2DX3: Microprocessor Systems

Final Project

Instructors: Drs. Boursalie, Doyle, and Haddara

Antheus Aldridge – aldria1 – 400339569 – 2DX3 – Wednesday Afternoon – L09

Device Overview

Features

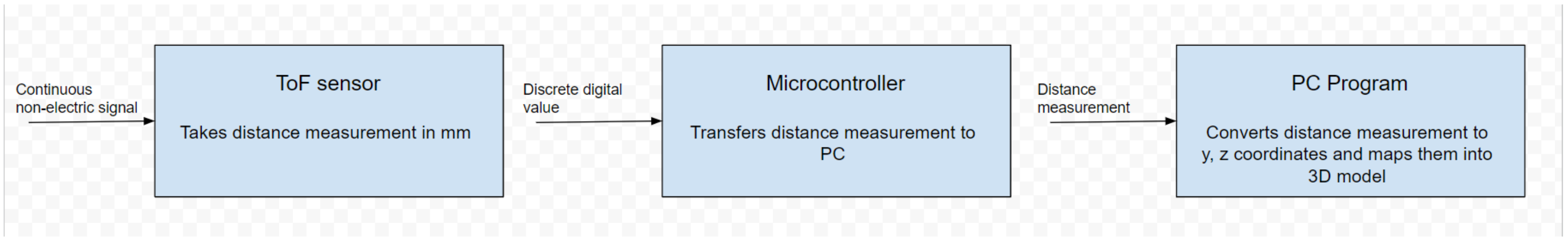
The device is able to take measurement scans of a room and recreate that room as a 3D model from the scans.

- The device takes y-z scans using a time-of-flight (ToF) sensor
- These measurements are fed into a microcontroller and then a PC program
- Multiple scans are made for different x displacements
- Scans are connected and mapped into a 3D model

The ToF sensor that we are using is the VL531X and the microcontroller is the TI MSP432E401Y. [1] The microcontroller operates at about 3.3 V at a clock speed of 120 MHz, however for my project I will lower this to 20 MHz. It has 1024 KB of flash memory, 256 KB of SRAM, and 6 KB of EEPROM, as well as internal ROM. It sports two 12-bit SAR-based ADC modules that sample at up to 2 million samples per second. To order from Texas Instruments, the microcontroller can be bought for about $50 CAD [2]. It runs on the Thumb instruction set and can be programmed in assembly language as well as C. As for serial communication, the device works on universal serial bus or USB, and can transfer data using eight universal asynchronous receivers or transmitters, also called UARTs. It can reach baud rates of up to 7.5 Mbps for regular speed and 15 Mbps for high speed [1]. The microcontroller uses Bob Smith termination [3].

General Descriptions

The parts in this system are the microcontroller, ToF sensor, stepper motor, a push button in the form of a breadboard circuit, and a PC. The microcontroller is used to interface between the PC and the various components such as the ToF sensor, push button, and stepper motor. The ToF sensor gathers distance measurements between it and the closest object. The stepper motor rotates when given an electric signal. The push button stops and starts the components in the circuit (ToF sensor and stepper motor). The PC supplies power to the whole system and also runs code which tells the microcontroller how to operate and later runs another program to convert the measurements into a 3D model.


Continuous non-electric signal
ToF sensor
Takes distance measurement in mm
Discrete digital value
Microcontroller
Transfers distance measurement to PC
Distance measurement
PC Program
Converts distance measurement to y, z coordinates and maps them into 3D model

The ToF sensor works by taking a distance measurement signal, processing it, and then turning it into a digital value. To acquire a signal, the ToF sensor will emit a photon of light, and at the same time start a timer. Once the photon bounces off an object and comes back, the timer will stop, and it can then determine how far away an object is. It will then process the signal and convert it to a digital value.

As for serial communication, the ToF sensor would send its data using I^2C to the microcontroller. Then, I used a UART protocol where the microcontroller would send data to my PC across UART. My python code would then interpret this and perform calculations with it.

Device Characteristics

This chart outlines the ports and pins that are being used on the microcontroller, and what purpose they are being used for. For instance, pins 2 and 3 of port B are being used to connect to the ToF sensor.

Port	Pins	Application
Port B	3:2	ToF Sensor Readings
Port H	3:0	Stepper Motor Control
Port M	0:0	Push Button

On top of the pins listed above, a 5 V and ground pin were used to power the stepper motor, and a 3.3 V and ground pin were used to power the ToF sensor. Finally, another 3.3 V pin and a ground pin were used to power the breadboard that the push button was on. The push button is connected to port m pin 0 and uses an active low configuration which means that when it is pressed it transmits a logic low.

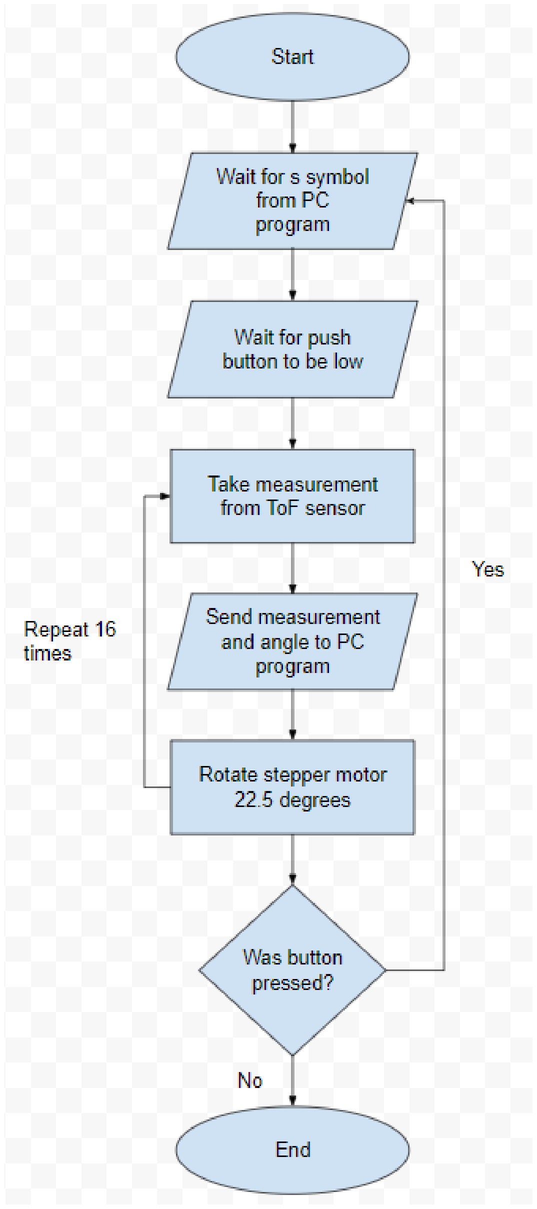
Apart from the connections, for the system to run the microcontroller needs to be connected to a PC with the appropriate Keil project and Python code being used. Inside the Keil code, the bus speed that I used in my design was 20 MHz. The serial port that was used was a UART protocol to send data from the microcontroller and ToF sensor to the PC program at a speed of 115200 baud. In the Python program the libraries I used include Open3D and NumPy to model the data.

Detailed Description

Distance Measurement

The ToF sensor works by taking an analog signal, processing it, then converting it to a digital value. The VL531X first sends out a 940 nm photon of light and starts a timer. Then, the photon bounces off a nearby object and comes back to the sensor. The VL531X then stops the timer and measures how long it took for the photon to emit and reflect. It then processes the data and converts it to a digital value which can be interpreted by the microcontroller and PC program. Once the digital value is acquired, it is sent to the microcontroller using I^2C. To do this, it uses two signals, serial data line (SDA) and serial clock line (SDL). These two pins are connected to pin 2 and 3 of port B. The microcontroller initiates data transfer, at which point a clock signal is generated. Considering the ToF uses I^2C, the data is transferred in bytes, with the first byte being the address of the follower or leader, and the second being the data to be transferred. In between these, there are ACK bits which signal that the follower or leader has acknowledged the transfer. The follower or leader address and ACK signal depends on who is receiving data, and who is sending it. In our case, the VL531X will be sending data and the microcontroller will be receiving it [4].

Once the microcontroller has received the data, it uses a UART protocol to send the data to the PC. To do this, we use the PC program to send a start signal to the microcontroller to sync the transfer. After this, the microcontroller starts sending over the distance measurements from the ToF sensor to the PC. Since we are using UART, the transfer is initiated by a start bit, after which a data frame of 7 or 8 bits is sent. A parity bit can be used to tell if the data is correct or if it has been altered. Finally, a stop bit signals the end of the data transfer. In my program, I send the distance reading from the ToF sensor, along with a manually generated angle measurement. The angle goes from 0 degrees to 337.5 degrees, incrementing by 22.5 degrees each time, for a total of 16 measurements being transferred.


Start
Wait for s symbol from PC program
Wait for push button to be low
Take measurement from ToF sensor
Send measurement and angle to PC program
Rotate stepper motor 22.5 degrees
Repeat 16 times
Was button pressed?
Yes
No
End

Once the program starts, and the PC program sends the s character, the microcontroller waits for the push button to be active low which sends a 0 to port m pin 0. At this point, the program begins, and the microcontroller starts taking input from the ToF sensor and outputs it to the PC program. After each measurement and transfer, the stepper motor is rotated 22.5 degrees and the angle value in the code is updated. After each set of 16 measurements, the stepper motor is rotated 360 degrees counter clockwise to prevent the wires from being twisted too much. The inputs for the program include the s character from the PC program, and the input from the push button. The outputs include sending data to the PC program and outputting values to the stepper motor to control its movement. After the desired number of y-z slices are taken (sets of 16 measurements), the program ends.

Visualization

On the PC side of things, I am using a laptop with an Intel i5 11th generation processor and 8 GB of RAM running on windows 10. The programming language that I am using for the PC program is Python. To achieve the desired functionality, I am using PySerial, Open3D, and NumPy. PySerial is used to transfer data from the microcontroller in real time, while Open3D and NumPy are used to convert the data to xyz coordinates and model it into a 3D visualization of the scanned object.

After the data has been acquired and transmitted, we have to process it and convert it into xyz coordinates. To do this, I used trigonometric functions to convert the reading from a polar coordinate system to a (y, z) coordinate system. To do this I first converted the measurements from a “byte” data type to integers. After this, I used the following formulas to convert them into y and z coordinates:

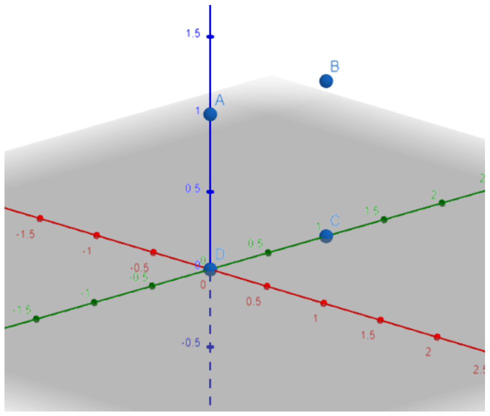
$$y = (distance) * \sin(\Theta)$$

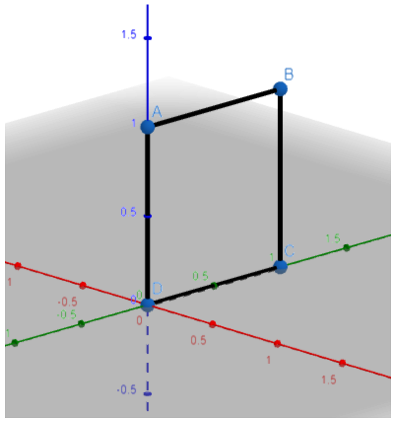
And

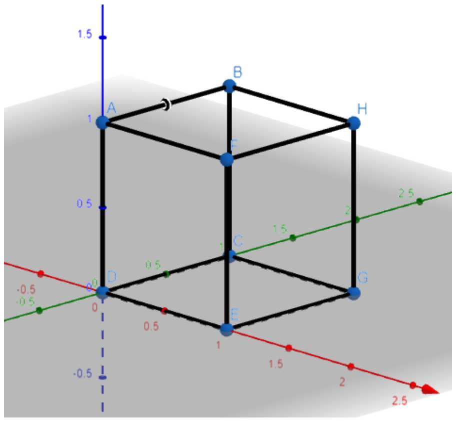
$$z = (distance) * \cos(\Theta)$$

For example, at an angle of 0 degrees, the y value would be equal to the distance, and the z value would be 0.

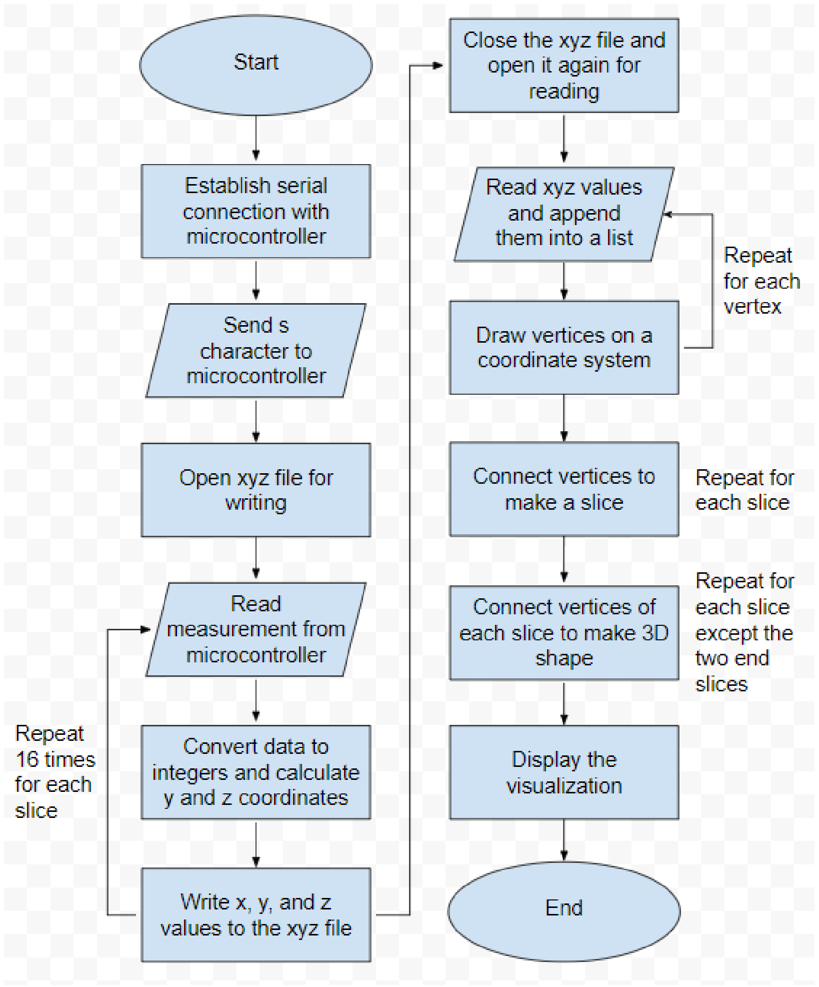
After getting the y and z values, I would write them into the xyz file along with an x measurement corresponding to the displacement along the ground of my system. I would move my system between measurements a displacement of about 30 cm and update my x value manually in the PC program. To visualize the data, Open3D will read the values from the xyz file and visualize them as points on a coordinate system. These are the vertices of the scan. After this, we can connect every vertex in a slice to the next one to create lines between the vertices. Finally, once the slices have been connected by lines, we can connect each slice to the next one by drawing lines between each vertex in the slice and the corresponding vertex in the next slice.



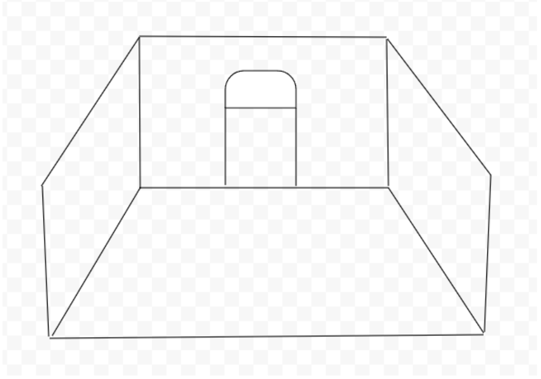




This is the process that the visualization follows. As you can see, in the first image points or vertices are drawn. In the second image, the vertices are connected to each other to form a slice. In the last image, there is another slice that has been drawn and connected, and the vertices from each slice are being connected to form a 3D shape such as a cube.


Start
Establish serial connection with microcontroller
Send s character to microcontroller
Open xyz file for writing
Read measurement from microcontroller
Convert data to integers and calculate y and z coordinates
Write x, y, and z values to the xyz file
Repeat 16 times for each slice
Close the xyz file and open it again for reading
Read xyz values and append them into a list
Draw vertices on a coordinate system
Repeat for each vertex
Connect vertices to make a slice
Repeat for each slice
Connect vertices of each slice to make 3D shape
Repeat for each slice except the two end slices
Display the visualization
End

Following the flowchart of the PC program, it starts by specifying the parameters of the serial connection and establishing said connection in PySerial. We are using an 115200 baud, reading from the microcontroller, and I have added a timeout function of 15 seconds so the program stops if it doesn't receive an input within 15 seconds. After this, we send the s character to the microcontroller to tell it to start. This also synchronizes the data transfer. Next, we open the xyz file for writing, so we can add in the xyz coordinates of the vertices later. Then, the microcontroller will start sending over measurements and the program will read each line of input in the byte format. Each line has the distance and angle measurement. After this, we have to convert the data from bytes to integers, and also separate the distance and angle measurement into two elements of an array. Then, I perform trigonometric calculations of the data mentioned above to convert the data from polar coordinates to (y, z) coordinates. Finally, we can write the (x, y, z) coordinate to the xyz file. The x value is hard coded to increase by a set amount for each slice, to represent me moving the system physically in the x-direction. After this we close the xyz file since we are done writing to it, then we open it again in read mode so we can use it to visualize the scans. First, all the xyz coordinates are appended into a list, where they can be accessed via indexing. Then, we go through a loop for each vertex, and connect them to the next vertex with a line. After one iteration of this loop, we will have drawn a y-z slice. We continue this process for every slice taken. After that, we will connect the vertices of each slice to the corresponding vertices of the next slice. For example, vertex one will have a line drawn between it and vertex one of the next slice. We continue this for every slice, except the last slice. Finally, we can display the visualization. The space that I recreated is my living room. Here is an isometric sketch of it:



Application Example

As an example of the system working as a whole, I will walk through the steps that it goes through, from acquiring a signal to displaying the visualization. In my case, I used the system to model a scan of my living room.

To start, the microcontroller should be flashed with the Keil code, the appropriate connections should be made between pins and the other devices, and the system must be physically set up in the correct orientation. To wire everything correctly, see the circuit diagram (figure 1). the steps are as follows:

1. Python PC program is run and reset button on microcontroller is pressed.

2. After about 10 seconds, the Python code will output 1, 2, 3 – at this point it is ready to receive data and the push button connected to the microcontroller should be pressed.

3. ToF sensor takes measurement and transmits it to the program.

4. Program reads the input and converts it to two integer values, one for distance and one for angle.

5. Program writes x, y, and z coordinate into the xyz file.

6. Stepper motor rotates 22.5 degrees.

7. Steps 3 to 6 repeat until the stepper motor has made a full 360 degree rotation and the ToF sensor has taken 16 measurements.

8. After the full clockwise rotation, the stepper motor rotates 360 degrees counter clockwise to prevent the wires from being twisted.

9. Move the entire system forward to meet the desired x displacement; I moved my system about 30 cm each time.

10. Steps 3 to 8 repeat until the number of slices specified in the program and Keil code has been satisfied.

11. After all the measurements are taken, the program takes all x, y, z measurements from the xyz file and appends them to a list.

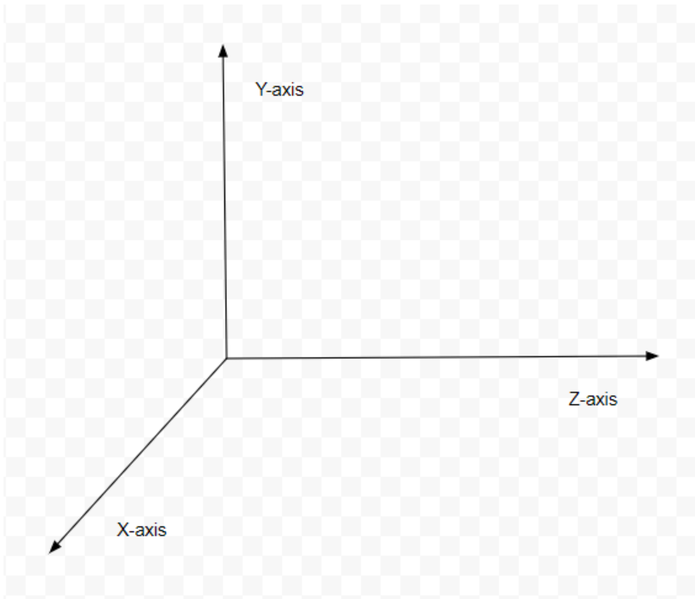
12. The program iterates through each x, y, z vertex in the list and connects the neighbouring vertices by appending the two vertex values to be connected into a list of line values.

13. The program iterates through every vertex except the ones in the last slice and connects them to the corresponding vertex in the next slice using the list of line values again.

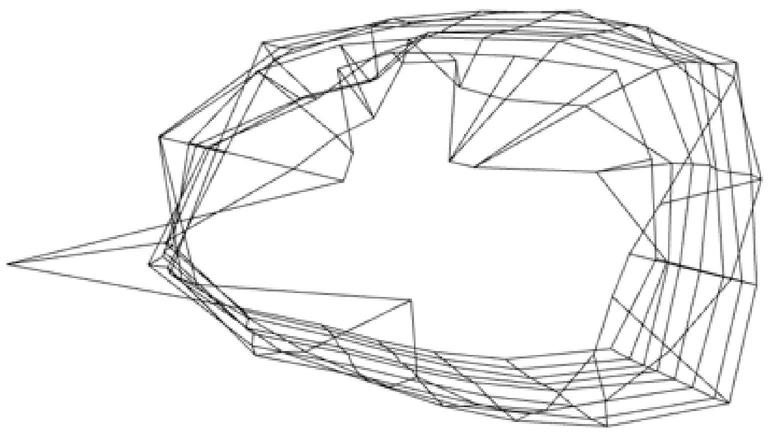
14. Next, the program creates a line set using the line values appended to the lines list. The line values are two vertex values together in a set.

15. Finally, the program uses Open3D to display the lines which have been drawn. This is the final visualization of the scans.

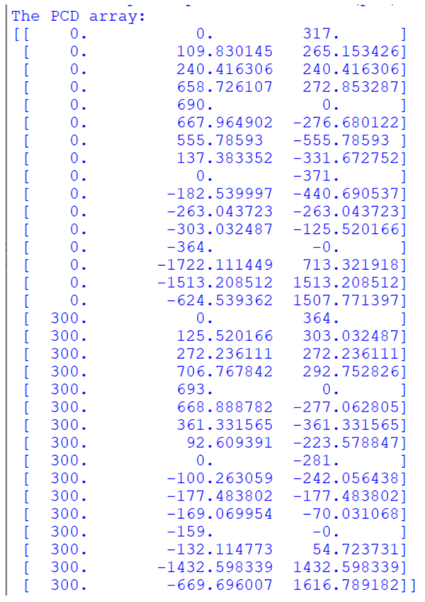
The application of this process is to create a scan of a room or surface that you desire. This could be used as a computer vision module or LIDAR equivalent. For example, a robot could take advantage of this by sensing nearby objects and acting accordingly. To interpret the data, the x measurement describes the forward or backward displacement along the ground. The y and z values represent the vertical and horizontal distances between the ToF sensor and the nearest objects. Here is the coordinate system that I followed for my visualization:



The expected output for the system should be distance readings that are proportional to the actual distance between the system and objects in the room or hallway that is being scanned. The angle value should increment by 22.5 each time from 0 to 337.5. In terms of the final output, the visualization should be similar to the area being scanned. For an example, my visualization for a room that I scanned looked like this:



While the code output looked like this:

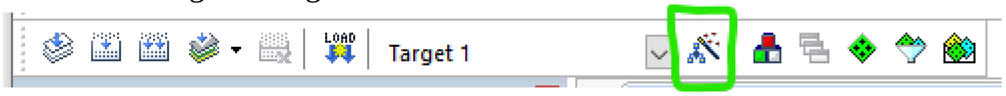


```
The PCD array:
[[    0.           0.         317.      ]
 [    0.         109.830145   265.153426]
 [    0.         240.416306   240.416306]
 [    0.         658.726107   272.853287]
 [    0.         690.           0.      ]
 [    0.         667.964902  -276.680122]
 [    0.         555.78593   -555.78593 ]
 [    0.         137.383352  -331.672752]
 [    0.           0.        -371.      ]
 [    0.        -182.539997  -440.690537]
 [    0.        -263.043723  -263.043723]
 [    0.        -303.032487  -125.520166]
 [    0.        -364.          -0.      ]
 [    0.       -1722.111449   713.321918]
 [    0.       -1513.208512  1513.208512]
 [    0.        -624.539362  1507.771397]
 [  300.           0.         364.      ]
 [  300.         125.520166   303.032487]
 [  300.         272.236111   272.236111]
 [  300.         706.767842   292.752826]
 [  300.         693.           0.      ]
 [  300.         668.888782  -277.062805]
 [  300.         361.331565  -361.331565]
 [  300.          92.609391  -223.578847]
 [  300.           0.        -281.      ]
 [  300.        -100.263059  -242.056438]
 [  300.        -177.483802  -177.483802]
 [  300.        -169.069954   -70.031068]
 [  300.        -159.          -0.      ]
 [  300.        -132.114773    54.723731]
 [  300.       -1432.598339  1432.598339]
 [  300.        -669.696007  1616.789182]]
```

User’s Guide

I used Open3D and PySerial for this project, so those must be installed first. Assuming you already have Python and Keil downloaded, to get Open3D, you have to open command prompt in windows and type: “pip install open3d”. Similarly, to get PySerial you have to open command prompt and type “pip install pyserial”.

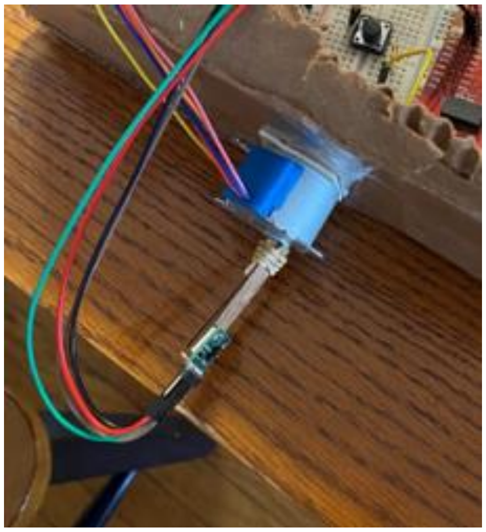
Another thing to do before setting up is the ensure that the Keil code file has the correct target. Go into the target settings here:



Then, go to debug and ensure that CMSIS-DAP Debugger is selected next to the use drop down. Next, click settings next to that and select XDS-110 with CMSIS-DAP.

To set up the device, the following steps must be taken:

- Connect IN1:IN4 on the stepper motor to PH0:PH3 on the microcontroller
- Connect the negative and positive terminals of the stepper motor to ground and 5 V on the microcontroller, respectively
- On the ToF sensor, connect the SDA pin to PB2 and the SCL pin to PB3
- Connect the ground pin to ground, and the Vinn pin to 3.3 V.
- To set up the push button, wire the button in an active low configuration with the output going to PM0
- Connect ToF sensor to the stepper motor such that the sensor is facing in the vertical plane, so that it can rotate from ground to ceiling



The ToF sensor should be orientated in the way shown in the image to the left

- Put all the components in a box or other object for easy movement
- Connect the system to a PC, preferably a laptop so you can move it with the system
- Download the zip file and unzip it to a folder
- Run the Keil project and flash the code to the microcontroller
- Open the Python file
- Ensure that the serial port is being opened with a baud of 115200, and the COM port is the same as the one that is specified in your device manager under Ports called UART communication port
- Run the Python file
- Once it outputs 1, 2, 3 to the shell, press the push button and the data measurements should begin to transfer
- Allow the stepper motor to complete one full rotation, once it starts rotating in the opposite direction move the entire system 30 cm forward in the x direction
- Continue this until the system is done scanning and the Python file outputs the visualization

Limitations

1) When the microcontroller is processing floating point numbers, there is an inherent error since it has to convert decimal point values to binary values, and binary digits cannot express decimal numbers exactly. For example, 0.1_2 is equal to 0.5_{10} and 0.01_2 is equal to 0.25_{10}. As you can see, each binary number beyond the decimal point is half of the value of the digit to the left of it. Therefore, when given a fixed number of bits to use, such as in a microcontroller's memory, it can be hard to represent decimal values exactly using binary digits. Our microcontroller has a 32-bit floating point unit or FPU so it may be hard to represent large floating point decimal values. Furthermore, when doing calculations in the Python program such as trigonometric functions, there is another inaccuracy since the results of these calculations are floating point numbers. Once again, the program cannot represent floating point numbers entirely accurately so there is an inherent error involved.

2) The max quantization error for the ToF is inversely proportional to the number of bits of its output ADC. In this case, the ToF outputs values in a 16-bit format. Also, using my distance mode the max distance is 3m or 3000 mm. Therefore, the max quantization error is:

$$Max\ Quantization\ Error = \frac{3000\ mm}{2^{16}} = 0.04577\ mm$$

3) The maximum standard serial communication rate that works with the PC is a baud of 115200. I verified this by checking the port settings of COM3 which is my port for UART data transfer. I also tried various values with the Python program, and 115200 was the highest one that worked with my system.

4) The communication method used between the microcontroller and ToF is I^2C. The speed of this transfer is 100 kbps.

5) The primary limitation on speed in the entire system would be the stepper motor since we have to wait for it to move into position before we can take a distance measurement. In terms of transfer rates, the I^2C transfer rate is 100 kbps which is less than the baud of the UART transfer which is 115200. Therefore, the I^2C would be the limiting factor in data transfer.

Circuit Schematic

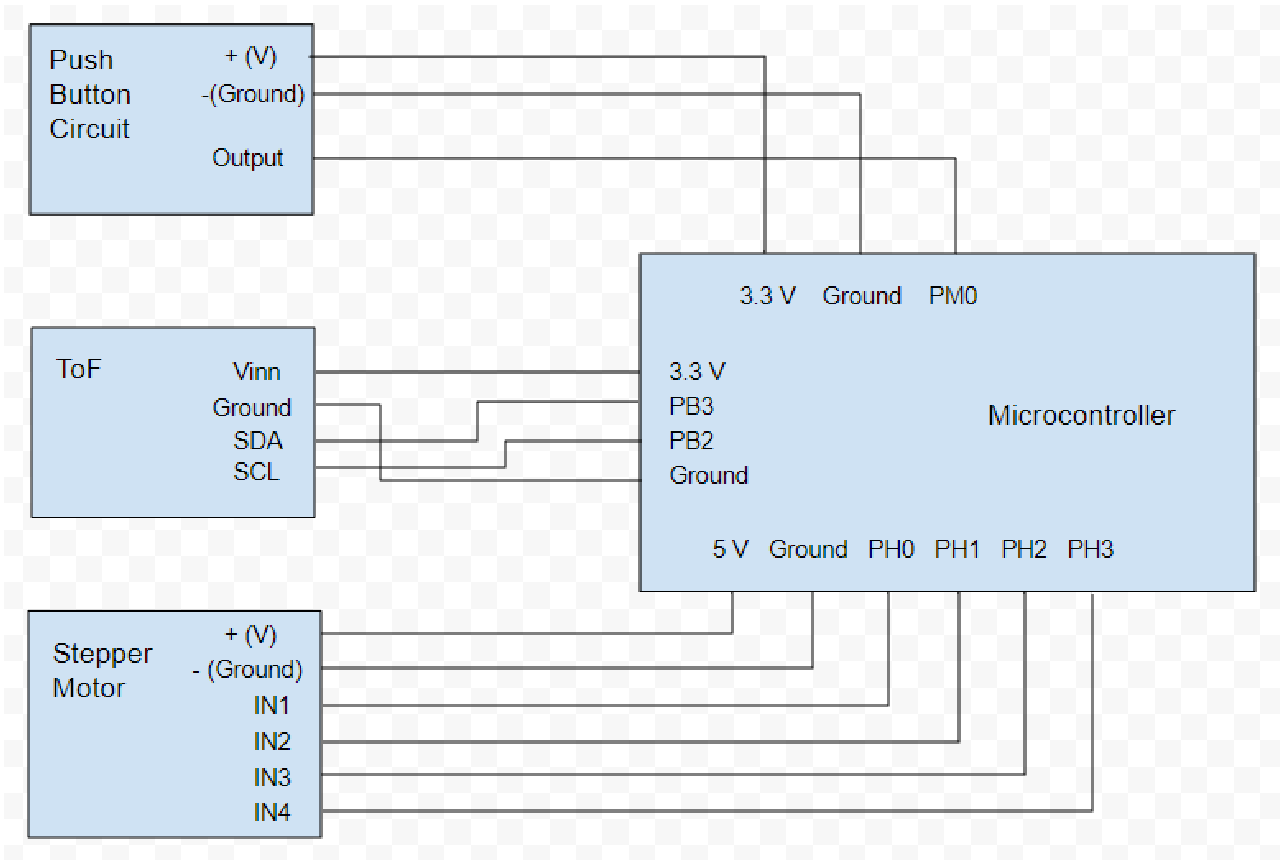


Figure 1. Circuit Schematic

Flowcharts

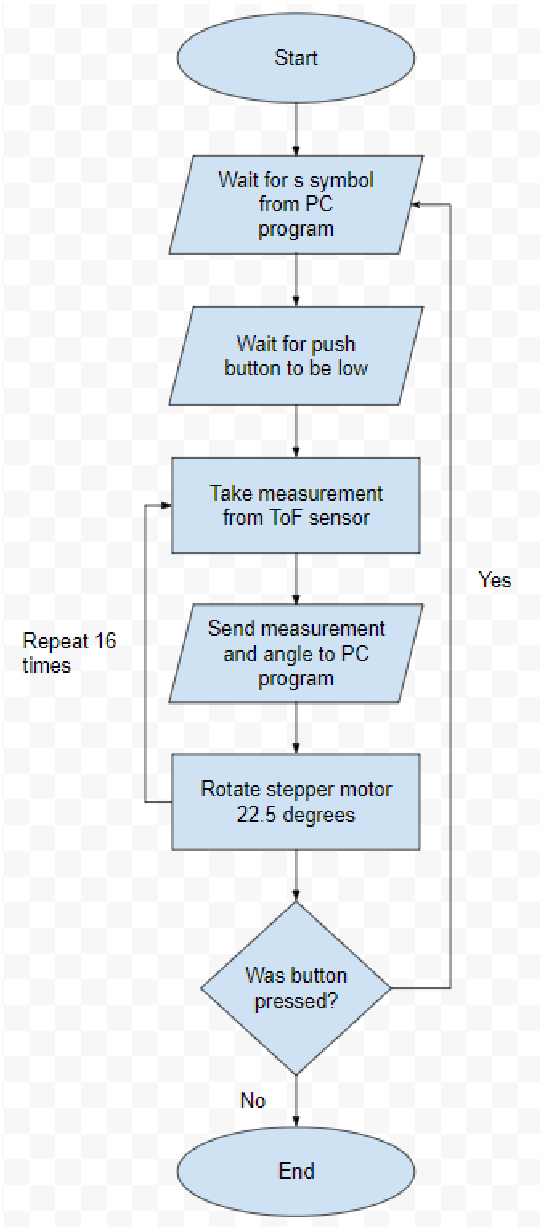

Start
Wait for s symbol from PC program
Wait for push button to be low
Take measurement from ToF sensor
Send measurement and angle to PC program
Rotate stepper motor 22.5 degrees
Was button pressed?
Yes
No
Repeat 16 times
End

Figure 2. Keil Program Flowchart

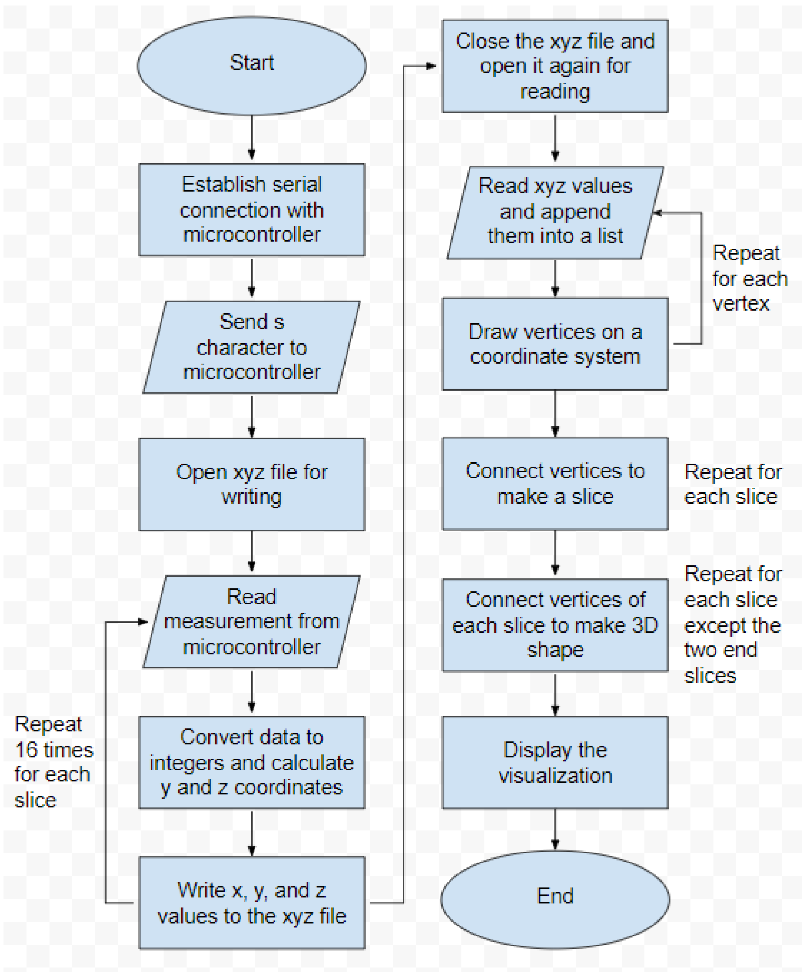

Start
Establish serial connection with microcontroller
Send s character to microcontroller
Open xyz file for writing
Read measurement from microcontroller
Convert data to integers and calculate y and z coordinates
Write x, y, and z values to the xyz file
Repeat 16 times for each slice
Close the xyz file and open it again for reading
Read xyz values and append them into a list
Repeat for each vertex
Draw vertices on a coordinate system
Connect vertices to make a slice
Repeat for each slice
Connect vertices of each slice to make 3D shape
Repeat for each slice except the two end slices
Display the visualization
End

Figure 3. Python Program Flowchart

References

[1] “MSP432E401Y datasheet,” *MSP432E401Y datasheet | TI.com*. [Online]. Available: https://www.ti.com/document-viewer/MSP432E401Y/datasheet/abstract#t121920850. [Accessed: 09-Apr-2022].

[2] “MSP-EXP432E401Y,” *MSP-EXP432E401Y Development kit | TI.com*. [Online]. Available: https://www.ti.com/tool/MSP-EXP432E401Y#order-start-development. [Accessed: 09-Apr-2022].

[3] “SimpleLink^{TM} Ethernet MSP432E401Y Microcontroller LaunchPad^{TM} Development Kit (MSP-EXP432E401Y) User’s Guide SimpleLink^{TM} Ethernet MSP432E401Y Microcontroller LaunchPad^{TM} Development Kit (MSP-EXP432E401Y),” 2017. [Online]. Available: www.ti.com

[4] “VL53L1X,” 2018. [Online]. Available: www.st.com