1. Device Overview

1a. Features

The 3D Spatial Mapper is a culmination of various different tools and features that work together to achieve the final product. It is a 360 degrees singular plane distance measurement device that communicates data via UART transmission through a USB to subsequently visualize a 3D mapping.

- MSP432E401Y Microcontroller (Texas Instruments)
  - 40MHz Bus Speed (Default Bus Speed: 120MHz)
  - Arm Cortex M4F Processor.
  - 1024 KB flash memory.
  - 256 KB SRAM.
  - 8 UART ports (16Mbps high speed baud rate)

- VL53L1X Time-of-Flight (ToF) sensor
  - 4m maximum measurement distance.
  - 2.6V – 3.5V operating voltage.

- UNL2003 Stepper Motor Controller
  - Full step rotation with 512 steps per 360 degrees.
  - 5V – 12V operating voltage.

- Data Visualization
  - Using open3D Python module.
  - Best supported for Python versions 3.6 – 3.11.

1b. General Description

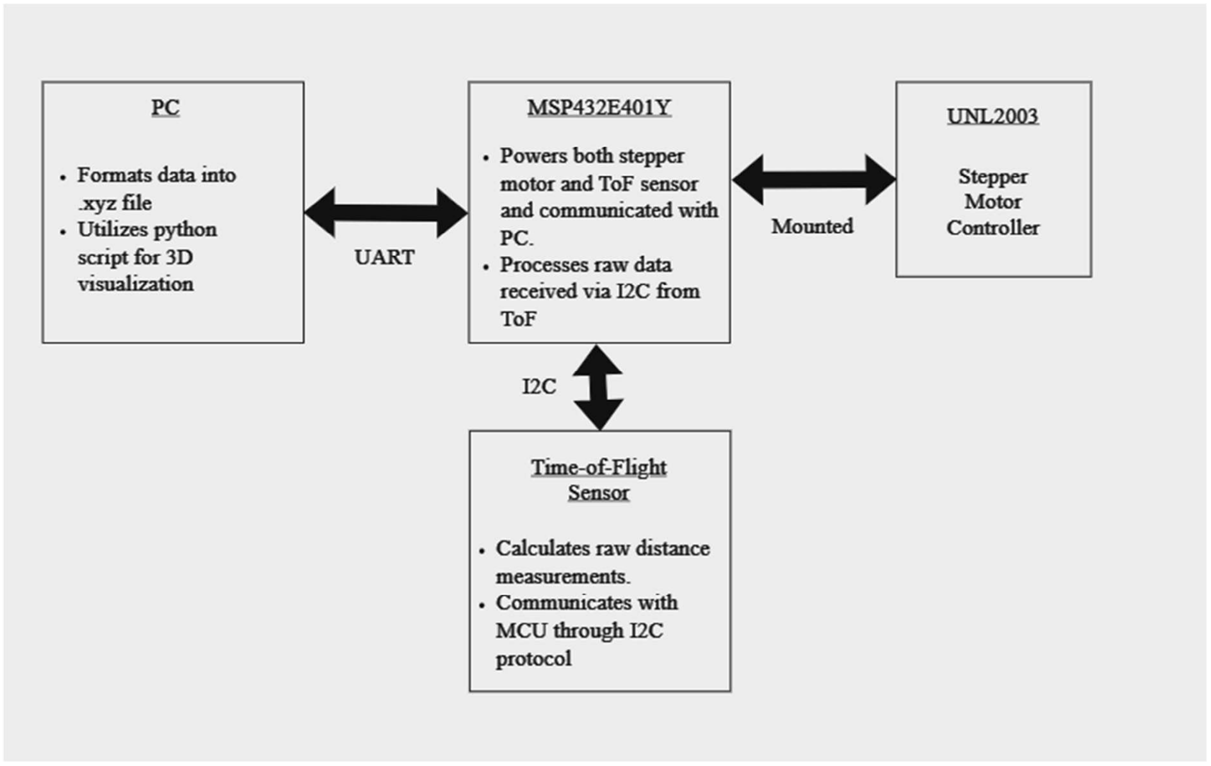
The 3D Spatial Mapper is an innovative, compact system developed for accurate spatial measurement and mapping. Utilizing the VL53L1X Time of Flight (ToF) sensor, which has a maximum distance range of 4m, this device is specifically designed to acquire information about spatial dimensions within a vertical plane (y-z axis), while manual adjustments are made along the x-axis. By emitting infrared light which bounces off the environment, the ToF is capable of gathering accurate distance measurements using the time taken for the pulses to bounce back. The sensor is mounted on a UNL2003 Stepper Motor Controller which enables it to rotate 360 degrees, capturing comprehensive distance measurement data of indoor environments.

The core of the system's data handling, processing and power generation is the MSP432E401Y microcontroller, which operates at a bus speed of 40MHz, ensuring swift and efficient data management. It interfaces directly with the VL53L1X sensor via I^2C communication protocol, which facilitates the collection of distance measurements to be processed into a 3D visualization.

After processing the data, the microcontroller serially transmits it to an open3D Python model using UART communication protocol for visualization at a baud rate of 115.2 Kbps. This allows for detailed graphical representation and analysis of the spatial data.

This device is engineered as a cost effective and compact alternative to conventional commercial LiDAR equipment, ideal for applications such as indoor navigation, autonomous robotics, and layout mapping.

1c. Block Diagram


PC
• Formats data into .xyz file
• Utilizes python script for 3D visualization
UART
MSP432E401Y
• Powers both stepper motor and ToF sensor and communicated with PC.
• Processes raw data received via I2C from ToF
Mounted
UNL2003
Stepper Motor Controller
I2C
Time-of-Flight Sensor
• Calculates raw distance measurements.
• Communicates with MCU through I2C protocol

2. Device Characteristics

Feature	Detail / Microcontroller Pin
Stepper Motor Driving Pins	PH0 - PH3
SCL (VL53L1X I^2C connection)	PB2
SDA (VL53L1X I^2C connection)	PB3
Onboard Push Button	PJ1
Measurement Status (LED1)	PN0
Additional Status (LED2)	PN1
Communication Port	COM4
Bus Speed	40 MHz
Communication Speed	115.2 kbps

3. Project Description

3a. Distance Measurements

The 3D LiDAR Spatial Mapper is engineered to perform precise and dynamic spatial measurements using the VL53L1X Time of Flight (ToF) sensor. This sensor uses a laser to emit light and measure the time it takes for the light to return after reflecting off surfaces. The distance to objects is then calculated based on the time delay, enabling accurate positioning within a 3D space. The system is specifically designed for environments with minimal light interference, as ambient lighting can affect the accuracy of the ToF sensor.

The device captures spatial data through a meticulous process where the ToF sensor is mounted on a stepper motor. This motor rotates the sensor incrementally through a full 360 degrees, with each step's angle predefined by the total number of scans desired (e.g., 11.25 degrees per step for 32 scans). This setup ensures comprehensive spatial data coverage within a vertical plane (y-z).

As the stepper motor rotates and the sensor collects distance measurements, these are immediately transmitted to the MSP432E401Y microcontroller via an I2C interface bus line. The microcontroller operates at a bus speed of 40MHz, efficiently managing the data throughout required for real time processing.

Concurrently, the ‘data_visualization.py’ Python script actively runs, awaiting data transmission of distance measurements from the microcontroller. Each scan's distance data is serially transmitted in real time to the script through UART communications at 115.2 kbps. This script employs the PySerial library to establish and manage the serial connection, ensuring seamless data flow from the microcontroller to the PC.

Upon receiving the distance data, the Python script calculates the y and z coordinates using the trigonometric relationships:

$$\text{Z Coordinate} = \text{Distance} * \cos(\text{angle})$$

$$\text{Y Coordinate} = \text{Distance} * \sin(\text{angle})$$

Note: The x coordinate on the other hand is manually incremented by the user, for this device specifically, 30cm is a suitable displacement amount.

The ‘angle’ variable in these equations corresponds to the current position of the stepper motor with respect to the 360-degree axis, which adjusts incrementally per scan inside the python script. This method allows the script to dynamically calculate and compile the 3D spatial coordinates in real time as the sensor progresses through each scanning angle.

Finally, the resulting coordinates are formatted into a ‘.xyz’ file, which is a standard format for storing three dimensional data. This file is then used for various applications, including 3D modeling and visualization in software platforms capable of rendering 3D environments as discussed in Section 3b.

3b. Data Visualization

The visualization process of the 3D Spatial Mapper is carried out using a Lenovo Thinkbook 14 G4 IAP equipped with a 12th Gen Intel Core i7-1255U processor and 16GB RAM. This setup ensures the system has sufficient processing power to handle complex 3D data sets and render them effectively in real time.

The core of the visualization process utilizes the Open3D and numpy libraries, which are deployed for handling and rendering point cloud data. The module reads the spatial coordinates from a '.xyz' file, which stores the three dimensional data. The data in this file represents the spatial points in the environment scanned by the ToF sensor, with each point corresponding to a specific location in space as determined by the sensor's readings.

Once the point cloud is formed, Open3D transforms this raw data into a visually comprehensible format, enabling detailed 3D visualization. This point cloud provides a holistic view of the scanned area, allowing users to rotate, zoom, and traverse through the 3D space virtually.

Additionally, the visualization process includes tools for further analysis and manipulation of the point cloud data. Users can apply various filters, perform segmentations, and conduct measurements directly within the visualization environment. These features make the 3D Spatial Mapper not only a tool for simple visualization but also a comprehensive platform for in depth analysis of spatial environments

4. Application Example

4a. Environment Setup

Key software debugging and execution modules are required for the functionality of this device:

- Download XDS Emulation Software for communication via XDS port as well as microcontroller debugging.
- Download and install any version of Python newer than 3.6 and ensure it is added to path.
- Install the necessary Python libraries required for the script. This includes 'pyserial', 'numpy' and 'open3D'. All of which can be installed using the 'pip install' command inside the user command prompt window run as administrator. For example, installing the numpy library on would execute 'pip install numpy' inside the command prompt window.

4b. Equipment Setup Guide

Once the key software has been installed, the environment is ready for calibration.

Assemble the microcontroller, stepper motor and ToF configuration using the Circuit Schematic as reference while having a USB connection from the microcontroller to the PC for UART transmissions.

- Open and run 'data_visualization.py' inside a python IDE such as IDLE. Default settings are as follows: COM Port: 4, x-displacement of 30cm up to 6600cm (22 scans total), 32 scans with a rotation angle of 11.25 degrees.

  - Before running, the user may adjust the default parameters of the spatial mapper depending on their desire.
  - The communication port (COM) and baud rates can be changed in line 18 depending on the user's configuration.
  - The rotation angle which depends on the number of scans can both be changed by modifying line 42 and changing the for loop in line 48. However, the C file in the Keil IDE must also be modified to change the number of scans, more details below.
  - Lastly, the x-axis displacement can also be changed in the for-loop parameters on line 47.

- Once the python script has ran, load and flash the microcontroller with the Keil C program file that contains the instructions for the device to operate. Once the microcontroller has been flashed, click the reset button to load it onto the microcontroller. The python module will then promptly display whether or not device configuration was successful.

  - The user may choose to modify the number of scans by changing line 198 specifically inside the 'spin' function. For example, the 'if' statement conditional is by default set to "*if(i%16==0)*" which makes the device take 32 scans. This value can be changed to 'mod 32', for example, if the user desires to have the device take 16 scans instead.

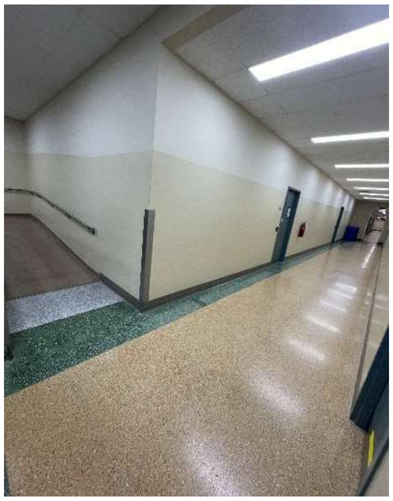
- Now that the python script is running, and the microcontroller has been flashed and loaded. Data acquisition can be conducted by clicking the GPIO onboard push button of the microcontroller, PJ1. Clicking this button will begin the data acquisition process until a full 360 degree rotation has been made.

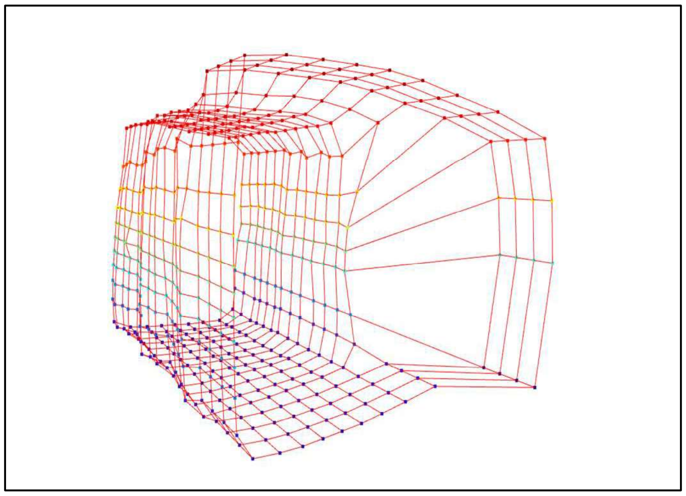
- Once one 360-degree rotation has been completed, the python script will await another set of data acquisition measurements (depending on what the user set as the number of x displacements).

  - For example, with the default settings, for each 360-degree scan, the device is to be moved along the x-axis up 30cm. This process will be repeated until the specified step range in line 47 of the python script (which is 6600 by default meaning 22 data acquisition steps will be taken). Again, these step values may be changed as the user desires depending on the x-axis of the environment they would like to gather a mapping of (e.g. 6600cm if a hallway is 6.6m long)

- Lastly, since the visualization protocol via open3D and numpy are integrated into the python transmission script, a point cloud 3D visualization of the environment will appear once data collection has been completed.
    - This will be evidently seen in the python environment where once data has been collected, the communication port will close, and a new open3D environment window will open displaying the results.

Expected Output:





5. Limitations

1. While the MSP432401Y microcontroller is equipped with a 32-bit Floating Point Unit (FPU), enabling it to perform single precision floating point arithmetic with operations such as addition, subtraction, multiplication, division, and square root, the trigonometric calculations are delegated to the Python script. This is due to the ease of implementing such functions in Python. The FPU provides the capability to handle these operations, but practical application prefers offloading complex math functions to more capable processors, thereby streamlining the embedded system’s performance for its primary task which is to collect raw distance data.

2. Calculating maximum quantization error which is the same as resolution:

$$\text{Resolution} = \frac{\text{VFS}}{2^n}$$

Where VFS = 4000mm and n = 16 since the ToF ADC stores data in 16-bit registers

$$\text{ToF quantization error} = \frac{4000\text{ mm}}{16} = 6.10 \times 10^{-2}\text{ mm}$$

3. The maximum standard serial communication rate that can be implemented with a PC is based on the UART transmission hardware of the MSP432401Y. The value for it is the same that is implemented in this device which is 115.2 kbps. This value was confirmed using RealTerm as a debugging tool.

4. The communication method used between the microcontroller and the ToF module was I^2C communication protocol which allowed serial communication between the two modules. Moreover, the two modules were configured on a common system clock (SCL) of 100 kbps.

5. The prime limitation is the ToF sensor. The limitation stems from the nature of how the ToF sensor operates. Specifically, it measures distance by emitting a light signal and calculating the time it takes for the reflected signal to return. This measurement process is sequential and bounded by time, as the sensor can only process one measurement at a time and must wait for the signal to return before emitting the next signal.

   Additionally, the maximum rate at which the ToF sensor can emit and receive light pulses is fixed by its design specifications. The physical constraints of the speed of light, the need for accurate time measurement of the light's time of flight, and the sensor's signal processing capabilities all contribute to the fixed upper limit on the measurement rate.

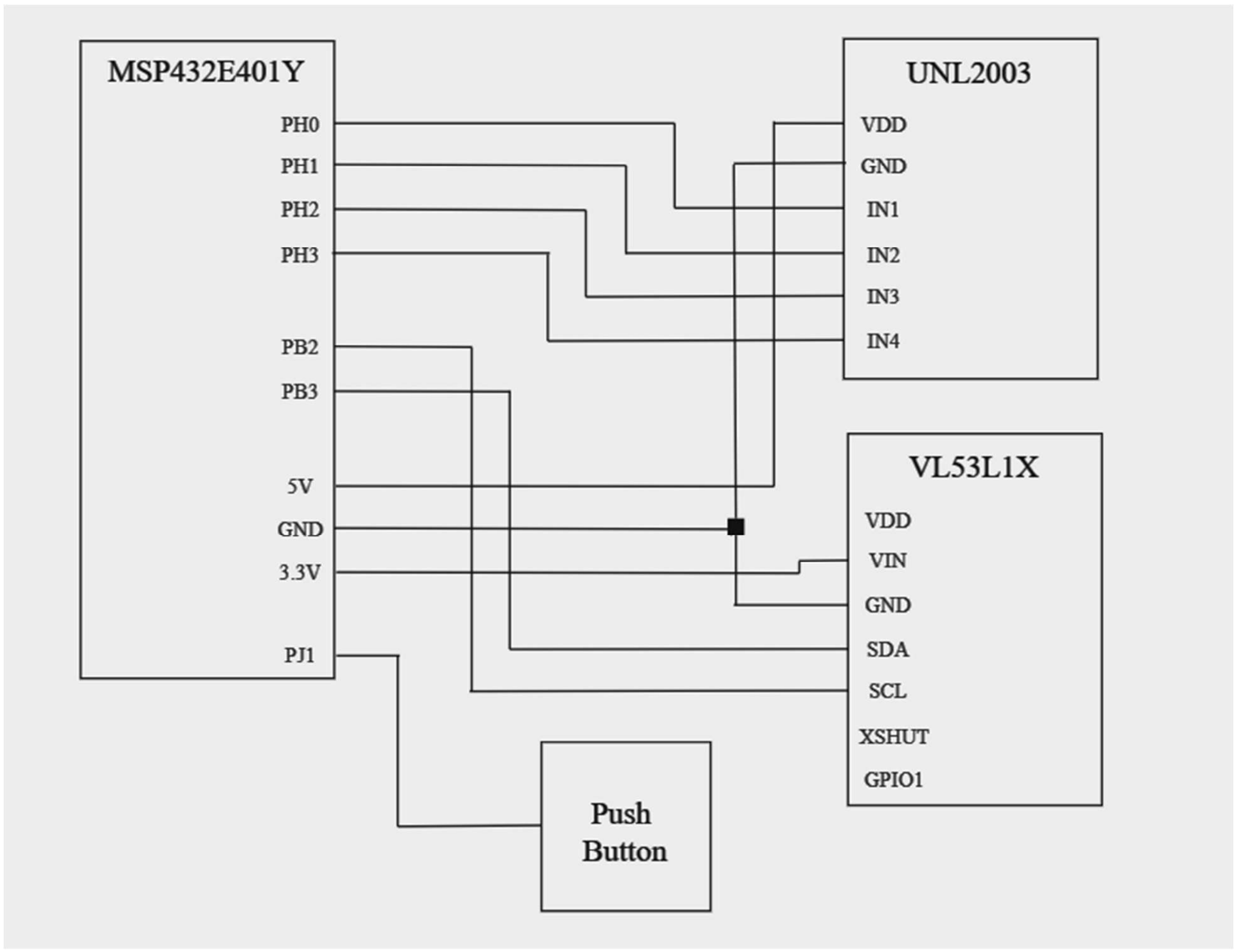
   This was tested by analyzing how the system worked when ‘SysTick_Wait’ was removed in the data collection function that uses the ToF core API functions which resulted in data acquisition often failing to give real distance data.

6. The assigned bus speed for this device is 40MHz. This bus speed was calculated and implemented by configuring both the Phase Lock Loop (PLL) PSYDIV 10-bit register as well as the 'SysTick'. More specifically, the value to configure PSYDIV to can be calculated using the formula:

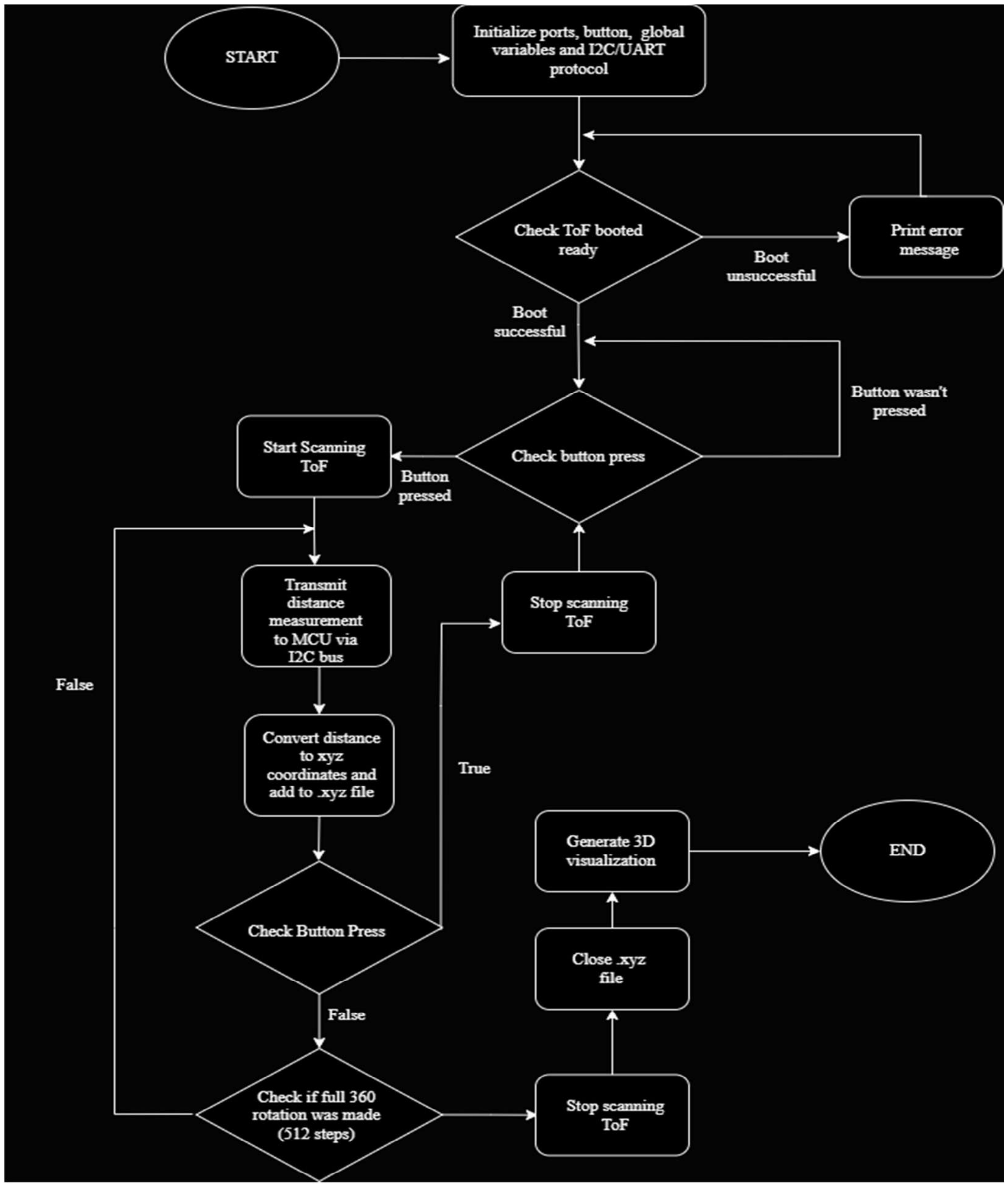
$$\frac{480\ MHz}{(PSYDIV + 1)} = Desired\ Bus\ Speed\ (MHz)$$

Since the desired bus speed is 40MHz for this implementation, PSYDIV can be solved to be equal to 11. This value is then configured in the PLL header file and the 40MHz value is plugged into the 'SysTick' delay methods.

6. Circuit Schematic



7. Programming Logic Flowchart


START
Initialize ports, button, global variables and I2C/UART protocol
Check ToF booted ready
Boot unsuccessful
Print error message
Boot successful
Button wasn't pressed
Check button press
Start Scanning ToF
Button pressed
Transmit distance measurement to MCU via I2C bus
Stop scanning ToF
False
Convert distance to xyz coordinates and add to .xyz file
True
Generate 3D visualization
END
Check Button Press
Close .xyz file
False
Check if full 360 rotation was made (512 steps)
Stop scanning ToF