2DX4: Microprocessor Systems
Final Project Report

Instructor: Drs. Boursalie, Doyle, Haddara

Kumod Ranuja Pinnaduwage – pinnaduk – 400269531 – COMPENG 2DX3 – Wednesday Afternoon – L03

Device Overview

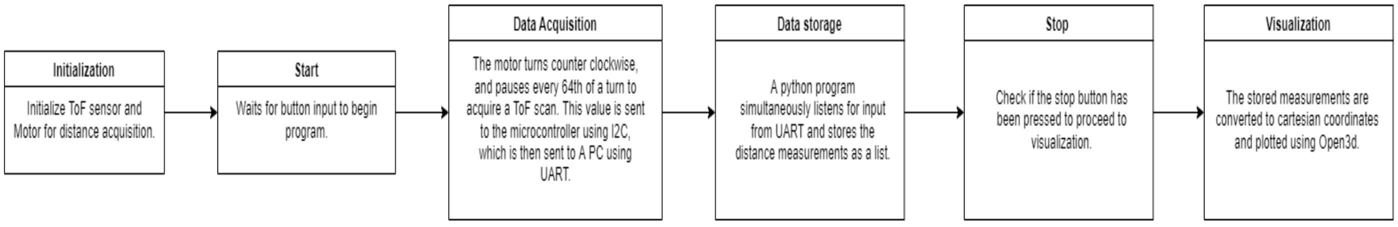
Features

- This device is made to capture a mapping of the devices surroundings.
- Utilizes the Texas Instruments MSP432E401Y Microcontroller.
- Utilizes VL53L1X Time of Flight sensor.
- The device works on a clock of 24 Mega Hertz.
- The Motor operates on 5 Volts and the Time of Flight sensor operates on 3.3 Volts.
- Memory space is used to store the acquired measurements as integers.
- The microcontroller operates on C code that is flashed onto it, and the 3D reconstruction is done through Python.
- The Time of Flight sensor sends its measurements to the microcontroller via the I2C protocol, and the microcontroller sends the computer the data through UART.
- The Baud rate of the data transfer is 115200 bits per second.

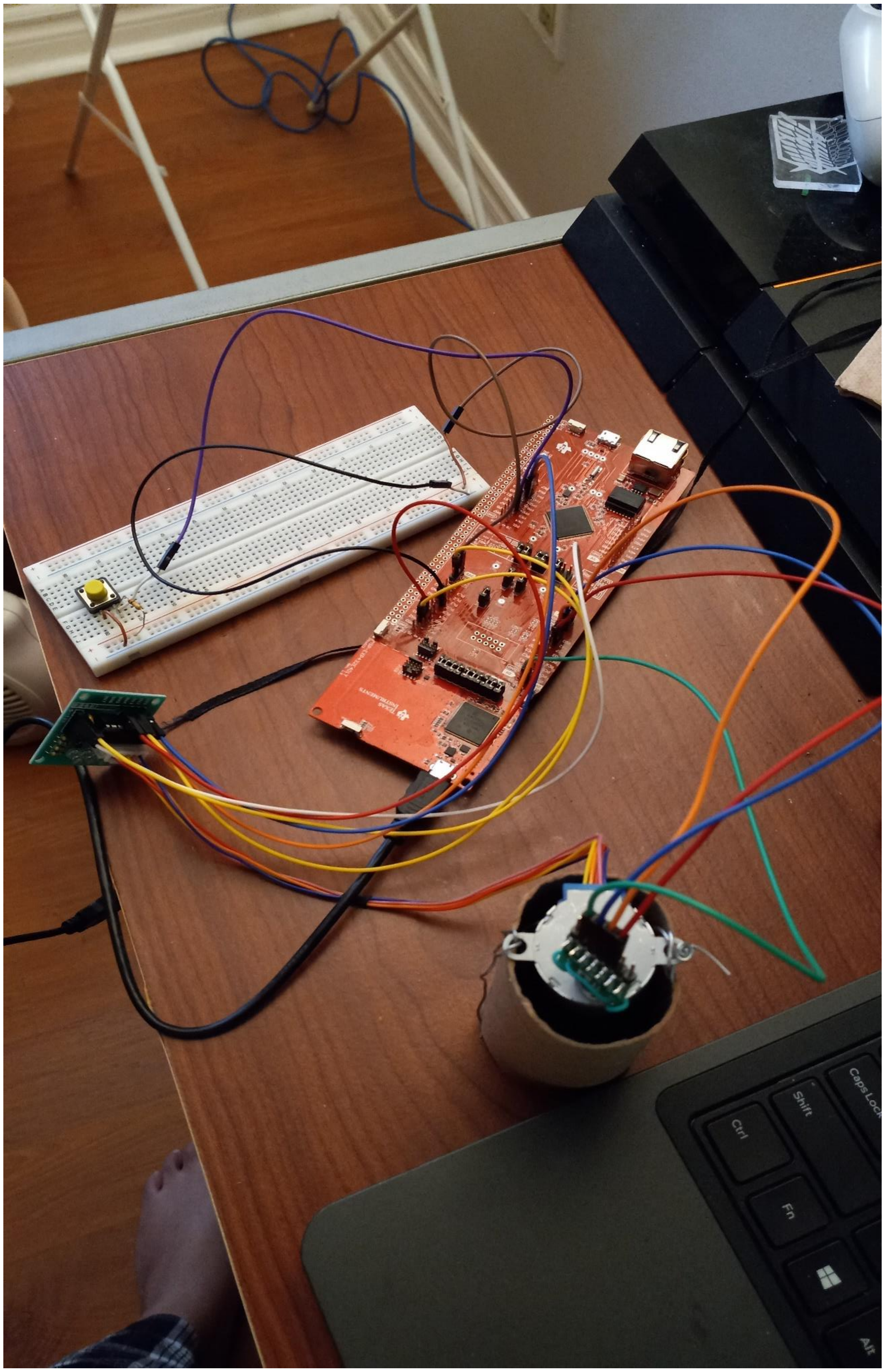
General Description

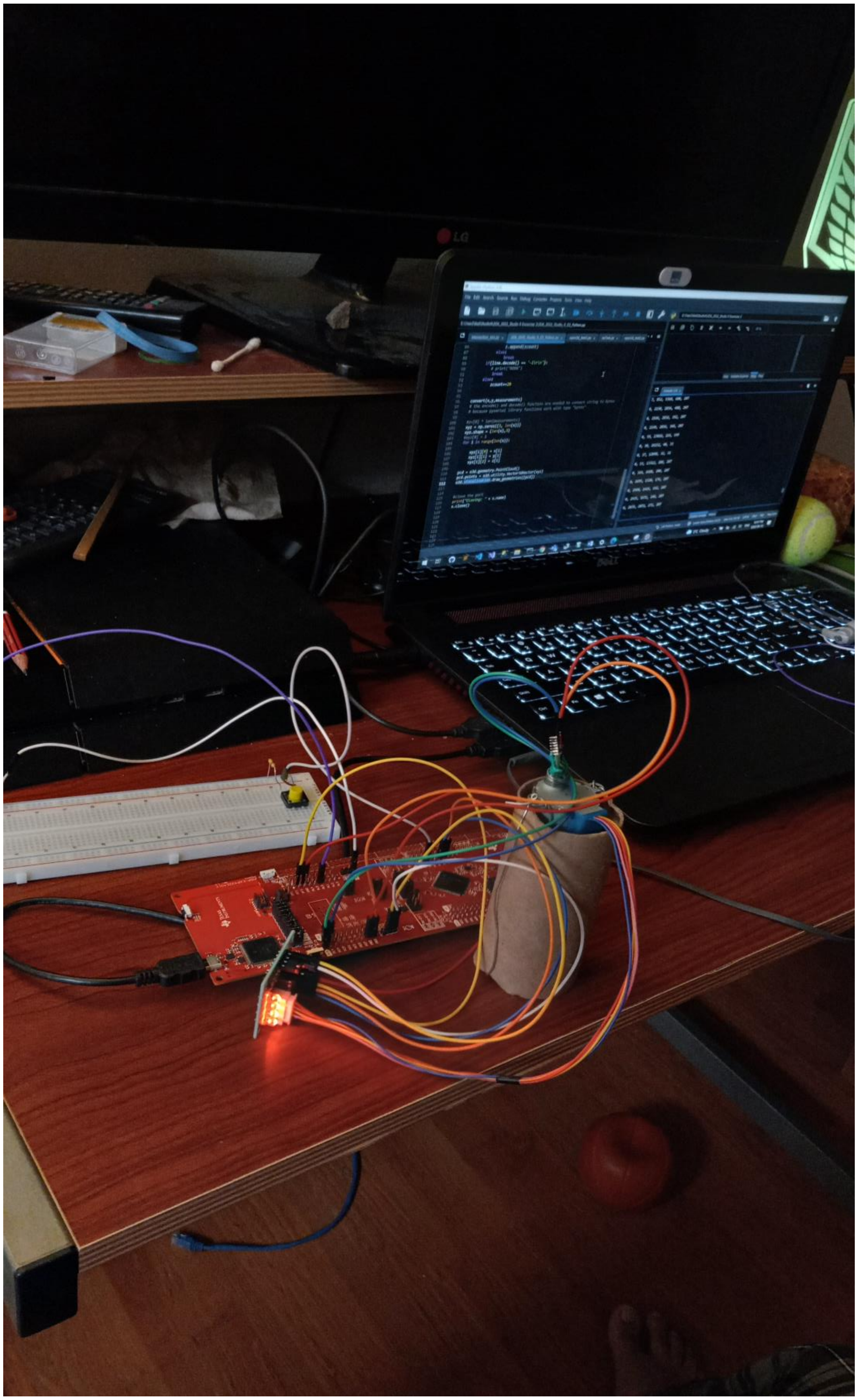
This system contains several parts that work together to create the final 3D model. There is the microcontroller which controls the operation of the stepper motor as well as that of the Time of Flight (ToF) sensor. The stepper motor is made to rotate 360 degrees whilst the time of flight sensor acquires distance measurements every 64th of a turn. The ToF sensor uses invisible light pulses and measures the time it takes for their return to determine the distance the light travelled to hit it. This way, the distance measurements that are sent to the microcontroller using the I2C protocol, where it is then sent to a PC using the UART protocol. The Python program waits for data being sent via UART and appends it to a list, which it can then use to acquire Cartesian coordinates. Once this is done, the Open3D library is implemented to create a 3D model of the measurements. The entire system begins and ends through the usage of a start/stop button. A polling method is utilized to check for the button push to indicate that the motor should start turning. Once the system has begun, the button status is polled to decide when to stop the program.

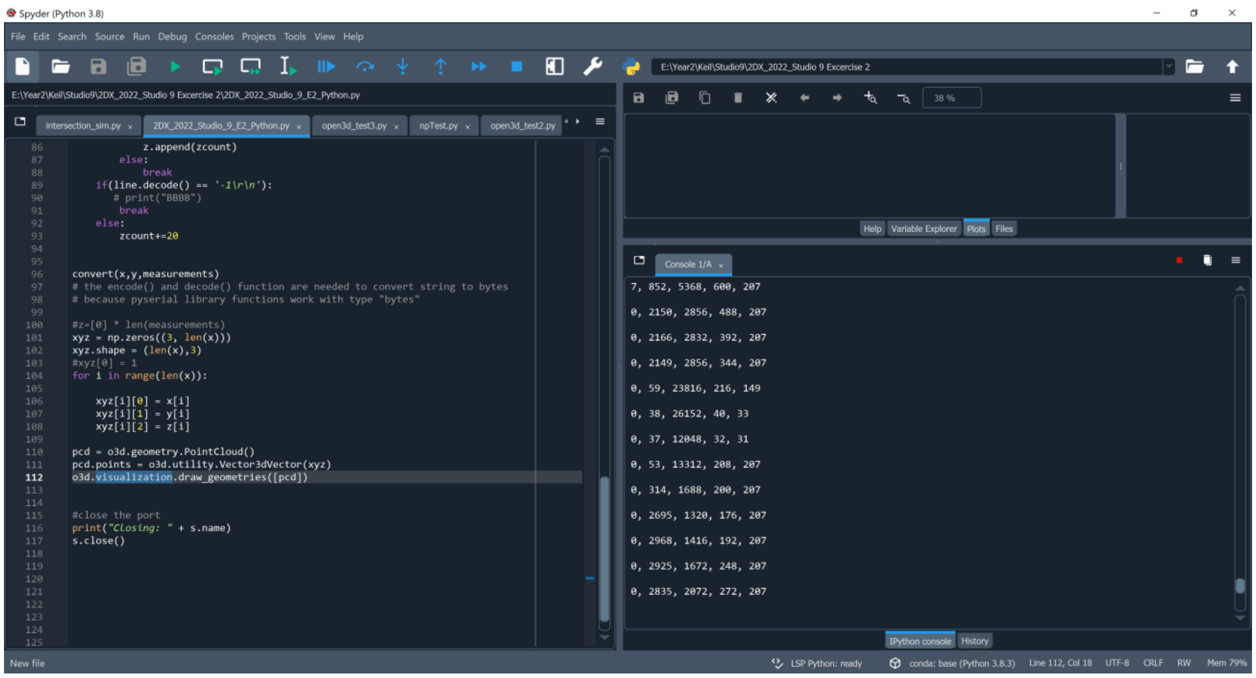
Block Diagram


Initialization
Initialize ToF sensor and Motor for distance acquisition.
Start
Waits for button input to begin program.
Data Acquisition
The motor turns counter clockwise, and pauses every 64th of a turn to acquire a ToF scan. This value is sent to the microcontroller using I2C, which is then sent to A PC using UART.
Data storage
A python program simultaneously listens for input from UART and stores the distance measurements as a list.
Stop
Check if the stop button has been pressed to proceed to visualization.
Visualization
The stored measurements are converted to cartesian coordinates and plotted using Open3d.

Device







Spyder (Python 3.8)
File Edit Search Source Run Debug Consoles Projects Tools View Help
E:\Year2\Keil\Studio9\2DX_2022_Studio 9 Excercise 2
E:\Year2\Keil\Studio9\2DX_2022_Studio 9 Excercise 2\2DX_2022_Studio_9_E2_Python.py
intersection_sim.py
2DX_2022_Studio_9_E2_Python.py
open3d_test3.py
npTest.py
open3d_test2.py
38 %
Help
Variable Explorer
Plots
Files
Console 1/A
IPython console
History
New file
LSP Python: ready
conda: base (Python 3.8.3)
Line 112, Col 18
UTF-8
CRLF
RW
Mem 79%

```python
                z.append(zcount)
            else:
                break
        if(line.decode() == '-1\r\n'):
           # print("BBBB")
            break
        else:
            zcount+=20


convert(x,y,measurements)
# the encode() and decode() function are needed to convert string to bytes
# because pyserial library functions work with type "bytes"

#z=[0] * len(measurements)
xyz = np.zeros((3, len(x)))
xyz.shape = (len(x),3)
#xyz[0] = 1
for i in range(len(x)):

    xyz[i][0] = x[i]
    xyz[i][1] = y[i]
    xyz[i][2] = z[i]

pcd = o3d.geometry.PointCloud()
pcd.points = o3d.utility.Vector3dVector(xyz)
o3d.visualization.draw_geometries([pcd])


#close the port
print("Closing: " + s.name)
s.close()
```

```
7, 852, 5368, 600, 207

0, 2150, 2856, 488, 207

0, 2166, 2832, 392, 207

0, 2149, 2856, 344, 207

0, 59, 23816, 216, 149

0, 38, 26152, 40, 33

0, 37, 12048, 32, 31

0, 53, 13312, 208, 207

0, 314, 1688, 200, 207

0, 2695, 1320, 176, 207

0, 2968, 1416, 192, 207

0, 2925, 1672, 248, 207

0, 2835, 2072, 272, 207
```

Device Characteristic Table

Bus Speed	24 MegaHertz
Stepper Motor Ports	PH0:PH3
Stepper Motor Power Source	5V
ToF I2C ports	PB2,PB3
ToF Power Source	3.3V
Libraries	Open3D, Numpy

Detailed Description

General

The PC used for this system is a Dell Inspiron 7559 with an Intel i7-6700HQ CPU and 16.0 GBs of RAM. The python code is run on Python 3.7.4 on the Windows 10 operating system. The entire system is created to wait for a button push to start executing the main body of the program. The pin PE0 is used as the scanning bit to activate the button for use; the button, PM0, must then be pressed for the program to begin. As implied earlier, the button press operates on a polling method, where the program is waiting for the button push such that it can move forward. When the button is pushed, bit 0 on port M is driven low, and this can be checked for by an if statement, where a successful button press will allow the program to continue to the main body.

Here, the motor will begin to turn, and measurements will be sent to the PC. There will be more details on the measurement portion next section. Once the program is in operation, the motor will turn a full rotation counter-clockwise, and then proceed to make a full rotation clockwise. This is so that the wires attached to the ToF sensor do not tangle. This process is repeated indefinitely until a second button press is executed. In the same manner as before, when a button press is detected, the motor will cease to turn, and an exit code will be sent to the PC. This will let the python code know that it should stop accepting new data, and plot the final results on the graph.

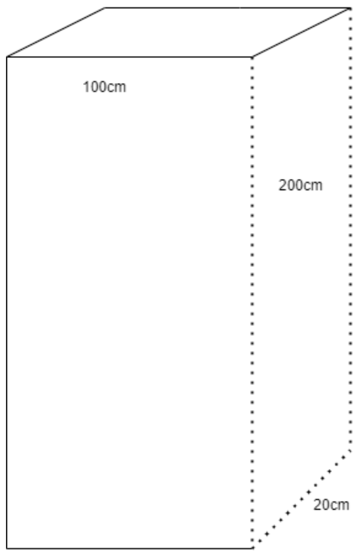
Distance Measurement

The distance measurement is done through the usage of the VL53L1X Time of Flight (ToF) sensor in tandem with a stepper motor. The ToF sensor creates a 940 nm laser, which is then picked up via a photon detector. Using the fact that the speed of light is known, and that the time taken for the light to be received back can be recorded, a simple application of kinematics can be used to determine the distance. This concept is called LIDAR, where it is similar to radar, except light is used instead of sound. As mentioned previously, the sensor is mounted onto a stepper motor, which rotates a full 360 degrees; every 5.625 degrees (1/64th rotation), a measurement is taken from the ToF sensor using the I2C protocol. This is done through the wires connected to SCL and SDA on the ToF sensor, which are connected to PB2 and PB3 respectively. The microcontroller receives these values from the sensor and they can now be manipulated in the C code. Here, the measurements are sent to the PC using the UART serial communication protocol, where the python code is waiting to pick them up. The python program begins its main body the moment it receives data via UART, since its default function is to wait for data to be sent to the PC. When the button is pushed a second time, an exit code will be sent to the PC via UART, where if the python code picks it up, it knows that it is time to stop waiting for input. Here it will move to the stage where it prepares the data for graphing.

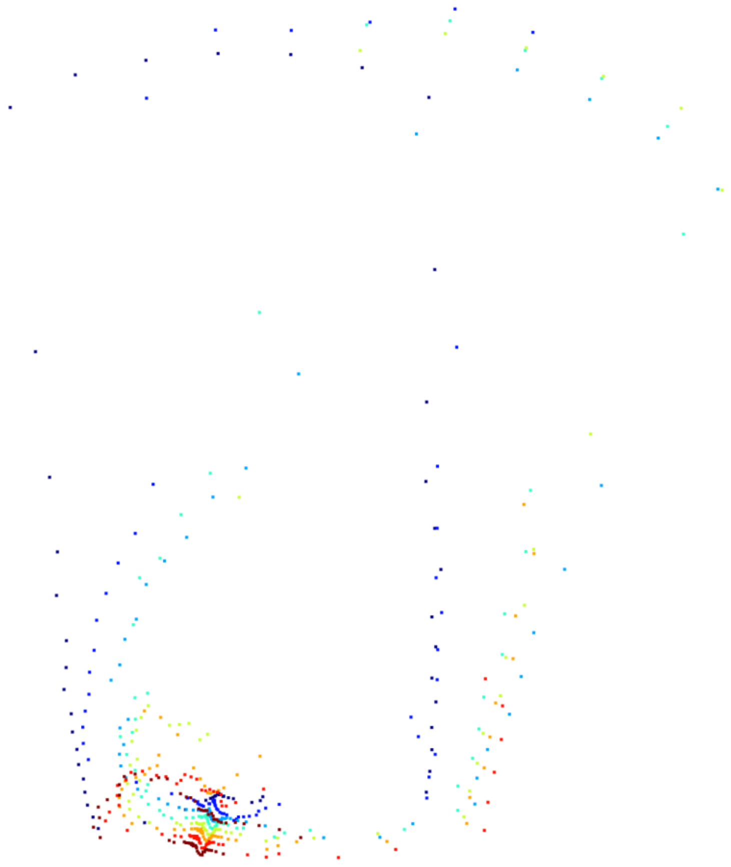
Visualization

From here, the data collected can be converted using the knowledge that each measurement is taken from a 64th of a turn. This means that the first measurement can be assumed to be on the x axis of the Cartesian plane with each measurement being taken 5.625 degrees from there. This allows for the usage of trigonometry to determine the x and y coordinates of the measurements. This is done by assuming the measurement is a vector who's horizontal and vertical components are the x and y coordinates. Since each measurement is consecutive, the value of 5.625 can be incremented to each measurement to get the equivalent degree of rotation from which the measurement was taken; by converting the degree value to radians, and taking the cosine and sine of the degree multiplied by the measurement, we can acquire the coordinates as per the earlier description. As for the z coordinates, it is assumed that the user moves the device across the z axis by 2 centimeters each rotation. This is because the rotation of the motor can only capture the surrounding slice of the device and therefore, the z axis must be assumed to be correctly traversed. From here, all Cartesian points have been acquired and they can now be used for graphing. The Numpy library is used to convert the points into a singular 2D array of points; this array can be given as a parameter for the visualization function of the Open3D library. This creates a 3D visualization of the distances measured by the device.

Isometric View



3D Reconstruction



User Guide and Application Example

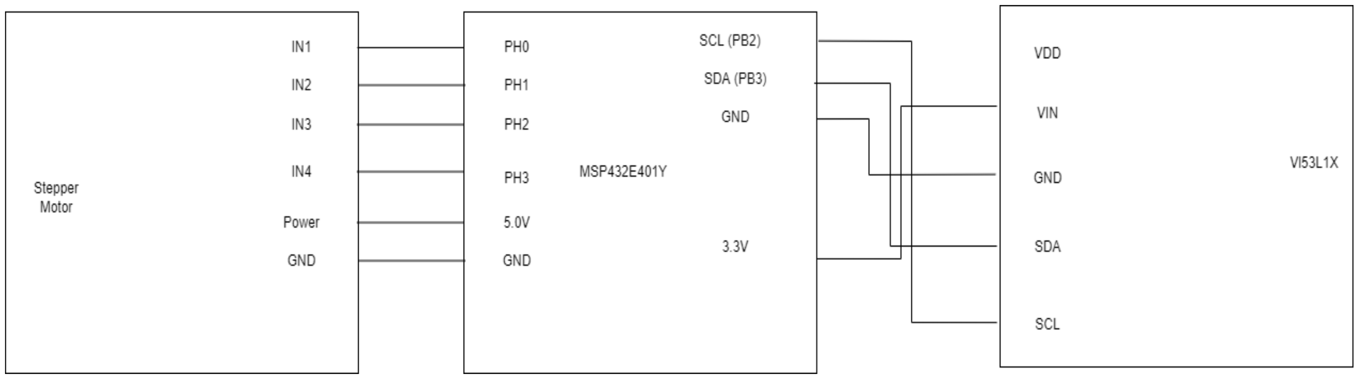
The creation of this device first requires the acquisition of the electronics utilized. These include, a PC that can run python 3.7, a breadboard and resistors to build a push button on, the Vl53l1X Time of Flight sensor, and the MSP432E401Y microcontroller. To begin, the push button must be placed in the middle of the breadboard such that it bridges the two halves. From here, connect a 10kΩ resistor from one leg of the button, to the positive line on the side. Between the resistor and the button leg, place a connective wire that will then be connected to PM0 on the microcontroller. Connect to the other leg on the same side of the breadboard, a wire that will be connected to PL0 on the microcontroller, and from the microcontroller, take a 3.3V power source and connect it to the positive terminal on the breadboard. Now the push button is functional. Connect the set wire that comes with the stepper motor to its chip, and connect IN1:IN4 to PH0:PH3 respectively. Connect the motors power source to 5V and ground to ground on the microcontroller. The ToF sensor should be placed on the rudder of the motor using the wire wrapping that is on the sensor and the wires should be connected to 3.3V and ground. From here SCL should be connected to PB2 and SDA should be connected to PB3. The hardware is now set for usage; make sure that the C code is flashed onto the microcontroller, and that the microcontroller is plugged into the computer for power. Open the python program to begin execution. Make sure that the correct port settings are being used in the python program. In the version of the device described in the report, the microcontroller was plugged into the first USB port on the left side of the machine, which corresponds to COM5. If unsure about the port being used for UART, look at the external port section on device manager for a more comprehensive understanding of what each port is being used for. From here, make sure the baud rate is set at 115200bps for proper acquisition of the data that is being sent. The software is now ready for execution.

Begin the program by running the python code on the machine, and pressing enter to begin UART communication. From here, the reset button on the microcontroller must be pressed to start the C program that is flashed onto it. A few lines will be sent to python to indicate that the C program has successfully booted up and is ready for use. Press the push button to begin the turning of the motor and acquisition of distance measurements. From here, the job of the user is to wait for a full rotation to occur, after which they should move the device forward in the direction that the motor is facing by two centimeters. The original direction in which the sensor is facing will be the x axis, and the direction perpendicular to that is the y axis. The direction that the motor is pointing will represent the z axis. After the user feels that they have acquired a sufficient length of data, they can press the stop button to exit the C program. The measurements will then be converted to Cartesian coordinates and displayed using Open3D. In the example presented in this report where a section of hallway is reconstructed, a 20cm portion of the hallway is taken. For the user to recreate such a measurement, they would need to take 10 consecutive measurements of the hallway, assuming that they move 2cm forward at each rotation.

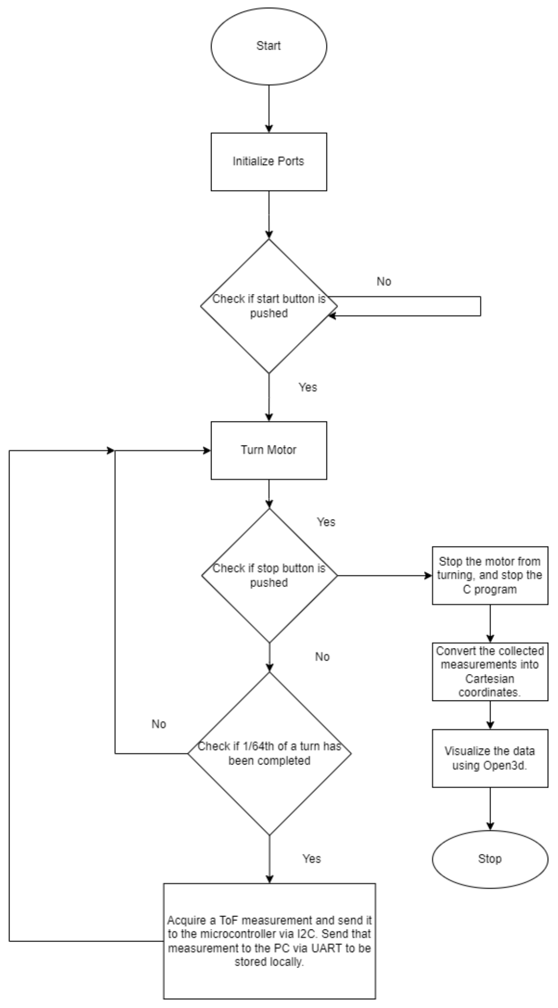
Limitations

1) The microcontroller, just like any binary system, is prone to floating point representation error. This is caused by the fact that it is sometimes floating point arithmetic that may occur internally cannot be converted to decimal in a 100% accurate manner and sometimes outputs the closest value it can provide. As for acquiring the trigonometric values for plotting, the value of PI which is provided in python is only accurate to a few decimal points, and is therefore only accurate to a certain degree.
2) The maximum quantization error of the ToF sensor can be calculated using the fact that it operates at 3.3V and sends 8 bits of data per transmission. Therefore, the error can be calculated as $V_i/2^n = 3.3V/2^8 = 0.01289$.
3) By referencing the microcontroller data sheet, the baud-rate generator allows speeds up to 7.5 Mbps [1].
4) The communication used between the microcontroller and the ToF sensor is the I2C protocol. The standard rate of transmission for the microcontroller is 100 kbps.
5) According to the answers to the previous questions, the I2C protocol speed in the microcontroller is slower than the UART protocol. Therefore, the I2C protocol, or in the case of this design, the sending of the measurement data to the microcontroller is the biggest limiter on speed.

Circuit Schematic



Flowchart


Start
Initialize Ports
Check if start button is pushed
No
Yes
Turn Motor
Yes
Check if stop button is pushed
Stop the motor from turning, and stop the C program
No
Convert the collected measurements into Cartesian coordinates.
No
Check if 1/64th of a turn has been completed
Visualize the data using Open3d.
Yes
Stop
Acquire a ToF measurement and send it to the microcontroller via I2C. Send that measurement to the PC via UART to be stored locally.

References

[1] Texas Instruments. (2018). MSP432E4 SimpleLink Microcontrollers – Technical reference Manual: Instruction Manual.