Table of Contents

1. Device Overview

a. General Description

This project seeks to create an inexpensive and compact alternative to Light Detection and Ranging (LIDAR) equipment using a sensor accompanied with a rotary mechanism. Through a TI-MSP432E401Y microcontroller, VL53L1X time-of-flight (ToF) sensor and a MOT-28BYJ48 stepper motor, the device can collect 3-dimensional spatial data in within a single plane. These components are accompanied by a 2-button off-board control system that can control the stop/start data collection and spin the stepper motor, or to unwind the motor and untangle wires. The ToF sensor collects distance data and stored to the microcontroller and finally transmitted to a personal computer through a wired micro-USB connection. Using Python 3.9 and associated libraries, the computer can parse the collected data and using the open3D library and display a 3D reconstruction of area scanned.

The operations of the stepper motor and ToF sensor are both hardwired to the microcontroller thus, it serves as the main control hub of the device. The microcontroller manages all operations of the peripherals and is responsible all steps from data acquisition to data transfer where the computer later generates a 3D reconstruction based on the data collected. Figure 1 illustrates the responsibility of the microcontroller through a flow diagram that shows the lifecycle of data transfer from signal acquisition to its destination at a digital system for further data manipulation.

The device acquires 360º data from the ToF sensor which takes periodic measurements while being mounted on the stepper motor. The VL53L1X ToF sensor has two gold-plated areas on its front-facing side, and they are used to reflect and redirect a photon by the sensor's internal light source, allowing the sensor to measure distances accurately. After each 11.25º interval, the stepper motor stops, and a measurement is taken. As such, for each data collection set of a 360º cycle, 32 distance measurements are taken. A higher number of data points collected yields a better image resolution. The ToF sensor collects the distance data by converting the analog signal to digital, and it is transmitted to the microcontroller via the I2C communication protocol. The microcontroller then communicates the data, peripheral status, and device state to the computer via the UART communication protocol.

The overall cost of the device totals around just under $75USD as the VL53L1X ToF sensor costs $14.95, the MOT-28BYJ48 stepper motor and UNL2003 Driver costs $6.95 whilst the MSP-EXP432E401Y costs $39.99. This alternative to LIDAR equipment is more compact, cost-effective thus making it ideal for indoor applications.

b. Features

This device performs two main operations: *scanning*, and *graphing*. The scanning portion is performed using a time-of-flight (ToF) sensor mounted on a stepper motor rotating 360°. The ToF sensor acquires 360-degree, distance measurements of the area within a single vertical geometric plane (y, z) and graphs it onto a geometric plane along the orthogonal x-axis. The data collected from the ToF sensor is later visualized on a personal computer through Python using the Open3D library.

The device has several individual components with features that include but are not limited to:

- Texas Instrument MSP432E401Y microcontroller.
    - 20 MHz Bus speed.
    - 2.5-5.5V operating voltage.
    - Arm® Cortex®-M4F 32-bit Processor Core.
    - Logic programmed in C within Keil environment.
    - 1024 KB of Flash Memory & 256 KB of SRAM.
- MOT-28BYJ48 Stepper Motor & UNL2003 Driver
    - 512 steps per 360º rotation.
    - 5-12V operating voltage.
    - UART communication protocol.
- VL53L1X Time-of-Flight Sensor
    - Capable of measuring distances up to 4000mm at 50Hz update rate.
    - 2.5-5V operating voltage.
    - I2C communication protocol.
- Communication Protocols
    - I2C serial communication protocol used to communicate data between ToF sensor and MSP432E401Y microcontroller.

    - UART serial communication protocol used to communicate data between MSP432E401Y microcontroller and computer with baud rate of 115200 bps.
- Associated Software
    - *data_3D_visualiser.py* and *data_collection.py* written in Python3.9.
    - Open3D library used for spatial reconstruction3D data processing and 3D image generation.
- Misc.
    - Total Cost: ~$75USD
    - 2 external push buttons to control device operation: to stop/start stepper motor; to unwind motor to untangle wires.
    - Published with MIT license.

c. Block Diagram (Data flow graph)

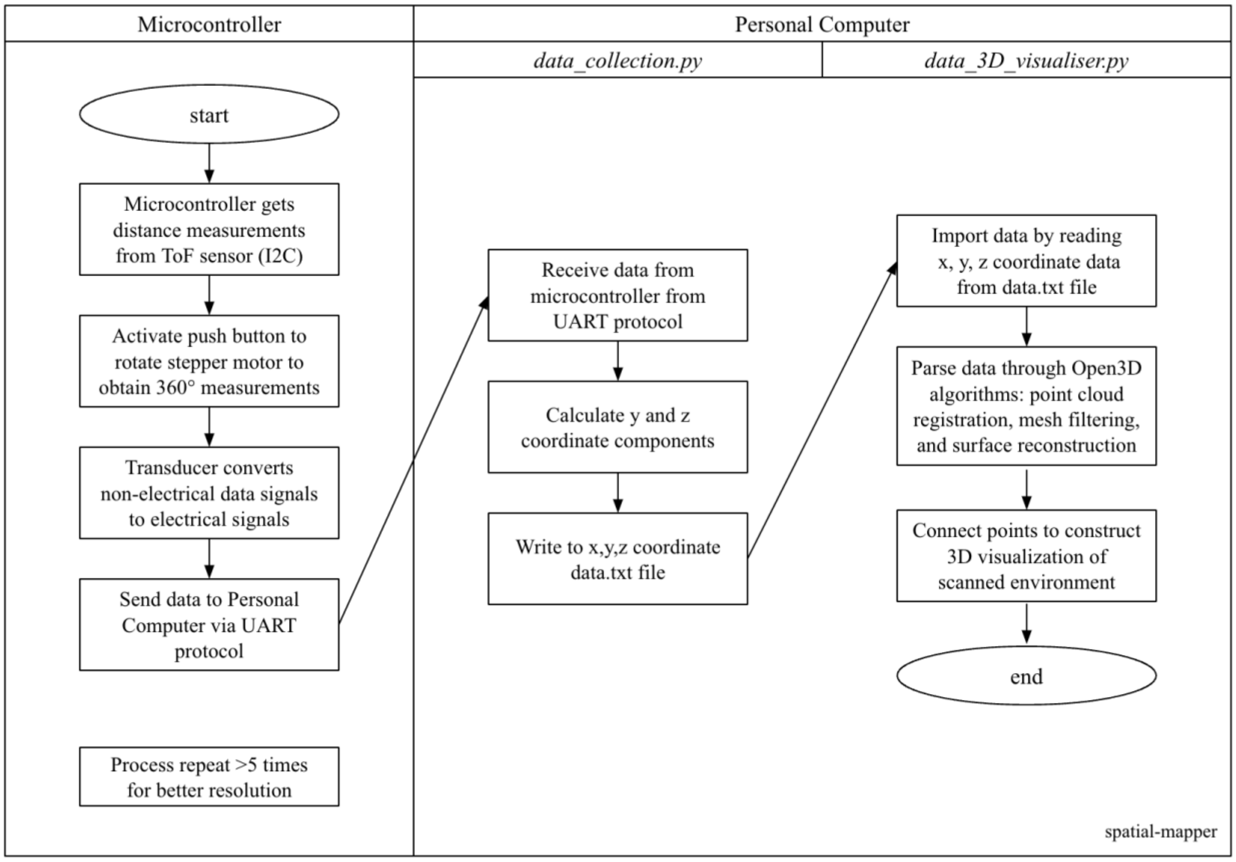

Microcontroller
Personal Computer
data_collection.py
data_3D_visualiser.py
start
Microcontroller gets distance measurements from ToF sensor (I2C)
Activate push button to rotate stepper motor to obtain 360° measurements
Transducer converts non-electrical data signals to electrical signals
Send data to Personal Computer via UART protocol
Process repeat >5 times for better resolution
Receive data from microcontroller from UART protocol
Calculate y and z coordinate components
Write to x,y,z coordinate data.txt file
Import data by reading x, y, z coordinate data from data.txt file
Parse data through Open3D algorithms: point cloud registration, mesh filtering, and surface reconstruction
Connect points to construct 3D visualization of scanned environment
end
spatial-mapper

Figure 1: Block Diagram for Data Flow

2. Device Characteristics Table

Peripheral/Program Specification	Details	
MOT-28BYJ48 Stepper Motor UNL2003 Driver	**UNL2003**	**MSP432E401Y**
	IN1	PL0
	IN2	PL1
	IN3	PL2
	IN4	PL3
	+	5V
	-	GND
VL53L1X ToF Sensor	**VL53L1X**	**MSP432E401Y**
	V_{IN}	3.3V
	SCL	PB2
	SDA	PB3
	GND	GND
External Pushbutton	**Function**	**MSP432E401Y**
	Stop/Start	PH0
	Unwind	PH1
Python Version	3.9	
Python Libraries	numpy, pyserial, open3d	
MSP432E401Y Bus Speed	20MHz	
Baud Rate	115200 bps	
ToF Sensor I2C Address	0x29	
Serial Port	COM3	

Figure 2: Device Characteristic Table

Figure 2 above indicates the details regarding the physical implementation of each hardware and software component related to the device. Figure 3 below is the physical implementation that employs the configurations outlined in the table above. Refer to Figure 8 for a schematic of the physical circuit implementation.

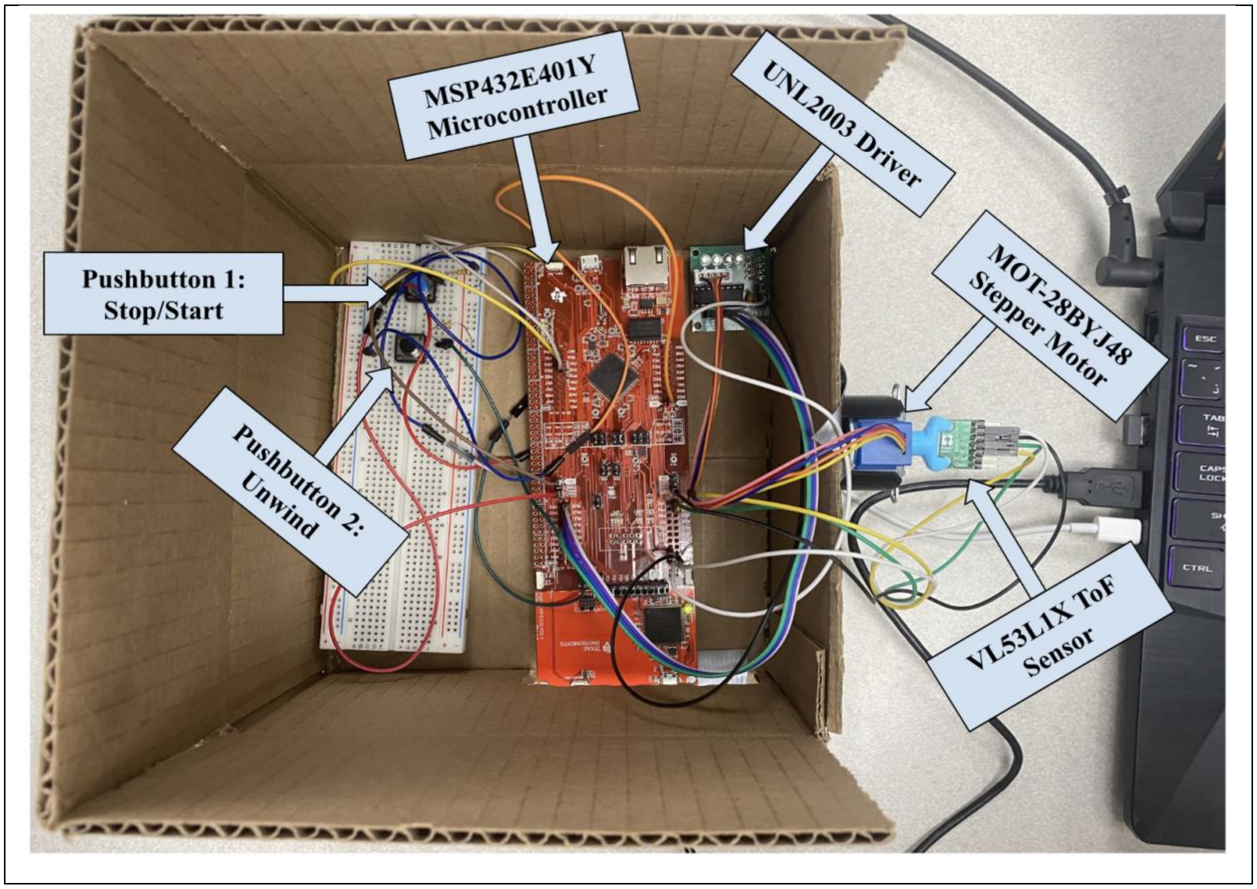


Figure 3: Spatial mapper, with components annotated.

3. Detailed Description

a. Distance Measurement

As previously mentioned in section 1.b, the VL53L1X ToF sensor has two gold-plated targets which are used to emit the light beam and receive it. The ToF sensor emits light pulses also known as photons to measure the distance to an object. The sensor emits a photon towards a direction and when it reaches a surface, the photon reflects, which gets received at the receiver target plate. This time that the photon travelled for is measured then used to calculate the distance to the object using the speed of light. The ToF sensor can accurately measure the time of flight up to picoseconds, thus it can provide distance measurements with a high degree of precision, typically in the range of 1 mm.

This is calculated by multiplying the speed of light, c, by the time it took for the photon to travel between the two sensors and dividing the result by 2. Dividing the product by 2 gauges the one-way displacement of the photon rather than the round trip. As the speed of light is a constant equal to: $3.0 \times 10^8\ m/s$, we can conclude that the distance travelled is proportional to the measured time for the photon to return to the sensor.

$$distance\ travelled = 0.5 \times c \times time$$

The measured distance is stored in the *distances[]* array to later be used to calculate the y, z coordinates in its relative space. By mounting the ToF sensor onto a MOT-28BYJ48 Stepper Motor, the device can collect 360º data. The MOT-28BYJ48 Stepper Motor is connected to the UNL2003 Driver which operates the motor through alternating the polarities of the motor's 4 internal electromagnets. By changing the polarity, it enables the shaft between the magnets to enable the rotational motion. The switching of polarities is called full step as two adjacent electromagnets will be powered at a time. The motor has 6 ports, Vcc, GND, IN1, IN2, IN3, IN4 which connect to 5V, GND, PL0, PL1, PL2, PL3 on the microcontroller, respectively. The mounted ToF sensor has 7 ports, however, only 4 are utilized to operate it, namely, Vin, GND, SDA, SCL which are connected to 3.3V, GND, PB2, PB3 on the microcontroller respectively. As depicted above in Figure 2, the data flow begins from the microcontroller which is flashed by the Keil project then the data is collected using *data_collection.py* which is further transmitted to the computer where the data is used to reconstruct the 3d model with the *data_3D_visualizer.py* module.

This data collection process is performed a set number of times between each motor state change as pre-set by variables: *SETS* and *RESOLUTION* in both *data_3D_visualizer.py* and *data_collection.py.* For each instance of the sensor collecting data, the motor stops for 5 milliseconds to ensure a stable scan was taken. Regarding the application example outlined in section 4.a, *RESOLUTION* was set to 32 while *SET* was set to 5, thus the device will take 32 measurements every 360°; one complete 360° rotation is 1 set. A resolution of 32 means data will be collected after each 11.25° rotation of the motor.

$$step\ degree = \frac{360°}{RESOLUTION}$$

After communication is established between peripheral devices and the microprocessor, data collection begins once the scripts are executed. An infinite loop is induced such that the code reads and parses the raw data from the serial port in string format. After each increment of the stepper motor, the corresponding x, y, z coordinates are calculated via trigonometric functions and mapped as polar coordinates. The following equations are representative of the data processing to obtain the coordinates to be mapped.

$$x = number\ of\ completed\ sets \times \Delta x/set$$
$$y[i] = \ distance[i] \times \sin\,(\frac{2\pi}{360°} \times \ angle[i])$$
$$z[i] = \ distance[i] \times \cos\,(\frac{2\pi}{360°} \times angle[i])$$

The x coordinate is orthogonally determined based on the devices relative position in relation to the y and z measurements. As such, for a raw distance measurement of 1240 mm at an angle of 22.5° while on the 3rd set with a Δx of 10 cm, the respective x, y, z coordinates are:

$$x = 3 \times 100mm = 300mm$$
$$y = \ 1240mm \times \sin\left(\frac{2\pi}{360°} \times \ 22.5°\right) = 474.527mm$$
$$z = \ 1240mm \times \cos\left(\frac{2\pi}{360°} \times 22.5°\right) = 1145.611mm$$
$$[x, y, z] = [300, 474.527, 1145.611]$$

Once the data is processed, they are written to *distance_data.txt*, and the *data_3D_visualizer.py* script is executed to create the 3D special reconstruction.

b. Visualization

From *data_collection.py,* the raw and process data is written to *distance_data.txt* as outlined in Figure 2 and Section 3.a. *data_3D_visualizer.py* is the python script that reads the data collected from *distance_data.txt* to create a 3D visualization of the collected data points. This script uses the Open3D library which reads the (x, y, z) coordinate data points to a point cloud object. This is done by calling the *read_point_cloud()* function that read from aforementioned *distance_data.txt* stored in variable *pcd.* The script adds additional points to the point cloud for boundary boxes and midpoints. The boundary boxes are defined by 4 points that create a rectangle and the midpoint is defined by a single point. These connections are added to an array, *lines,* which is then used to create a LineSet object. The processed coordinate points are visible on the 3D space however, they require connecting lines to clarify each set. *line_set* objects were created by which contained every point and a line connecting to an adjacent point. The series of *line_set* objects allows ease for the 3D reconstruction by calling the *open3D.visualization.draw_geometries()* function. By running the module, we are able to see the Open3D reconstruction.

4. Practical Use

a. Application Example

For this specific use-case exemplar, the ITB A180 hallway was `scanned as seen below in Figure 4 denoted by section A. Figure 5 is the photographic representation of the hallway A180 as illustrated by section A from Figure 4. Figures 6 and 7 illustrate the Open3D reconstruction of hallway A180. For this reconstruction, 5 sets of 32 measurements were taken by taking one step forward along the orthogonal x-axis.

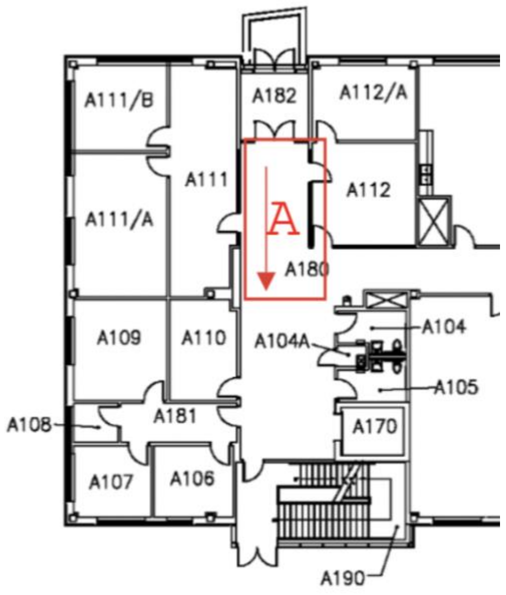


Figure 4: Floorplan of Section A

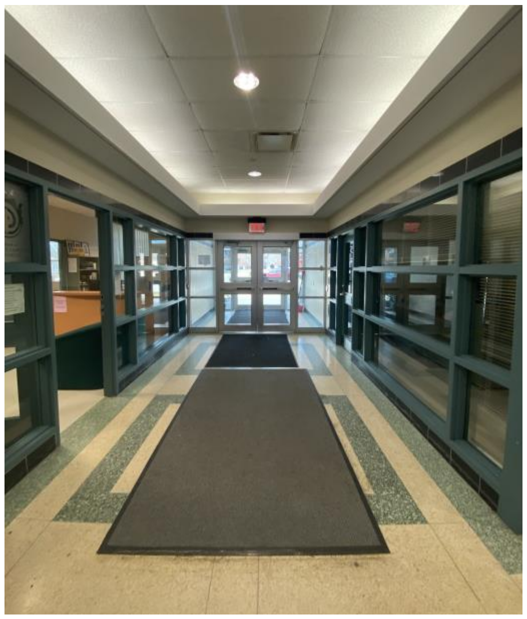


Figure 5: Photo of hallway covered by Section A

b. User Guide Instructions

1. Install Python version 3.9 on local machine.
2. Install associated Python libraries: *numpy, pyserial, open3d* by executing the following in cmd.

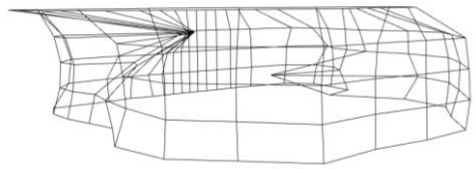
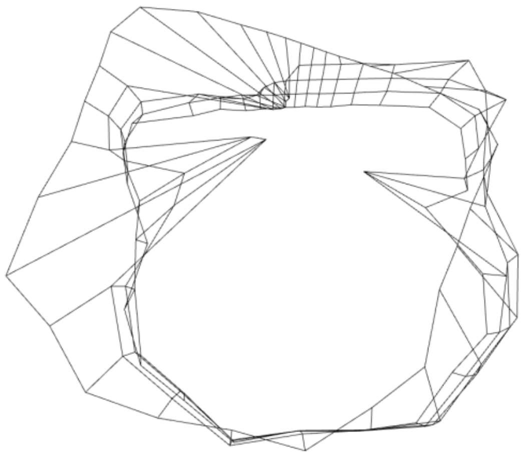
   ```
   pip install numpy

   pip install pyserial

   pip install open3d
   ```

3. Establish connection between microcontroller to local machine via micro-USB cable.
4. Open *Final.uvoptx* Keil project in Keil environment.
5. Open *data_collection.py* and *data_3D_visualizer.py* in IDLE environment.
6. Flash, Build and Load *main.c* in Keil then execute *data_collection.py* module.
7. Reset the microcontroller then wait until 'Set-1' appears on IDLE terminal or check to see all on-board LEDs are off.
   a. LED4 will flash while set up is initializing.
8. Ensure your device is properly secured and set up then press the pushbutton with a connection to PN0 once.
9. Wait for the stepper motor to complete a complete 360º rotation.
   a. Note: For every 360º rotation, the sensor will take 32 measurements.
10. One set of measurement has been taken, now take one step forward and press the pushbutton with a connection to PN1 once to untangle the wires.
    a. Note: You may also manually untangle the wires by disengaging the sensor mount from the stepper motor, however, ensure that the sensor is in the same position as it was when disengaged.
11. Complete steps 8 to 10 until you have satisfied the number of sets.
12. Execute *data_3D_visualizer.py* module and a open3d interface showing the 3D reconstruction of the data points collected should appear on screen.

Note: For every new scan/use of the program, delete *distance_data.txt* from your local repository as the program does *not* overwrite when a new execution of *data_collection.py* is ran.

c. Expected Output

Figures 6 and 7 illustrate the 3D reconstructions of Hallway A180 as seen in Figures 4 and 5. Note the discrepancies are due to the glass in the hallway and as such, the light bounces off the glass thus affecting the final measurement.

 *Figure 6: Side view of hallway A180*	 *Figure 7: Frontal view of hallway A180*

5. Limitation

While the MSP432E401Y microcontroller and the other peripherals used in the system are very capable systems, they exhibit some limitations namely in the microcontroller floating point capability, the quantization errors, maximum standard serial communication rate, speed used between the microcontroller and the ToF modules and its associated communication protocols and system bottle necks regarding speed from the ToF sensor and stepper motor.

Microcontroller Floating Point Capability

The MSP432E401Y is a 32-bit system as it's based on the ARM Cortex-M4F processor, which is a 32-bit processor with a hardware floating-point unit (FPU). As such, when it performs 64-bit calculations, it is far more computationally intensive than 32-bit calculations, and therefore may require more processing time and consume more power. Additionally, the memory requirements for storing 64-bit data can be higher than for 32-bit data, which may be a limitation for some applications with limited memory resources. The microcontroller also has limitations with trigonometric calculations as it does not support for the implementation of large trigonometric functions. Thus, for this device, these calculations are implemented through the python libraries when the 3d model is being reconstructed.

Quantization Errors

The ToF sensor emits a photon to measure the time taken for the pulse to travel to a target and back and distance can be calculated as such. While there is a high degree of accuracy, quantization errors regarding the resolution of the sensor's ADC. The quantization error is calculated as following:

$$ToF\ Sensor\ Quantization\ Error = \frac{Maximum\ Range\ Value}{2^n}$$

Where the maximum range value is 4000 mm as pre-determined by the sensor manufacturer, and n is 16 as the ToF module uses a timer with a 16-bit resolution.

$$Quantization\ Error = \frac{4000\ mm}{2^{16}} = 6.10 \times 10^{-2} mm$$

Maximum Serial Communication Rate and Associated Speeds

When the ToF sensor communicates with the microcontroller, it implements a I2C communication protocol which transmit data at 400,000 bits/second. Furthermore, data transfer between the microcontroller and the PC is via the UART protocol at a 115200-bps baud rate.

Bottle Necks

The main limitation regarding the data acquisition speed of the device is the rotational speed of the stepper motor. Beyond the pre-determined bus speed of 20MHz, for each measurement taken, there is a required 0.01 second pause at each 11.25º increment such that the ToF sensor can take a measurement unaffected by other factors. Furthermore, the ToF sensor also has a bottle neck in its maximum frequency as it can only perform distance measurements up to 50MHz. Not only does this affect the data acquisition itself, but also the speed of the microcontroller receiving the data from the sensor. A sample test for the speed limitation can be performed by comparing the time it takes for the microcontroller to receive and process data from the ToF sensor in different test conditions. This may be through altering bus speed frequency, number of measurements taken or altering ToF frequency.

6. Circuit Schematic

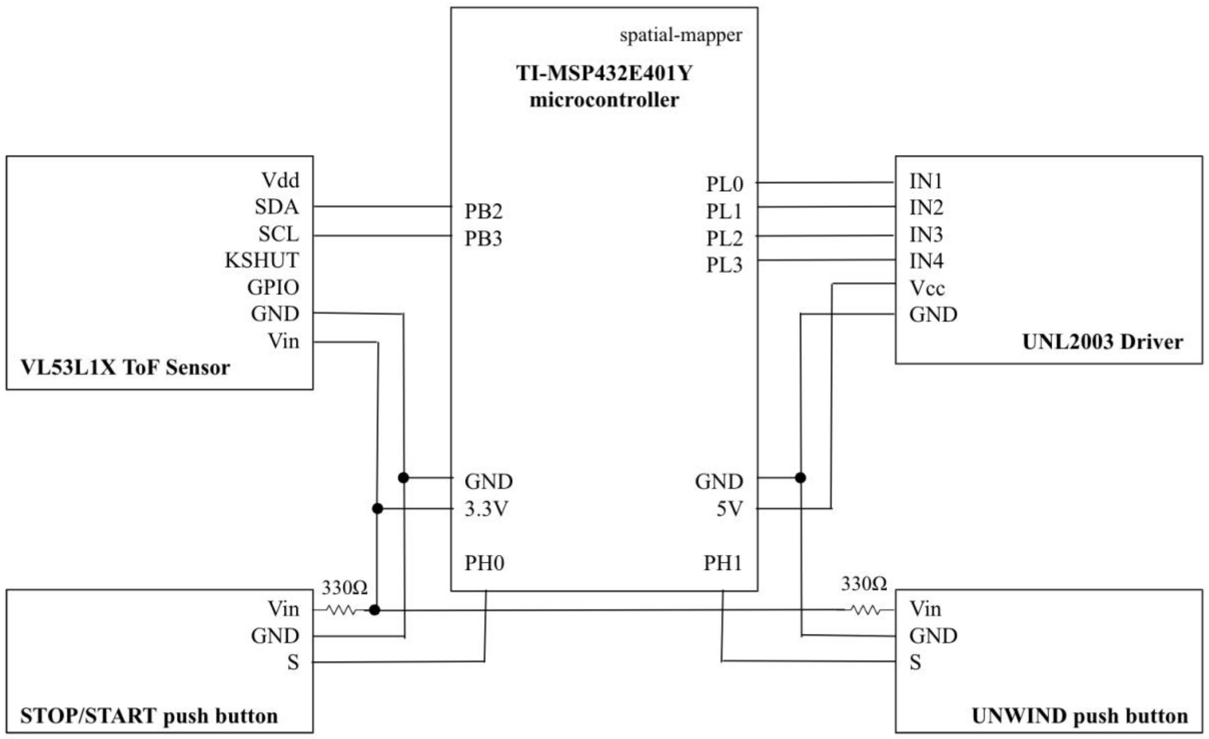


Figure 8: Device Circuit Schematic

7. Programming Logic Flowchart(s)

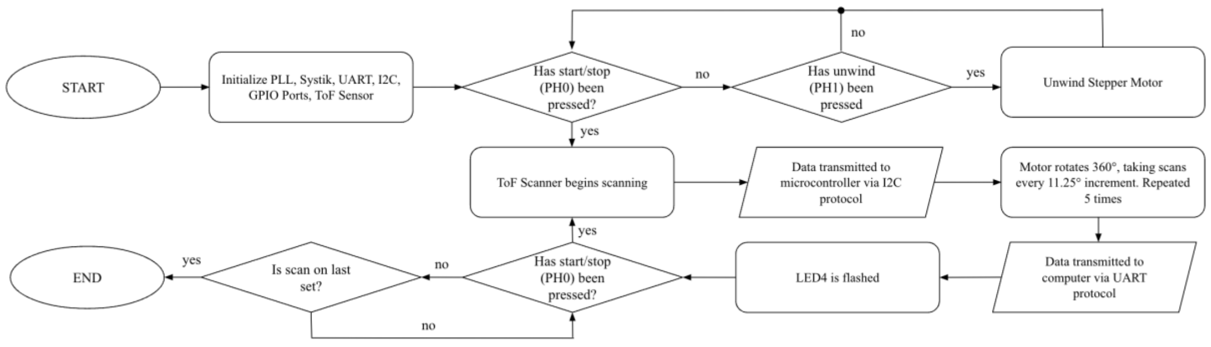

START
Initialize PLL, Systik, UART, I2C, GPIO Ports, ToF Sensor
Has start/stop (PH0) been pressed?
no
Has unwind (PH1) been pressed?
no
yes
Unwind Stepper Motor
yes
ToF Scanner begins scanning
Data transmitted to microcontroller via I2C protocol
Motor rotates 360°, taking scans every 11.25° increment. Repeated 5 times
Data transmitted to computer via UART protocol
LED4 is flashed
Has start/stop (PH0) been pressed?
yes
no
Is scan on last set?
yes
END
no

Figure 9: Keil program logic flowchart

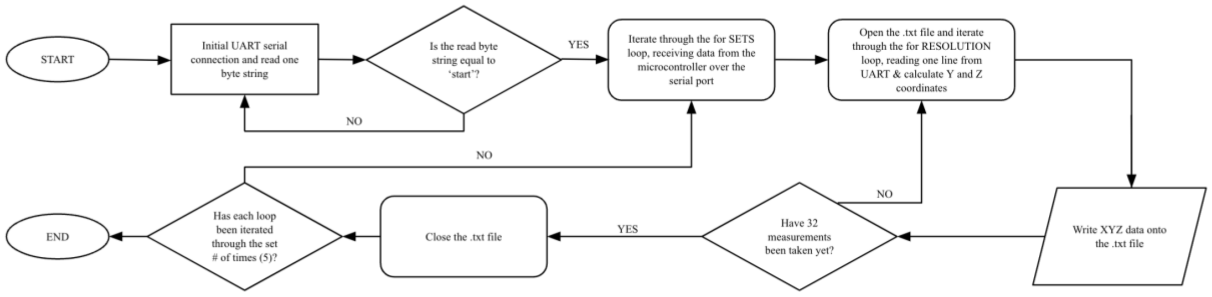

START
Initial UART serial connection and read one byte string
Is the read byte string equal to 'start'?
YES
NO
Iterate through the for SETS loop, receiving data from the microcontroller over the serial port
Open the .txt file and iterate through the for RESOLUTION loop, reading one line from UART & calculate Y and Z coordinates
Write XYZ data onto the .txt file
Have 32 measurements been taken yet?
NO
YES
Close the .txt file
Has each loop been iterated through the set # of times (5)?
NO
END

Figure 10: data_collection.py logic flowchart

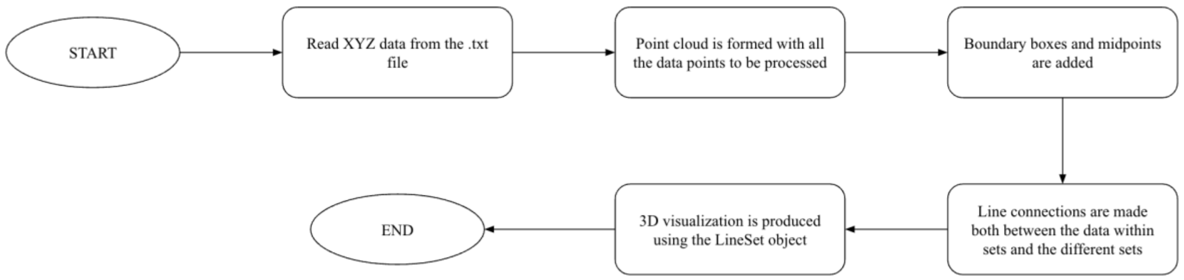

START
Read XYZ data from the .txt file
Point cloud is formed with all the data points to be processed
Boundary boxes and midpoints are added
Line connections are made both between the data within sets and the different sets
3D visualization is produced using the LineSet object
END

Figure 11: data_3D_visualiser.py logic flowchart