2DX4: Microprocessor Systems
Final Report

Instructors: Drs. Boursalie, Doyle, and Haddara

Mit Patel – patem70 – 400261761 – 2DX4 – Wednesday – L03

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 **[Mit Patel, patem70, 400261761]**

Device Overview:

Features:

- A fully functioning and cheap lidar system that can be used anywhere and map anything within a 4 meter distance and create a 3D projection of the mapped area
- The heart of the system is the TI MSP432E401Y Microcontroller which connects the Stepper Motor and the VL53LIX ToF sensor with the software
- Stepper Motor allows the system to spin and measure distances through the ToF sensor in a 360 degree range of motion
- The ToF sensor can measure distances accurately up to 4 meters away in low ambient light and high reflective surfaces
- The system is programmed using C language in Keil and the 3D projections of the area and mapped using Python
- I2C communication established between ToF sensor and microcontroller and UART communication established between microcontroller and computer through port 3.

Features Table:

Assigned Clock Speed	24 MHz
Cost	Approximately $250
TI MSP432E401Y Microcontroller	Arm Cortex-M4F Processor Core SRAM: 256KB, EPROM: 6KB Flash Memory: 1024KB Push Button Operating Voltage: 3.3 V
28BYJ-48 Stepper Motor + ULN2003 Driver Board	Operating Voltage: 5 V Full rotation: 512 steps
VL53LIX ToF sensor	Operating Voltage: 2.8V - 5 V Distance Measurable: 4 cm - 4 m
Software	C language in Keil, Python 3.8.10
Serial Communication	I2C and UART Baud Rate: 115200 bits per second Terminator: Stop Bit

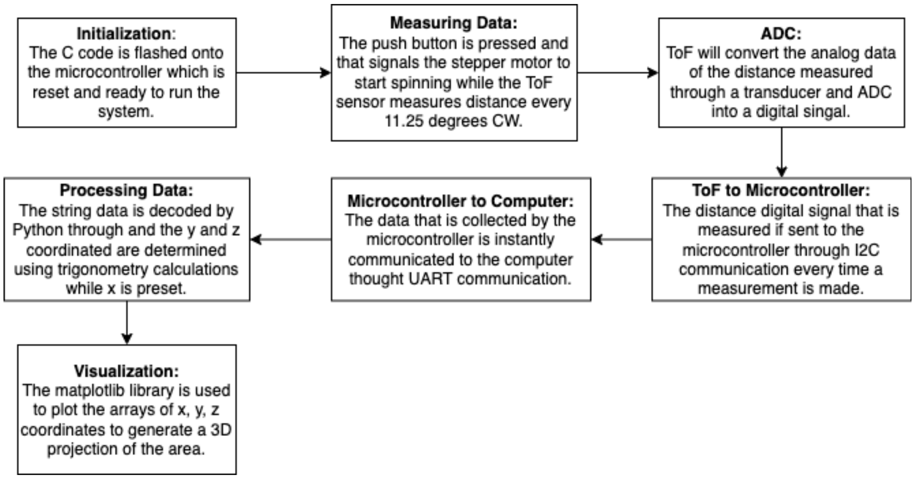
General Description:

This embedded system includes 3 main hardware components and 2 software components. The heart of the system is the TI MSP432E401Y Microcontroller that connects the stepper motor and the ToF sensor with the C and Python software in the computer. The stepper motor allows precise rotation 360 degrees with it stopping every 11.25 degrees to allow the ToF sensor to take distance measurements in a single geometric plane. Once 32 measurements are taken or the stepper motor has spun 360 degrees and reset, the system is moved forward by 15 centimeters and the process of taking the distance measurements is repeated.

The ToF sensor takes distance measurements through the use of a laser that is shot and reflected by hitting an object which is then returned to the sensor. The sensor will take the time that it takes for the light to return back multiplied by the speed of light all divided by 2. The distance is measured in millimeters.

The distance is communicated from the ToF sensor to the microcontroller through the synchronous I2C communication protocol. The microcontroller will then communicate this data to the computer through Port 3 using the asynchronous UART serial communication. This data is then processed in Python using math and pyserial library and the 3D projection of the area is graphed using matplotlib. The x coordinate is preset while the y and z coordinates are determined using trigonometry.

Block Diagram:



Initialization: The C code is flashed onto the microcontroller which is reset and ready to run the system.

Measuring Data: The push button is pressed and that signals the stepper motor to start spinning while the ToF sensor measures distance every 11.25 degrees CW.

ADC: ToF will convert the analog data of the distance measured through a transducer and ADC into a digital singal.

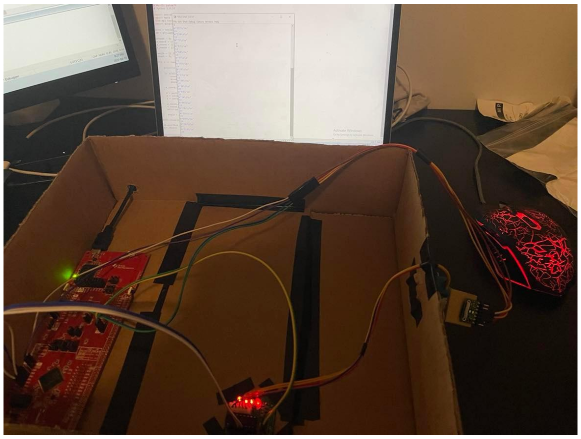
ToF to Microcontroller: The distance digital signal that is measured if sent to the microcontroller through I2C communication every time a measurement is made.

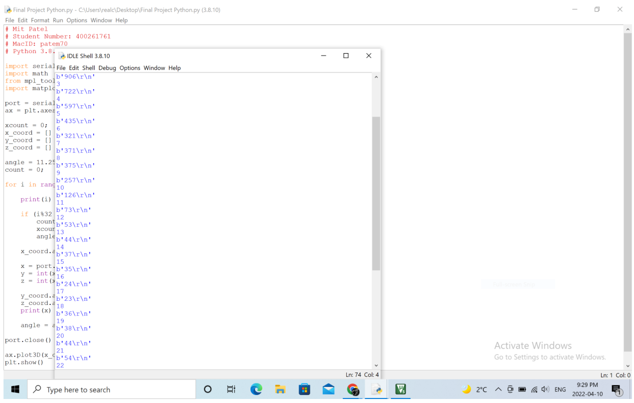
Microcontroller to Computer: The data that is collected by the microcontroller is instantly communicated to the computer thought UART communication.

Processing Data: The string data is decoded by Python through and the y and z coordinated are determined using trigonometry calculations while x is preset.

Visualization: The matplotlib library is used to plot the arrays of x, y, z coordinates to generate a 3D projection of the area.

Picture of System and PC Display:




Final Project Python.py - C:\Users\realc\Desktop\Final Project Python.py (3.8.10)
File Edit Format Run Options Window Help
IDLE Shell 3.8.10
File Edit Shell Debug Options Window Help
Ln: 74 Col: 4
Ln: 1 Col: 0
Type here to search
Activate Windows
Go to Settings to activate Windows.
2°C
ENG
9:29 PM
2022-04-10

```
# Mit Patel
# Student Number: 400261761
# MacID: patem70
# Python 3.8.

import serial
import math
from mpl_tool
import matplo

port = serial
ax = plt.axes

xcount = 0;
x_coord = []
y_coord = []
z_coord = []

angle = 11.25
count = 0;

for i in rang

    print(i)

    if (i%32
        count
        xcour
        angle

    x_coord.a

    x = port.
    y = int(x
    z = int(x

    y_coord.a
    z_coord.a
    print(x)

    angle = a

port.close()

ax.plot3D(x_c
plt.show()
```

```
b'906\r\n'
3
b'722\r\n'
4
b'597\r\n'
5
b'435\r\n'
6
b'321\r\n'
7
b'371\r\n'
8
b'375\r\n'
9
b'257\r\n'
10
b'126\r\n'
11
b'73\r\n'
12
b'53\r\n'
13
b'44\r\n'
14
b'37\r\n'
15
b'35\r\n'
16
b'24\r\n'
17
b'23\r\n'
18
b'36\r\n'
19
b'38\r\n'
20
b'44\r\n'
21
b'54\r\n'
22
```

Device Characteristics Table:

Device and Pin Connection Table:

Device:	Pin on Microcontroller:
Onboard Button	PJ1
Stepper Motor	PK0, PK1, PK2, PK3, 3.3V, Ground
ToF Sensor	PB2, PB3, 3.3V, Ground
Onboard LED	LED D3 PF4

Serial Communication And Special Functions Table:

Type	I2C, UART
Bus Speed	24 MHz
Communication Speed	115200 bps
Serial Port	COM3
Python Libraries	math, serial, matplotlib, mpl_toolkits
Keil Special Functions	PLL.c, SysTick.c, onboardLEDs.c, VL53L1X_api.c, vl53l1_platform_2dx4.c, uart.c

Detailed Description:

Distance Measurements:

In order to determine the distances and map out the area around the device, the more integral part of the system was the Time of Flight sensor. This VL53L1X sensor is a lidar sensor that uses a 940nm Class 1 laser which cannot be seen by the naked eye and is safe. Unlike other lidar sensors which use the intensity of the reflected light to measure distance, the VL53L1X ToF sensor uses the time it takes for the light to be reflected back to the sensor and calculate the distance from that time. The benefits of using time is that it is independent of ambient lighting and other environments of testing conditions such as color, reflectiveness, or texture of surface that may alter the originality of the light that is to be reflected back. The sensor is able to make accurate measurements up to 4 m with a 1 mm resolution and the maximum sampling rate in short distance mode is 50 Hz and in long distance, it is 30 Hz.

In order to make the ToF sensor work, there are API calls that can be used. These API calls for this sensor are located in the VL53L1X_api.c file. The main function that were used in my system included the VL53L1X_StartRanging(dev), VL53L1X_CheckForDataReady(dev, &dataReady), VL53L1X_GetRangeStatus(dev, &RangeStatus), VL53L1X_GetDistance(dev,

&Distance), and finally VL53L1X_ClearInterrupt(dev). While others were to ensure that the sensor was ready to start the measurement of data, the main function that actually got the distance was the VL53L1X_GetDistance(dev, &Distance). As stated above, the distance was dependent on the time it took for the light to be reflected back. The formula that allowed the sensor to calculate this was d = (c*t)/2 where c is the speed of light and t is the time and d is in mm. It was divided by 2 as you only want the distance that the laser traveled from the sensor to the object and not back.

While that is just how one measurement was made, the system was able to make 32 distance measurements around 360 degrees in 1 x plane. So to make these measurements in a circle is where the stepper motor came into use. 28BYJ-48 stepper motor is the special motor that is concerned more about the precision of the rotations it makes. It rotates in discrete full steps, and it takes 512 steps to complete a rotation. This sort of precision is achieved due to the gear ratio of 64:1. My system was making distance measurements from the ToF sensor at every 11.25 degrees, which was every 16 steps. So every 16 steps, onboard LED 3 would flash and the ToF sensor repeated the cycle of shooting a laser and calculating the distance, giving 32 distance measurements around the device in the plane.

The motion of starting and stopping the stepper motor was done with a GPIO interrupt button. One of the requirements was to ensure that the user can stop the device, move it forward and repeat measurements for a different plane or halt the process in the middle of measuring in case something needed to be changed. This was done with an onboard button using interrupts. Interrupts are used to stop the system from performing a current task to run the interrupt service routine which includes a task that is of higher priority and requires immediate attention. So, in the case that the stepper motor was spinning and for the ToF to stop measuring, when the button was pressed, it raised an interrupt that halted that process and waited for the button to be pressed again. This allows the user to change the position of the device or fix something in the middle of running the system without the need to terminate the process.

Now that the distances were being measured, there needed to be a way to communicate this data from the sensor to the microcontroller and from there to the computer. To get the data from the sensor to the microcontroller, the I2C communication protocol was used. This is a synchronous communication protocol that has a 2 wire design, one being the data line and the other is the clock line. That data is then communicated from the microprocessor to the computer and Python more specifically through the UART communication protocol. This is an asynchronous serial communication protocol that is used for both receiving and transmitting and it transmits 1 bit at a time. This communication for my system was done through port COM3 and the Python code was able to receive the data through the serial communication library 1 bit at a time.

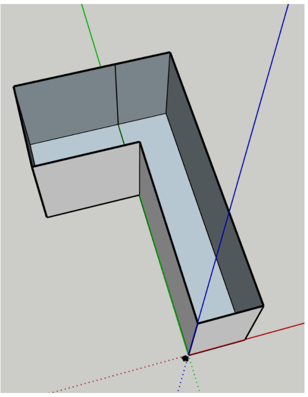
That data was raw and in string format which needed to be decoded into an int before being able to use it to determine y and z. The x coordinate was predetermined to be intervals of 100 and incremented every full rotation of the stepper motor after which the device needed to be moved up by 15 cm. The y and z coordinates were determined through trigonometric calculations. The formulas followed as such: y = d*sin(angle) and z = d*cos(angle) where angle is in radians and d is the decoded distance. The angle was the value at which the distance measurement was taken so it was 11.25 initially but incremented by 11.25 every time the for loop ran and reset to 11.25 once 32 measurements were taken and a new cycle was to start. The x, y, and z values were stored in their respective arrays.

Visualization:

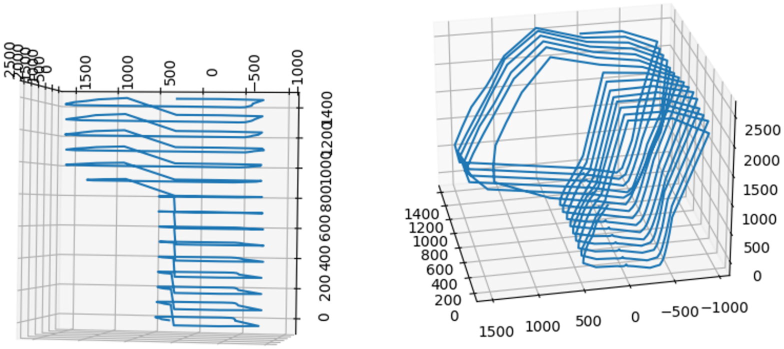
The computer system used for creating, testing and running the device was a Macbook Pro bootcamp partition of Windows 64 bits that has a 1.7 GHz Quad-Core Intel Core i7 processor with 16 GB of RAM.

As explained above in the device description, the device makes 32 readings at 11.25 degree intervals. That data is transmitted to the computer and decoded in Python where that distance measurement is used to calculate y and z while x is preset. The Python libraries that made this visualization possible were math, serial, mpl.toolkits and matplotlib.pyplot. As the coordinates were calculated, they were stored in their respective arrays: x_coord, y_coord, and z_coord. These arrays were used as parameters in the ax.plot3D() function to graph the 3D projection.

According to the microprocessor flowchart below, we can see that the sensor will take distance measurements every 11.25 degrees and send that data to Python through UART where it will continue to accept data for 480 times or 15 full rotations. It will take the raw data, decode it every time, and use it to calculate y and z through trigonometry as explained above and append those x, y, and z values to their respective arrays. After every 32 interactions of the for-loop, angle is reset to 11.25 and the x coordinate is moved up by 100 units. After 480 readings have been made, the python code will close the port and use the arrays with the coordinates to plot the 3D projection. Each of these coordinates is plotted and a line is drawn between the consecutive points that allows us to visualize this 3D model as if the lines created act like a surface. We can see this model in different angles by moving the model around in the pop up window.

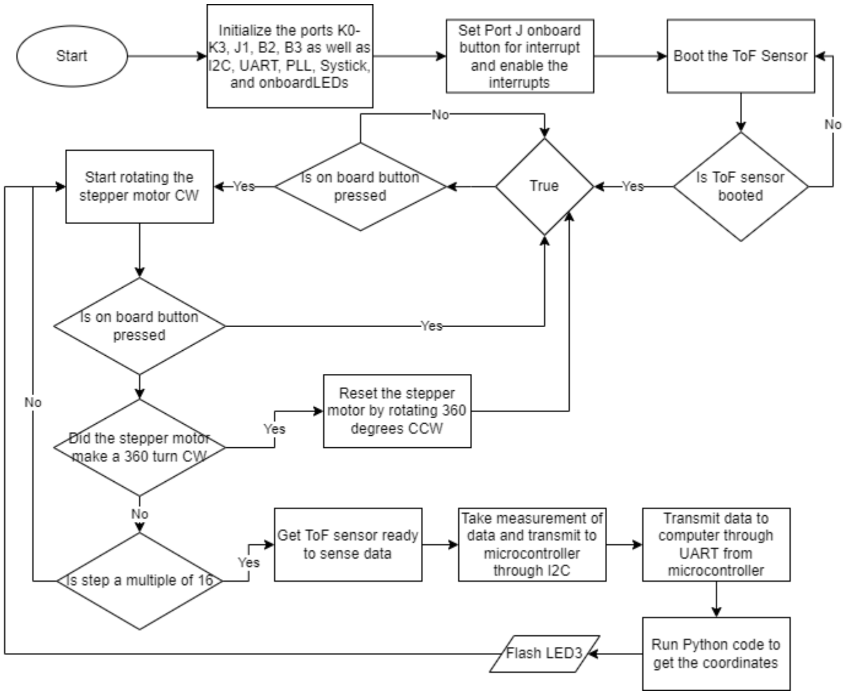


Isometric View of Area

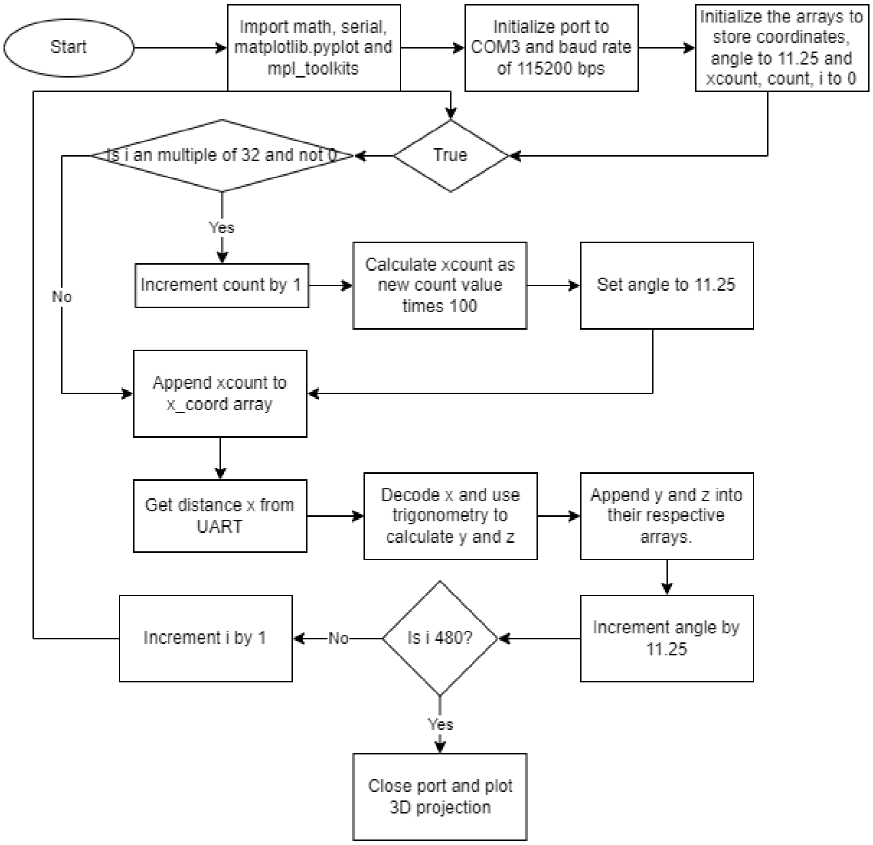


Actual 3D Models

Microcontroller Flowchart:


Start
Initialize the ports K0-K3, J1, B2, B3 as well as I2C, UART, PLL, Systick, and onboardLEDs
Set Port J onboard button for interrupt and enable the interrupts
Boot the ToF Sensor
No
Is ToF sensor booted
Yes
True
No
Is on board button pressed
Yes
Start rotating the stepper motor CW
Is on board button pressed
Yes
No
Did the stepper motor make a 360 turn CW
Yes
Reset the stepper motor by rotating 360 degrees CCW
No
Is step a multiple of 16
Yes
Get ToF sensor ready to sense data
Take measurement of data and transmit to microcontroller through I2C
Transmit data to computer through UART from microcontroller
Run Python code to get the coordinates
Flash LED3

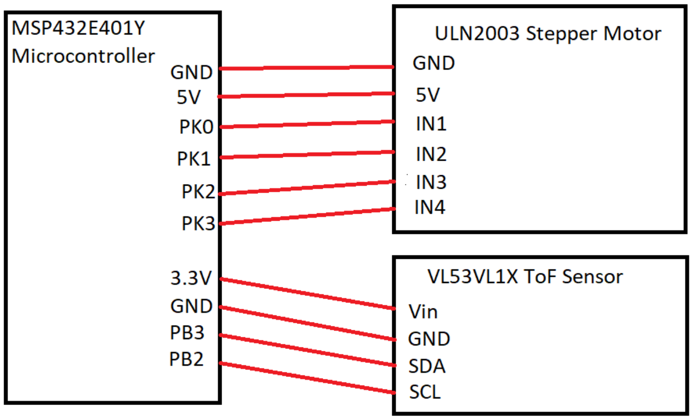
Python Flowchart:


Start
Import math, serial, matplotlib.pyplot and mpl_toolkits
Initialize port to COM3 and baud rate of 115200 bps
Initialize the arrays to store coordinates, angle to 11.25 and xcount, count, i to 0
True
Is i an multiple of 32 and not 0
Yes
No
Increment count by 1
Calculate xcount as new count value times 100
Set angle to 11.25
Append xcount to x_coord array
Get distance x from UART
Decode x and use trigonometry to calculate y and z
Append y and z into their respective arrays.
Increment angle by 11.25
Is i 480?
No
Increment i by 1
Yes
Close port and plot 3D projection

Application Example and User Guide:

1. Download Keil and set it up for the MSP432E401Y microcontroller and install its packs.
2. Download Python 3.8.10 as we will be using the IDLE to run the python component of the software.
3. Install PySerial, matplotlib, and m libraries through the Command prompt using *pip install pyserial* and *pip install matplotlib*
4. Ensure that you have the hardware components connected and set up in order to test the system.
5. Mount the ToF sensor on the stepper motor so that it can spin with the rotation of the motor.
6. Connect the ToF sensor and the stepper motor to the microcontroller using the circuit schematics layed out in the figure below.

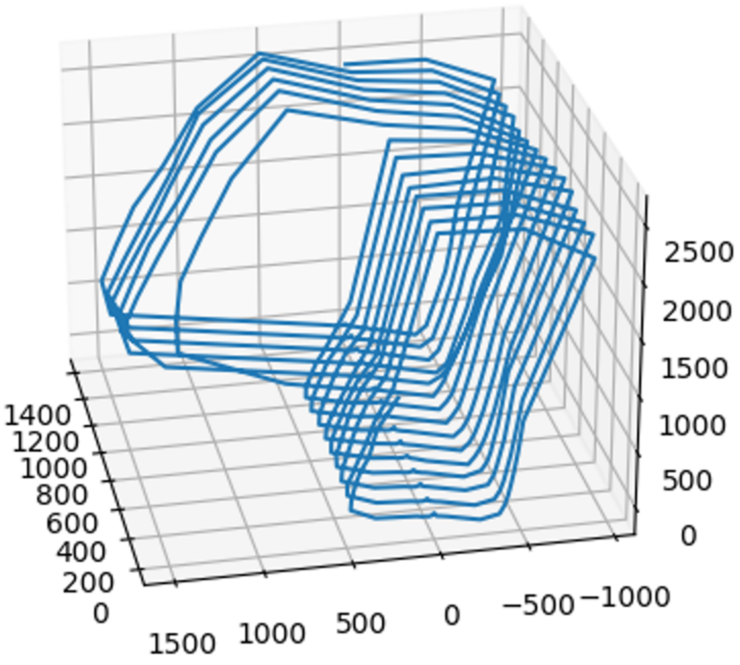
Circuit Schematic of System



7. Connect the microcontroller to your computer device with the USB cable.
8. Determine the port on your computer device for which the microcontroller is connected to through Device Manager. Open the Python file with "Edit with Idle" and change the value of your port in the Python file, line 13 and the baud rate to the right of the port to 115200.
9. Download the Keil project. In Keil, open up the magic wand tab, ensure that you have set the debug to CMSIS-DAP Debugger and the appropriate clock speed for your port (1 MHz works) and Adapter to XD110 with CMSIS-DAP in the settings beside where you selected CMSIS-DAP Debugger.
10. Compile and build the program into your microprocessor.
11. Press the reset button on the microcontroller and wait for onboard LED3 to turn off indicating that the program is ready to be run.
12. Ensure that the ToF sensor mounted on the Stepper Motor is facing up initially. This step is not necessary but recommended.

13. Run module for the Python code first. Press the onboard button to start the process for every cycle of measurement for 1 plane. Press again to stop in between processes or once the stepper motor resets, press the button to start measuring again.
14. If you want to run a certain number of cycles of measurements (2 cycles is 1 rotation and reset of the stepper motor) change the range(number) in the python file for loop on line 26, to the number of cycles you want to run multiplied by 32.

Expected Output:



* The output shown above is representative of the area that was mapped and yours could be different due to the number of cycles you ran and the area you choose to map.

Limitations:

1. There can be limitations caused by the microcontroller floating point capability and use of trigonometric functions. As we know, the MSP432E401Y microcontroller will automatically convert integers with 7 digits or less which could potentially reduce the accuracy and precision of the values that are measured by the sensor. Also, the floating point unit on the microcontroller only supports 32 bits wide and the 64 bit operations needed to be performed might have to split the upcoming data into more than 1 word, which can again reduce the accuracy of the data. Due to this limitation, the coordinates that are measured through trigonometry functions in python could be slightly off from the actual value.
2. Calculating maximum quantization error for each ToF module:

   V_{FS} = 3.3 V (input voltage from the microcontroller)
   m = 8 as the ToF sensor uses a 8-bit ADC
   Formula to calculate Max Quantization Error (MQE) is $MQE = V_{FS} / 2^m$

   Therefore:
   $MQE = V_{FS} / 2^m$
   $MQE = 3.3 / 2^8$
   $MQE = 3.3 / 256$
   $MQE = 0.0129$

   The maximum quantization error for the ToF module is 0.0129.
3. The maximum standard serial communication that I can implement for my computer is 128000 bps which was verified by checking the port setting for COM3 in device manager on my macbook.
4. The communication methods that could be used are the half duplex for two way communication and the simplex for one way communication. Since the data was only being transmitted one way, the simplex method was used with a speed of 115200 bps. The I2C protocol was used for communication between the ToF and the microcontroller.
5. The primary limitation on speed of the entire system was caused by the stepper motor. Using the full step method, there is a set Systick delay placed so to move 1 of the 512 steps, there is at least a 40 ms delay. When adding this up all together for all the 512 steps, that is a total of 20480 ms for a full rotation. I realized and tested this by playing with the Systick time and realized that when the stepper motor was rotating faster due to the smaller delays, the time it took me to graph a rotation was slightly faster but over 15 cycles, would have been significantly faster. This is why I believe that the speed of the system was based on how fast the stepper motor took to complete a rotation.