2DX4: Microprocessor Systems
Final Project

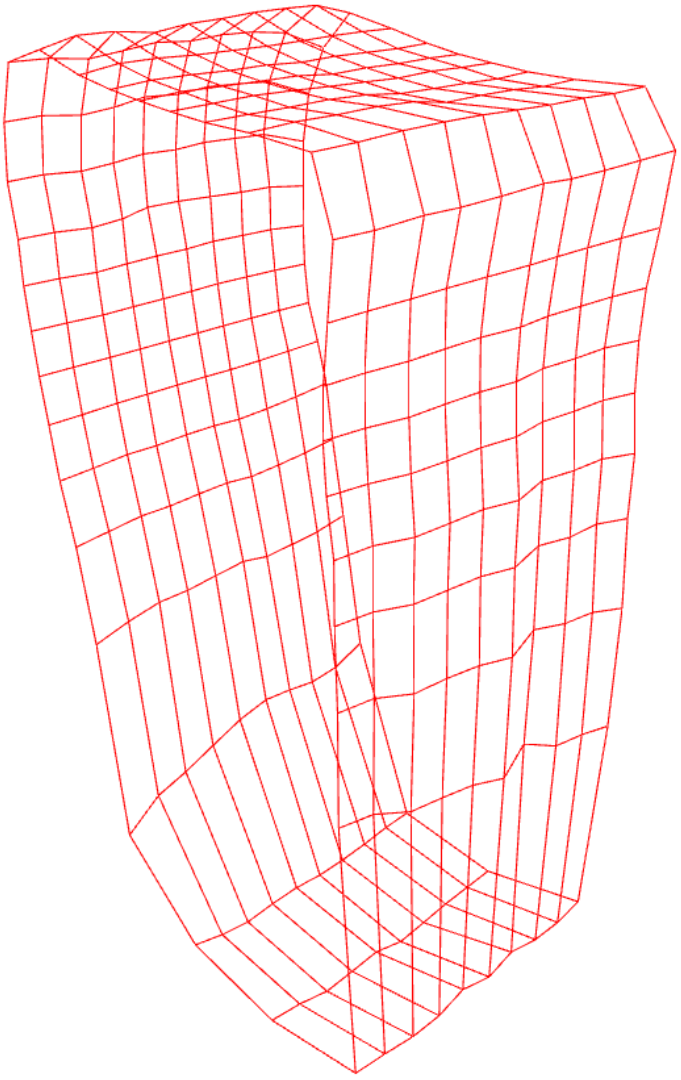
Instructor: Drs. Doyle, Haddara, and Shirani

Abdelmoniem Hassan – L05/L06 – hassaa73 - 400248003

As a future member of the engineering profession, the student is responsible for performing the required work in an honest manner, without plagiarism and cheating. Submitting this work with my name and student number is a statement and understanding that this work is my own and adheres to the Academic Integrity Policy of McMaster University and the Code of Conduct of the Professional Engineers of Ontario. Submitted by **[Abdelmoniem Hassan, hassaa73, 400248003]**

Videos

Demo and Questions:
https://drive.google.com/file/d/1uTyftaKfEKlYSDEQAITQAFTd-OePyrk3/view?usp=sharing



1. Device Overview

a. Features

- SimpleLink™ Ethernet MSP432E401Y Microcontroller LaunchPad™.
    - 60 MHz clock speed.
    - Drives stepper motor and ToF sensor.
    - Onboard LED to signal successful data transmission.
    - Programmed using C.
- 28BYJ-48 Stepper Motor and ULN2003 Driver;
    - Operated with a 5V supply
- VL53L1X Time-of-Flight Sensor.
- Single Button Interface.
- Serial Data Communication I2C to microcontroller, UART to computer.
    - 115200 baud rate.
    - CR/LF Terminator.
- USB A 2.0 interfacing with computer.
- 3D Visualization/Rendering.
    - Written in Python, used Open3D for visualization

b. General Description

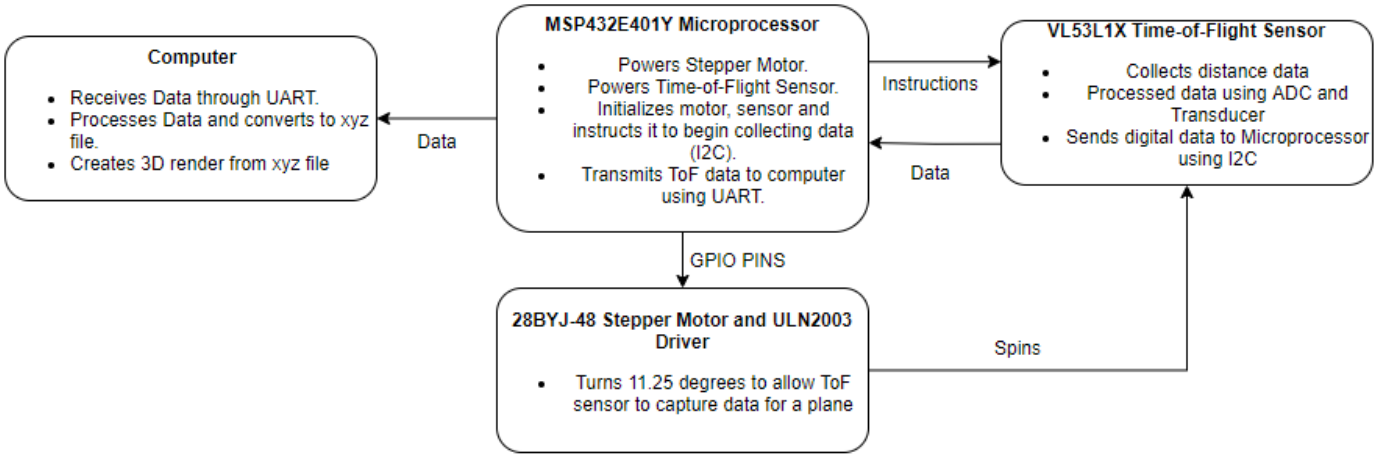
The embedded spatial measurement system generates a 3-Dimensional render of the surrounding area using the VL53L1X Time-of-Flight sensor. By attaching the sensor to the 28BYJ-48 Stepper Motor we are able to capture 32 points on the Y-Z plane, and by moving the system on the X axis and taking multiple scans, a 3D render can be created. The system consists of the VL53L1X Time-of-Flight sensor, the 28BYJ-48 Stepper Motor, the MSP432E401Y microcontroller and a laptop.

The Time-of-Flight sensor captures distance data using LIDAR technology. It uses infrared light as a pulsated laser to measure the distances, this is done by calculating the amount of time required for the emitted light to return to the sensor. The sensor's internal Analog-to-Digital converter converts the analog data to digital data to be sent to the microcontroller. The 28BYJ-48 Stepper Motor rotates 11.25 degrees between scans, allowing the sensor to capture 32 data points.

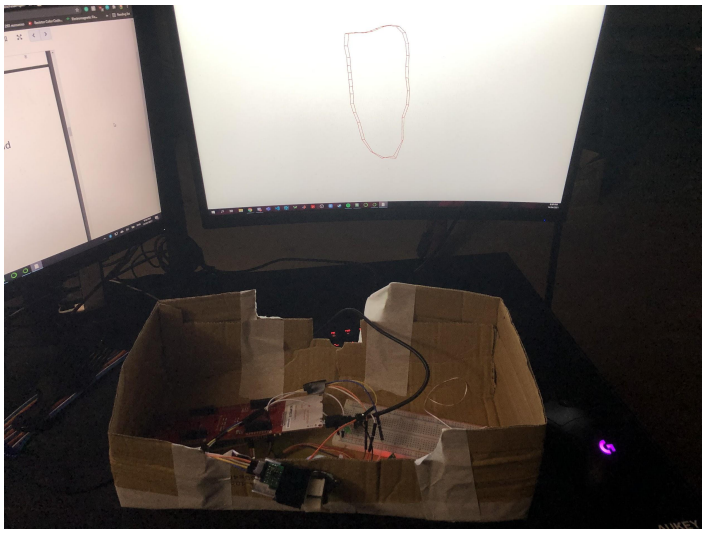
The MSP432E401Y microcontroller powers and controls the stepper motor as well as the ToF sensor. The distance data is communicated to the microcontroller using 12C from the ToF sensor; This is then communicated to the computer using UART and a baud rate of 115200.

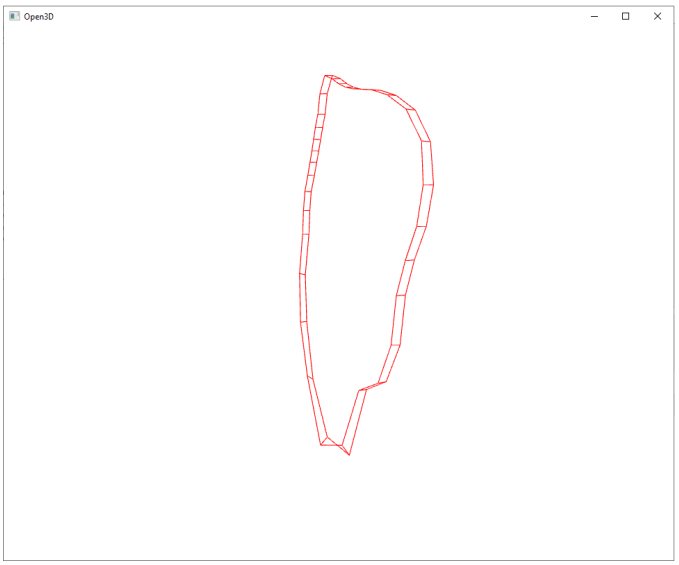
The distance data is read byte by byte and collected together until a newline/enter character is received, the distance data is then parsed and converted from cylindrical coordinates to xyz coordinates, this data is then written to a .xyz file. The xyz file is then read in by the script, and a mesh is created by connecting points within an individual plane together then connecting the following planes together. A 3D render is then created using Open3D for visualization.

c. Block Diagram



- Device Images.





*This is a sample output and not the final output of the project the final render of the room is in the video

2. Device Characteristics Table

Device	Specification	
MSP432E401Y Microcontroller	Bus Speed	60 MHz
	Baud Rate	115200 bps
Computer	Serial Port	COM3*
Software	Python	3.7.4
	Open3D	0.7.0.0
	Numpy	1.16.5
	Pyserial	3.4
VL53L1X Time-of-Flight Sensor	**Pins**	**Microcontroller**
	VDD	null
	VIN	3.3V
	GND	GND
	SDA	PB3
	SCL	PB2
	XSHUT	null
	GPIO	null
28BYJ-48 Stepper Motor and ULN2003 Driver	**Pins**	**Microcontroller**
	VIN	5V
	GND	GND
	IN1	PL0
	IN2	PL1
	IN3	PL2
	IN4	PL3
Extenal Button Input		PE0

3. Detailed Description
    a. Distance Measurement

The VL53L1X Time-of-Flight Sensor is the device computing the distance. The Time-of-Flight sensor using Light Detection and Ranging (LIDAR) technology to determine the distance. The sensor emits an infrared light signal towards the target, the reflection is then captured by the sensor, by calculating the time between the signal being sent out and it being measured again by the sensor we get the Time-of-Flight. Using the equation

$$D = (ToF/2) * \text{Speed of Light}$$

We can determine the distance between the sensor and the object. The sensors internal Transducer and Analog-to-Digital Converter (ADC) convert the result into a digital signal ready to be communicated to the microcontroller. The data is communicated to the microcontroller

using I2C. The MSP432E401Y microcontroller is first initialized setting up all the relevant GPIO ports, the I2C, and UART functionality, as well as the ToF sensor using the functions from its API. The microcontroller then polls for an input through PE0 which is connected to an active high button which controls data acquisition. Once the button has been pushed the microcontroller then sends "StartRanging" through UART to indicate to the script running on the computer that data points will be sent in, and then starts ranging the ToF sensor, gathering a datapoint, sending it out through UART, and then rotates the Motor 11.25° this process takes place 32 times, to gather 32 data points through 360°. Once this process is completed the "break" keyword is sent to the computer to indicate that there are no more data points for this plane, and the microcontroller returns to the polling state, waiting for an input from the start button.

The serial communication is received by the script with the help of the pyserial module which allows python to read serial communication on the computer's ports. This data is sent by the microcontroller through UART meaning it is sent byte by byte. The python program appends everything to a string until a '\n' newline character is received and each string is appended to a list containing all the data from the plane being scanned. This gives us an array with all the communication from the microcontroller for a particular plane. The actual data points are then parsed out and converted from cylindrical coordinates to cartesian ones. The ToF gives us the distance as a radius from the sensor. The angle (θ) can be determined by the index of the radius within the data string,

$$i \text{ is the index of the radius within the data array; } \theta = (i+1)*pi/16$$

the z coordinate is a function of the plane currently being scanned (first one is zero, second one is 10cm, ...). The cartesian coordinates can then be obtained using

$$X = R*\cos(\theta),\ Y = R*\sin(\theta),\ Z = z$$

This data is then appended to the "tof_radar.xyz" file and is ready for visualization. After 10 sets of data have been received by the computer, the data is visualized into a 3D render.

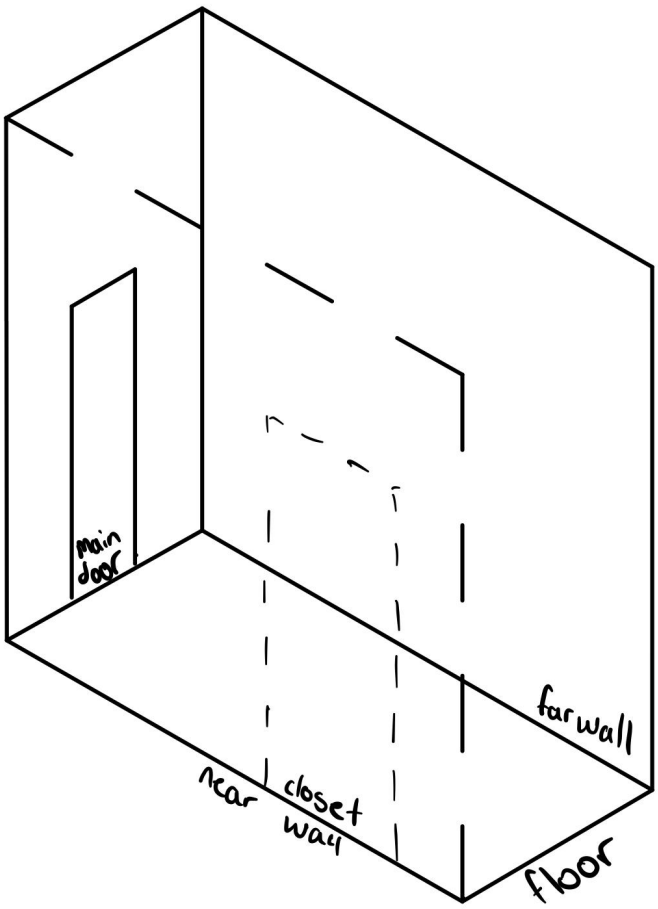
b. Visualization

This simulation was run on an HP Pavilion Laptop 15-cw1 with a AMD Ryzen 5 3500u central processing unit and a Radeon Vega Mobile built in graphics card. This program was tested on Windows 10, but will work for Mac OSX, and Linux. The python script was tested through JupyterLab, with pyserial version 3.4, numpy version 1.16.5, Open3D version 0.7.0.0 and Python 3.7.4. This script is supported until Python 3.8.8.

The 3-Dimensional coordinates are written to the .xyz file in the format of "x y z\n". This data is read in and stored in a 2-Dimensional array with each sub array containing all 3 coordinates for a single point. The 3D rendering of the area takes place by reading in the processed .xyz file. The data is read in once again using the open3d.io.read_point_cloud method which converts an xyz file into a 3D point cloud. A lines array is then created and appended with the indices of points that need to be connected using lines. Once this stage is completed, an Open3D line_Set object is initialized with the points, and the lines array this object can then be

plotted using the Open3D built in method draw_geometries.

- Isometric View of Hallway Scanned



4. Application Example with Expected Output

This system requires a computer with a USB A port capable of delivering 5V. It is recommended that you use the same software listed below as the program has only been tested on them.

1. Install Anaconda to your device from https://www.anaconda.com/products/individual This is the IDE of choice and the next steps will describe how to install packages within anaconda, however any IDE can be used.
2. Launch The JupyterLab IDE. Anaconda comes preinstalled with python, if a different IDE is used install Python 3.7.4 from https://www.python.org/downloads/

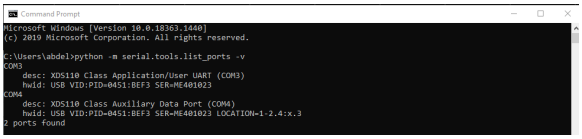
3. Open an new notebook file and run

```
!pip install pyserial==3.4
!pip install open3d open3d-python
```

4. Plug in the the system through a USB A port, Open the Command Prompt and run

```
python -m serial.tools.list_ports -v
```

Note the name of the UART port in this example the correct port is COM3.



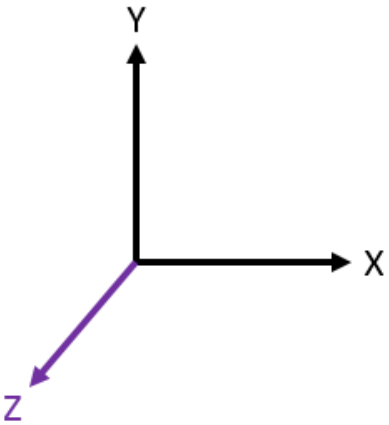
5. Modify line 11 in the python script to match the correct port of your device.

```
1  import serial
2  import math
3
4  print('start')
5  #Modify the following line with your own serial port details
6  #   Currently set COM3 as serial port at 115.2kbps 8N1
7  #   Refer to PySerial API for other options.  One option to consider is
8  #   the "timeout" - allowing the program to proceed if after a defined
9  #   timeout period.  The default = 0, which means wait forever.
10
11 s = serial.Serial('COM3', 115200)
```

6. Run the python script and wait for ‘Opening: COM3’ to be outputted, once this is done you are ready to begin collecting data. Locate the RESET button on the microcontroller and press it once, then locate the external push button on the system and press it once.

   The LED associated with the external button will light up to indicate that the button has been pushed and that the program has started collecting data. The LED PN1 will blink every time a datapoint is successfully collected. The motor will then spin 11.25° degrees to collect the next data point.

   The ToF sensor will collect 32 points and then complete another 360° in the opposite direction to prevent wires interfering with the scan. Once the process is completed the program will output ‘break’, this indicates the system is ready to collect data for the next plane displaced 10cm forward in the Z direction. The next set of data can be collected simply by pressing the external button again.



The diagram on the left indicates how the XYZ plane is defined with respect to the systems measurements. The vertical slices are along the XY plane and the forward displacement is along the Z axis.

7. After 10 scans the program generates a 3D render using the Open3D module, this will open a new window with the visualization.

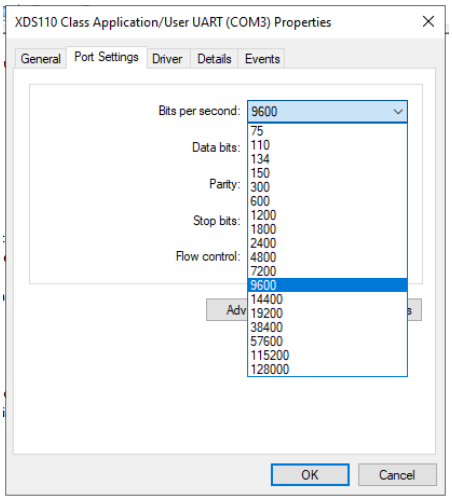
5. Limitations
    a. The MSP432E401Y microcontroller uses a 32-bit ARM Cortex M4 which supports a single precision add, subtract, multiply, divide, multiply-accumulate and square root operations. It is a EEE754-compliant single-precision floating-point unit. It also has 32 dedicated 32-bit single precision registers. Trigonometric calculations can be computed on the microcontroller through the math header file, however for this program all computations are computed on the computer within the python script.
    b. The maximum quantization error for each ToF module is
        - Distance

$$\frac{4m}{2^{16}} = 6.10 \cdot 10^{-5}m = 0.061mm$$

        - Voltage

$$\frac{5V}{2^{16}} = 7.63 \cdot 10^{-5}V = 0.076mV$$

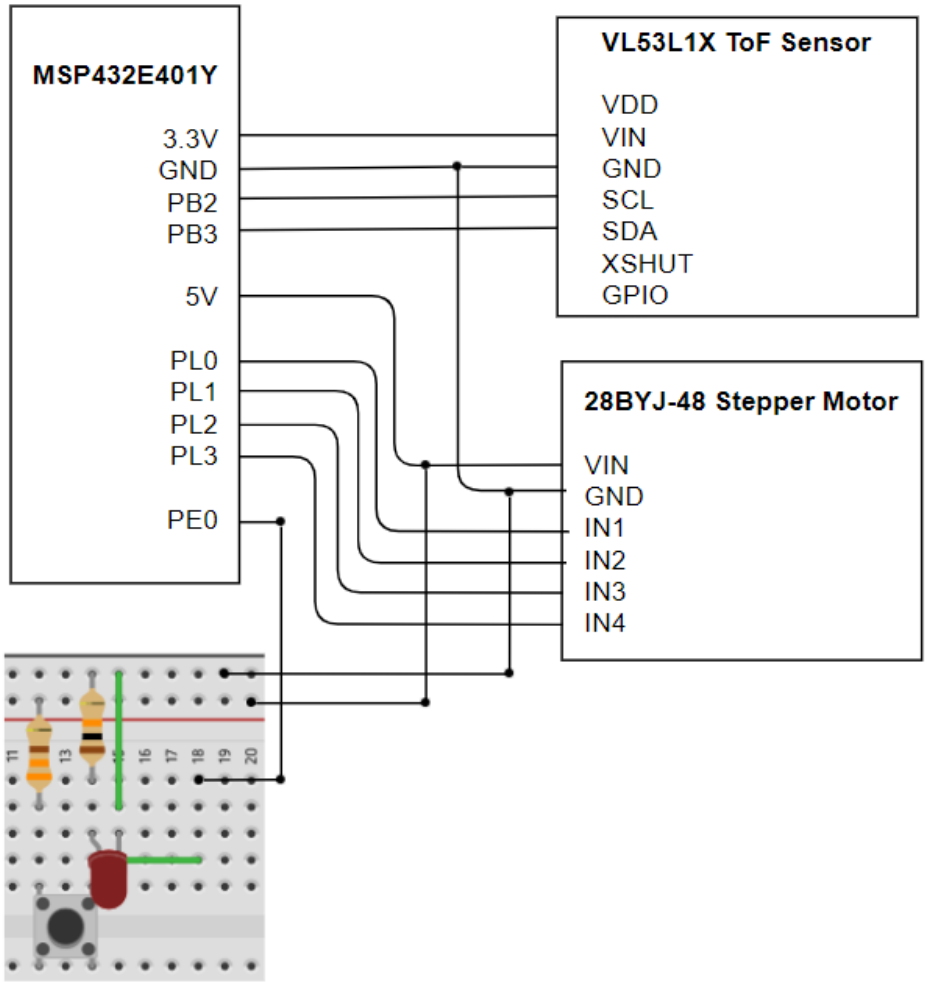
    c. The maximum standard serial communication rate that can be implemented on this computer is 128000 bps. This has been verified through the device manager.



    d. The microcontroller and the ToF sensor communicate through I2C with a clock speed of 100kbps.

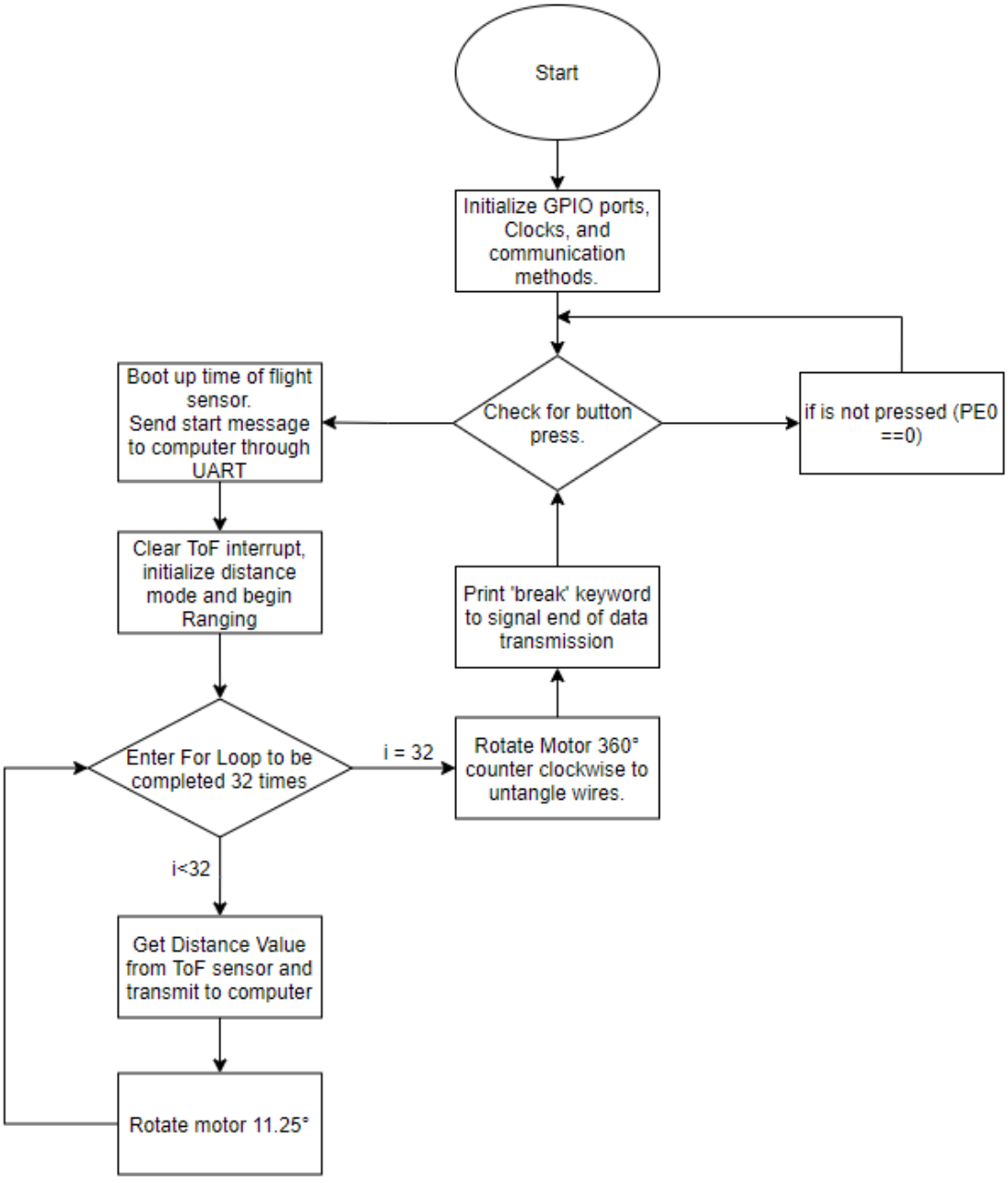
    e. The limiting factor for the speed of this system has been the stepper motor. Due to the physical connections of the system the stepper motor has to take an extra rotation to prevent the wire from getting tangled together and stopping the system from working all together. If the build quality was improved and the extra rotation was not required, the ToF sensor would be the limiting factor on the speed of the system. The ToF sensor requires a large delay for the long distance measurements required for this project, which builds up quickly as the number of points per plane increases. Trying to reduce this delay renders the ToF sensor useless as it is no longer capable of calculating the distances.

6. Circuit Schematic



7. Programming Logic Flowchart

- Microcontroller



- Python Script