COMPENG 2DX4
Final Project

Instructor: Drs. Bruce, Haddara, Hranilovic, and Shirani

Julian Woo – L02 – wooj10 - 400192044

Google Drive Link

[link here]

1. Device Overview

1.1. Features

This device was constructed using a TI MSP432E401Y microcontroller in combination with a VL53L1X time-of-flight (ToF) module. The MSP432E401Y costs $71.96 CAD and has the following specifications:

- Default bus speed of 120MHz, (adjusted to 30MHz for this device)
- Operating voltage of 2.5 - 5.5V
- 1024KB of flash memory, 256KB of SRAM and 6KB EEPROM
- Two 12-bit SAR ADC modules, each up to 2 Msps
- Eight UARTs, each with a baud rate of up to 7.5 Mbps for regular speed and 15 Mbps for high speed

The VL53L1X module can measure distances up to 400cm away at a frequency of up to 50Hz, with an operating voltage of 2.6 – 3.5 V.

1.2. General Descriptions

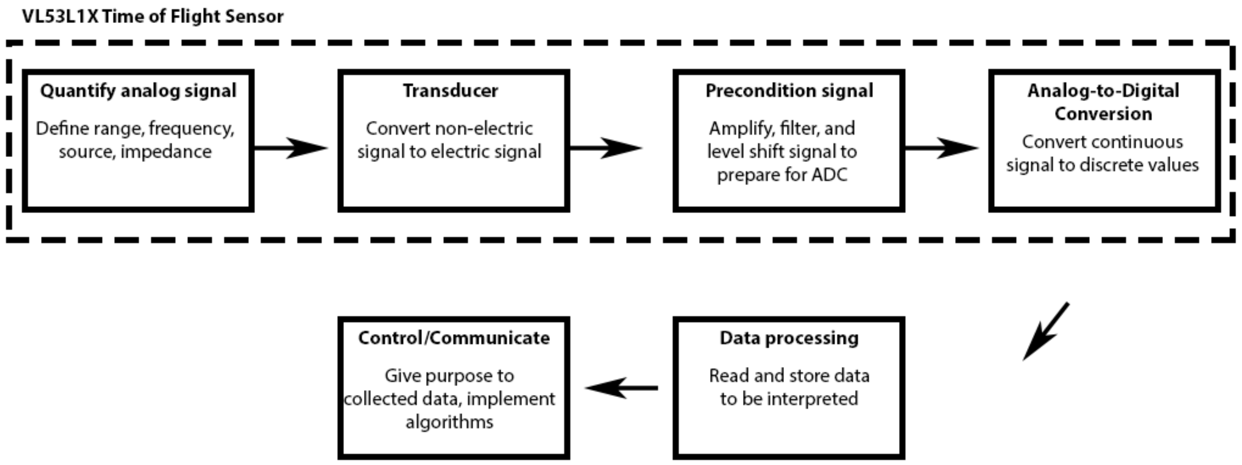
This device is an embedded system which integrates a ToF sensor to map enclosed indoor environments, such as hallways. This device is intended to be used as a component of other systems (e.g., robotics navigation, autonomous drone, layout mapping, etc.). Its core components include:

1. Digital I/O: momentary push button to control data acquisition
2. Digital I/O: LED status of each distance measurement
3. Transducer: ToF sensor to measure distance in y-z plane
4. Data processing: coordinate collection, computation, and storage of distance and displacement
5. Data processing: manually activated collection of new data, 360 degrees at a time, once defined fixed displacement reached
6. Implementation mode: interrupt design implementation
7. Control: rotation of stepper motor controlled to facilitate data collection with ToF sensor
8. Communicate/Control: data communicated between ToF sensor and microcontroller
9. Communicate/Control: data communicated to PC application, distance and displacement data stored
10. Control: data read from Excel spreadsheet and visualized using Open3D
11. Control: data imported to MeshLab to generate 3D model in .stl format

Please note that, due to hardware issues, a supplied Excel dataset was used. However, for the purposes of the source code and this document, the data that would be collected by the ToF will be stored in an Excel sheet formatted the same way as the given dataset.

There are several stages of this device’s operation. The VL53L1X ToF sensor handles all distance (y-z plane) data acquisition. The signal is captured via transducer. The captured signal is then processed and discretized from an analog to a digital signal. Each of these components is managed by the ToF sensor. Once the digital measurement has been acquired, it is sent to the PC, processed into an excel document, and visualized using Open3D. The displacement (x) data is captured manually, moving the device in increments of 0.2m in the -x direction.

After the data has been acquired by the ToF sensor, it is communicated to the PC using serial communication. Operating in interrupt mode, the sensor sends data via UART, one byte at a time. The PC receives this data using a Python script, which then stores the measured distances in an Excel sheet (i.e. the provided dataset). The data from that sheet is read into a separate Python script that then visualizes the data using Open3D. A .stl file was generated using MeshLab.


VL53L1X Time of Flight Sensor
Quantify analog signal
Define range, frequency, source, impedance
Transducer
Convert non-electric signal to electric signal
Precondition signal
Amplify, filter, and level shift signal to prepare for ADC
Analog-to-Digital Conversion
Convert continuous signal to discrete values
Data processing
Read and store data to be interpreted
Control/Communicate
Give purpose to collected data, implement algorithms

Block diagram of device operation

2. Device Characteristics

Feature	Detail
Ext. LED Pin	PL5
Motor Driving Pins	PE0 – PE3
SCL/SDA Pins	PB2/PB3
Bus Speed	30MHz
Communication Port	COM3
Communication Speed	115.2 kbps

To setup this device, please see Section 4.2 Setup. The user must assemble the circuit seen in Figure 1, flash the project code onto the microcontroller, and run the Python script responsible for opening the correct serial communications port and reading and storing the data from the ToF sensor.

3. Detailed Description

3.1. Distance Measurement

Collecting distance measurement and providing a useful output generally involves three components: a transducer, to interpret the non-electric signal as an electric signal; a pre-conditioning circuit, to filter that electrical signal into something that can be more easily converted into a digital form; and an analog-digital converter (ADC), to discretize the conditioned analog signal into a digital format. The VL53L1X time-of-flight sensor incorporates all three of these components into its design, outputting a distance in millimeters. To create this data, the module outputs a light signal and waits for it to return to the sensor. The sensor works best in a dark room for this reason. The distance is calculated from the amount of time that it takes for the signal to return, after it has been preprocessed and discretized via the built-in ADC. This distance data is outputted after each step of the motor, sent to the PC via UART. The PC receives this data through a waiting Python script (receivedata.py), which opens serial communications port COM3 at a baudrate of 115.2 kbps, 8N1. It writes the data to an Excel sheet as it receives it using a Python library called xlsxwriter. The Excel sheet is formatted so that each row represents the stepper motor at a certain angle within a plane, and each column is a separate plane of displacement. The operation of receivedata.py is described in the form of a flowchart in Figure 3. The reformatting of the measurement data is accomplished in visualizedata.py, but the process of calculating (x, y, z) coordinates from the raw measurement data will be described in this subsection. Seeing Section 4.3 Defining Axes as a guide, the raw data must be reinterpreted to represent a point in a series of y-z planes. These coordinates can be determined using trigonometry. The two components of the calculation are the angle and the distance measurement itself. The angle can be determined from the step of the motor as follows:

$$angle = \frac{360}{512} * motor\ position + 270\ degrees$$

Each step of the motor increments the angle by 360/512 degrees, as there are 512 steps in a full rotation. An additional 270 degrees are added to represent the motor beginning facing straight down. This angle is in degrees, but is converted into radians in the Python script to be used with Python's math library. To find the y and z coordinates, one must incorporate the raw distance measurement as follows:

$$y = \frac{distance}{1000} * \sin(angle)$$

$$z = \frac{distance}{1000} * \cos(angle)$$

The distance is divided by 1000 to convert the raw measurement data from millimeters to meters. As the y axis is pointing straight up and z axis is pointing straight to the right, their components are found by the sine and cosine of the angle, respectively. The full operation of visualizedata.py can be found in the following Section 3.3 Visualization, as well as in Figure 4.

3.2. Displacement Measurement

An IMU sensor, standing for Inertial Measurement Unit, measures angular rate and direction of the device. It would effectively implement the x/displacement axis of the device. IMU sensors are effectively a combination of a gyroscope, magnetometer, and accelerometer. The linear acceleration is measured with accelerometer(s), and the rotational axis is noted using a gyroscope. The MPU-9250 module specifically implements three accelerometers and three gyroscopes, one of each for each axis (x, y, z). Each internal sensor feeds into an ADC that discretizes the data signal. Instead of using the MPU-9250 IMU module, however, a manual approach was taken. Each distance measurement was taken at a manually determined displacement, starting at x = 0 and incrementing by 0.2m in the -x direction up to x = -3.8m. At each displacement, a set of 512 distance measurements is taken, one for each step of a full rotation of the motor. These datapoints effectively create a vertical slice of the environment in the y-z plane at each displacement. These planes are visualized and connected in the following Section 3.3. Visualization.

3.3. Visualization

The visualization was achieved on a Dell XPS 15 9570 (Intel Core i7-8750H, GTX 1050 Ti) running Windows 10 and Python 3.6. The Python libraries used were as follows:

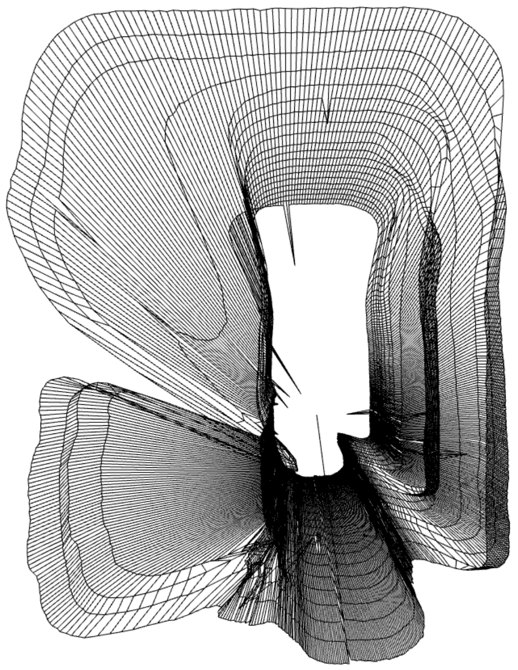
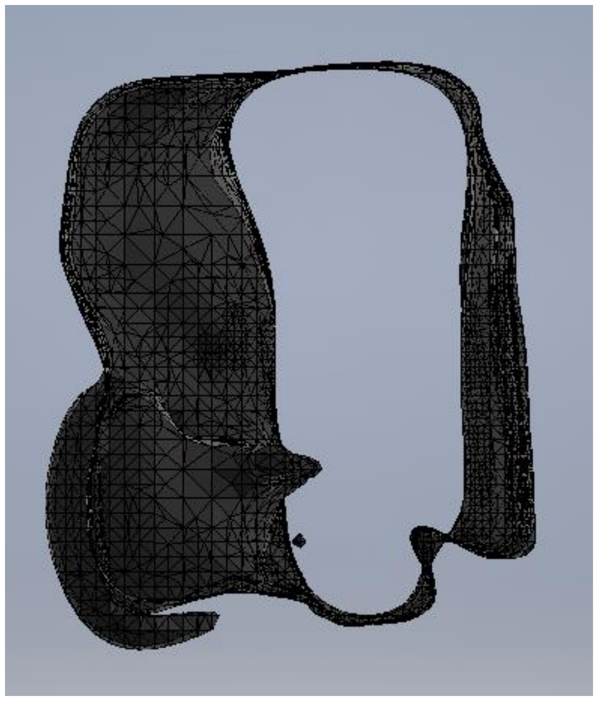
- The Python math library, used for trigonometric functions
- xlrd, used to read data from Excel spreadsheet
- open3D, used to represent data as point cloud and visualize
- numpy, used to represent point cloud as array

The measurement data is stored in an Excel spreadsheet (see 3.1. Distance Measurement), and so it must be read into the script first. This is achieved using the xlrd library, reading all of the distance data for one displacement plane (one full rotation) into an array. Each plane will have 512 data points, given that the stepper motor must undergo 512 steps to achieve one full rotation. This distance data is converted into [x, y, z] coordinates and written to a .xyz file. The x value, manually determined from earlier, is consistent for each point in a y-z plane; the y and z coordinates are determined trigonometrically from the number of steps (from which the angle can be determined) and the distance measurement. The process is repeated for each plane, 20 planes in total. Using Open3D, a point cloud is read from the new .xyz file. Next, a line set must be created for the visualization. Each point in each plane is linked to the next point in the same plane, making each plane continuous. Additionally, each point in each plane is linked to the same point in the next plane, linking the planes together. The point cloud and lineset are used together to create a visualization. The .xyz file was then imported into MeshLab, where normals for the point set were calculated (neighbour num increased to 100 to ensure accuracy after trial and error) and the surface was reconstructed using the screened Poisson method. The result was exported into a .stl file. For a flowchart of visualizedata.py’s operation, see Figure 4.

4. Application Example

4.1. Example

1. Place device at one end of a dark hallway, or other suitable room.
2. Run I2C data reception Python script on PC.
3. Press push button to begin data collection for first plane and wait for the device to complete a full rotation.
4. Move the device up 0.2m in the -x direction (see 4.3.).
5. Repeat steps 2-3 for twenty sets of measurements, making sure to move the device the same amount each time.
6. A full dataset has now been collected and stored locally on the PC. Run visualizedata.py to display a 3D representation of the data points. A flowchart describing the operation of visualizedata.py can be found in Figure 4.
7. To generate .stl file from the data, open Meshlab and import the points.xyz file generated by the visualization script as a mesh.
8. Navigate to Filters → Normals, Curvatures and Orientation → Compute normal for point sets. Set “neighbour num” parameter to a suitably large number (100 in this case) and apply.
9. Navigate to Filters → Remeshing, Simplification and Reconstruction → Surface Reconstruction: Screened Poisson. Apply with default settings.
10. A surface has now been generated from the datapoints. To create a .stl file, navigate to File → Save As… → your_filename.stl. Open in any CAD software to view.

Open3D output (left) vs .stl representation in Autodesk Inventor 2017 (right)

4.2. Setup Guide

1. Assemble the circuit shown in Figure 1. Note that the pin connections are as described in Section 2. Device Characteristics.
2. Attach the VL53L1X module to the rest of the device using the wired slip ring with flange.
3. Flash the project code onto the MSP432E401Y. A flowchart describing the operation of the project code can be found in Figure 2.
4. Run the Python script receivedata.py. This script will open COM3 at a baud rate of 115.2kbps 8N1 and wait to receive data. A flowchart describing the operation of receivedata.py can be found in Figure 3.
5. The device is now setup and ready to be used. Please see Section 4.1. Example above to see an example of the operation of this device.

4.3. Defining Axes

The coordinates are defined by x, y, and z coordinates. The positive x direction is pointing towards the user, the positive y direction is pointing up, and the positive z direction is pointing to the right. Each set of measurements is taken along the y-z plane, with each displacement being in the -x direction.

5. Limitations

1. On most microcontrollers, if floating-point values are supported, they will have a floating point unit (FPU) designated specifically for the task. The MSP432 series has an IEEE754 32-bit FPU integrated into the Cortex-M4F Core, meaning that it can support floating points with up to 32 bits of precision. The microcontroller is capable of performing trigonometric functions in theory, but these operations were performed in a Python script due to the difference in processing power and ease of use.

2. Maximum quantization is equivalent to resolution. Resolution can be calculated as
$$\frac{V_{FS}}{2^m}$$
Where m = number of bits and VFS = full-scale voltage (5 volts in our case). The ToF sensor has an 8-bit ADC, the MSP432 has a 12-bit ADC, and the IMU sensor has a 16-bit ADC, giving us:
$$ToF\ res. = \frac{5}{2^8} = 19.53\ mV$$
$$MCU\ res. = \frac{5}{2^{12}} = 1.22\ mV$$
$$IMU\ res. = \frac{5}{2^{16}} = 76.30\ \mu V$$

3. The maximum standard serial communication rate is the one that was implemented in this project, 115.2 kbps. This is fast for the majority of microcontrollers, MSP432E401Y included. This maximum speed was verified when higher speeds were attempted, and errors were noticed in transmission using RealTerm.

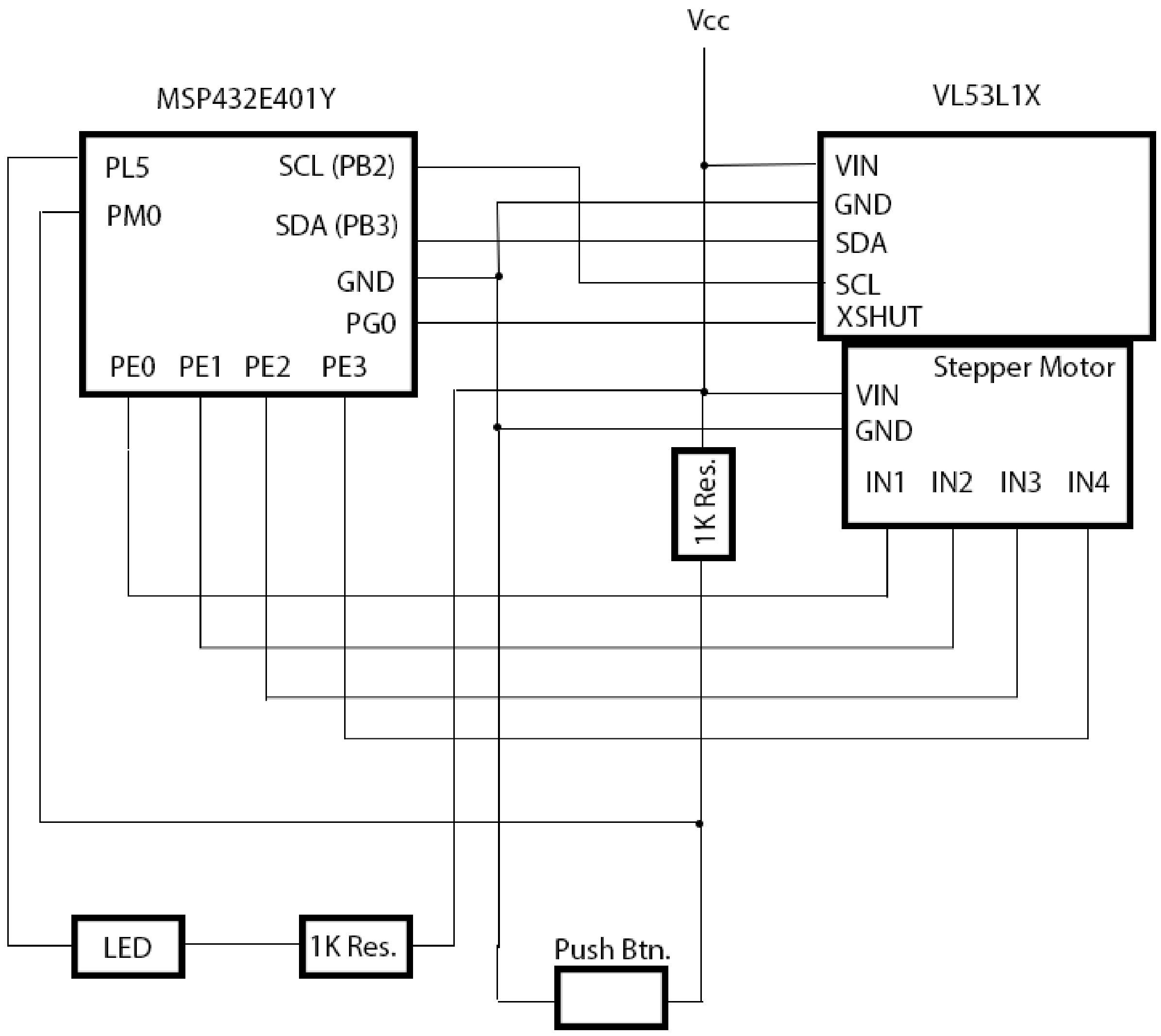
4. This device used I2C to communicate between the ToF sensor and the PC. One of the MSP432E401Y's eight UARTs was used, implementing asynchronous serial transmission. The communications port used was COM3, and the baudrate was set at 115.2 kbps, 8N1. The microcontroller would send one character/byte of data at a time across COM3 to a waiting Python script, which would receive that data and process it.

5. The speed of the microcontroller (specifically the CPU) and the speed of the user's PC have the greatest impact on performance.

6. The Nyquist Rate must be at least double the frequency of the bus speed, which is 80MHz. Therefore, the Nyquist Rate is equal to 160MHz. A higher sampling rate may increase the clarity of the signal read at the cost of more power/memory.

6. Circuit Schematic

Figure 1: Circuit schematic



7. Programming Logic Flowcharts

Figure 2: Microcontroller flowchart

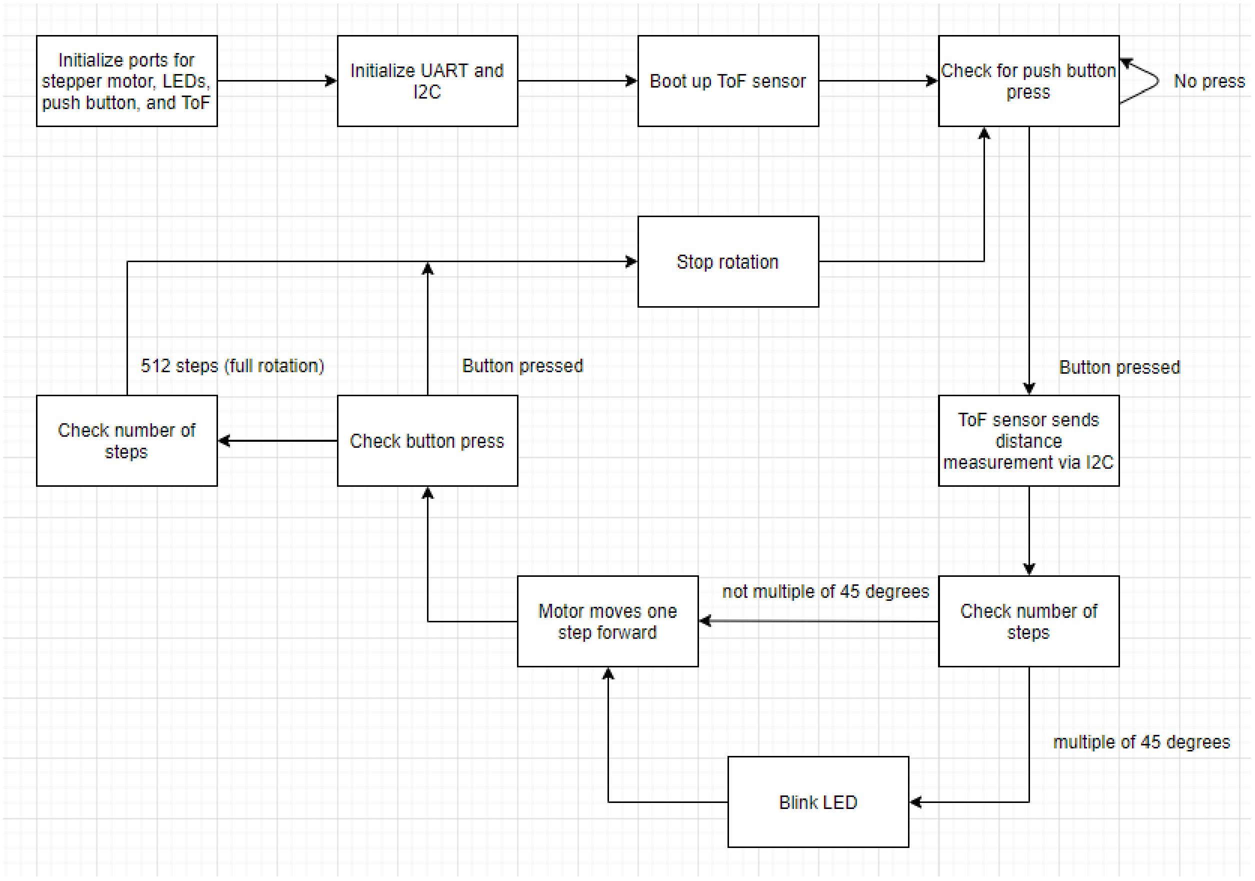

Initialize ports for stepper motor, LEDs, push button, and ToF
Initialize UART and I2C
Boot up ToF sensor
Check for push button press
No press
Stop rotation
512 steps (full rotation)
Button pressed
Button pressed
Check number of steps
Check button press
ToF sensor sends distance measurement via I2C
Motor moves one step forward
not multiple of 45 degrees
Check number of steps
multiple of 45 degrees
Blink LED

Figure 3: Python receivedata.py flowchart

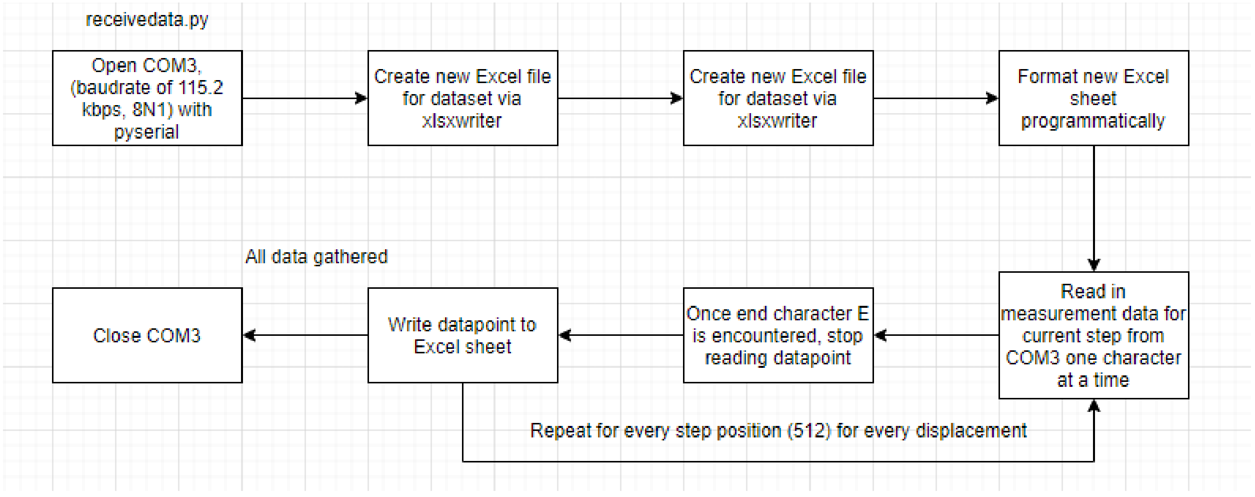
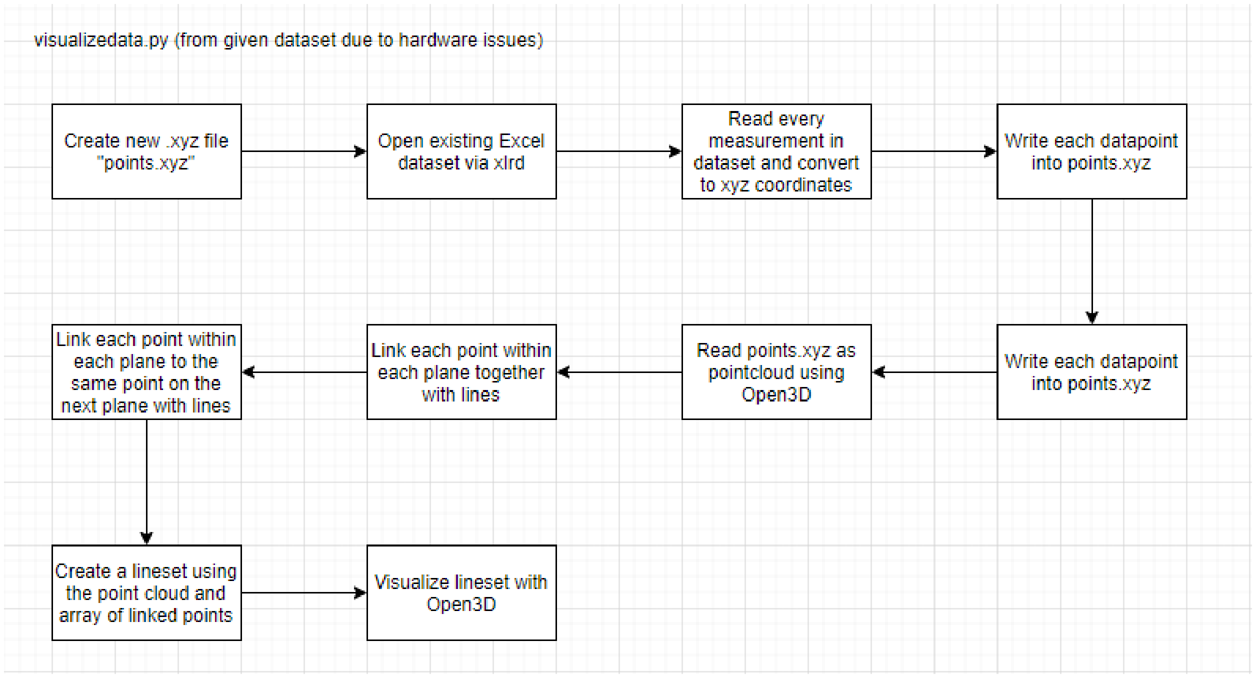

receivedata.py
Open COM3, (baudrate of 115.2 kbps, 8N1) with pyserial
Create new Excel file for dataset via xlsxwriter
Create new Excel file for dataset via xlsxwriter
Format new Excel sheet programmatically
All data gathered
Close COM3
Write datapoint to Excel sheet
Once end character E is encountered, stop reading datapoint
Read in measurement data for current step from COM3 one character at a time
Repeat for every step position (512) for every displacement

Figure 4: Python visualizedata.py flowchart


visualizedata.py (from given dataset due to hardware issues)
Create new .xyz file "points.xyz"
Open existing Excel dataset via xlrd
Read every measurement in dataset and convert to xyz coordinates
Write each datapoint into points.xyz
Write each datapoint into points.xyz
Read points.xyz as pointcloud using Open3D
Link each point within each plane together with lines
Link each point within each plane to the same point on the next plane with lines
Create a lineset using the point cloud and array of linked points
Visualize lineset with Open3D