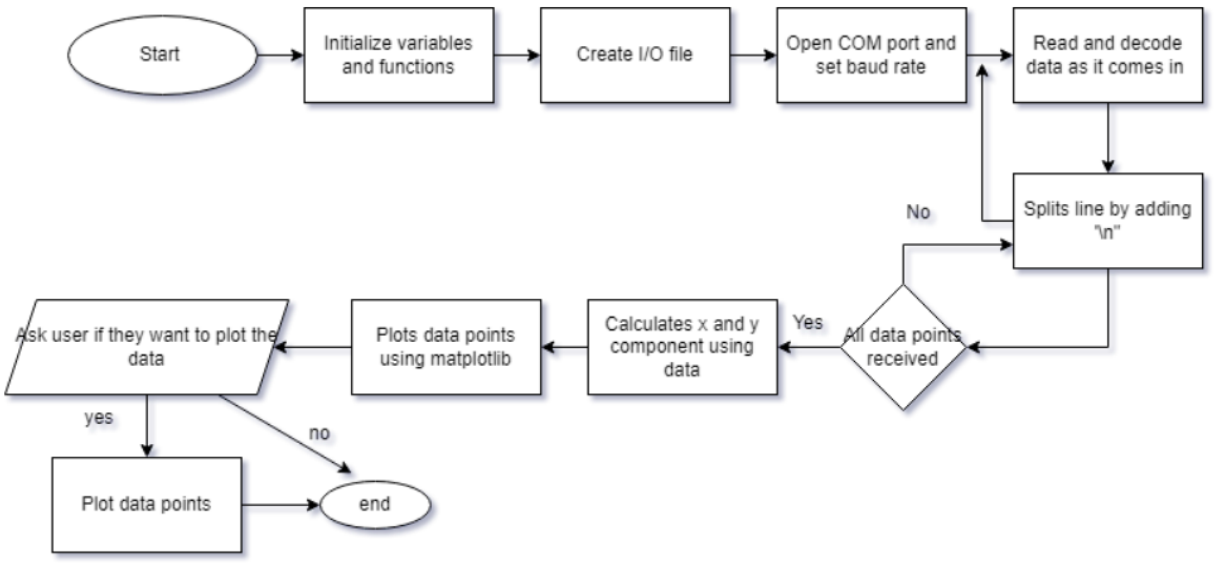

Figure 9: Python flowchart