Hamza Electronics
Navigate with precision

Spatial Mapping System using ToF v0.1

Core Device Components

- Microcontroller Unit (ARM® Cortex®-M4F TI MSP432E401Y SimpleLink™)
- Stepper Motor (28BYJ-48 Velleman® 5 VDC Stepper Motor with ULN2003 Driver Board)
- Time-of-Flight Sensor (ST FlightSense™ VL53L1X)
- Additional Push Button
- Removable Sensor Mount (Hamza Electronics™)
- Fiberboard Case (Hamza Electronics™)

Spatial Mapping System Using Time-of-Flight

Hamza Siddiqui - 400407170 - siddih38

April 17, 2023

As a future member of the engineering profession, the student is responsible for performing the required work in an honest manner, without plagiarism and cheating. Submitting this work with my name and student number is a statement and understanding that this work is my own and adheres to the Academic Integrity Policy of McMaster University and the Code of Conduct of the Professional Engineers of Ontario.

1 Device Overview

1.1 Features

- ARM Cortex-M4F TI MSP432E401Y Microcontroller ($53.47)
  - Bus frequency of 20MHz (configured via phase-locked loop; default core frequency of 120MHz)
  - Four onboard LEDs - D3 flash functionality on each measurement
  - Two onboard buttons (PJ1 for start/stop of motor rotation) and one reset button
  - 1MB of flash, 256kB of SRAM
  - Communication interfaces including USB-OTG, CAN, Quad-SPI (QSSI), $\mathrm{I^2C}$, SPI, and UART
  - Two 12-Bit SAR-Based ADC Modules (2 Msps)
- MOT-28BYJ-48 Stepper Motor with ULN2003 driver ($6.99)
  - 5-12 VDC Operating Range
  - 64 steps/revolution
  - Frequency of 100Hz
  - 4 phases with 4 step-indicator LEDs
- VL53L1X Time-of-Flight Sensor ($25.32)
  - Communication via $\mathrm{I^2C}$ (upto 400kHz)
  - 4 m ranging and ranging frequency of 50Hz
  - 2.6-5.5V supply range, 2.8V operating range
  - Emitter: 940 nm invisible laser (Class1)
- Additional Push Button ($0.60)
  - Start/stop functionality for data acquisition
- Additional System Features
  - UART serial communication between device and PC
  - $\mathrm{I^2C}$ serial communication between device and Time-of-Flight sensor
  - Baud rate of 115200
  - C-language programming for device instructions, Python programming for visualization (via Pyserial and Open3d)

1.2 General Description

To begin data acquisition, the additional push button is pressed to enable data processing, then the onboard button is pressed to begin taking measurements by spinning the motor and interfacing with the Time-of-Flight (ToF) sensor. The SMSUTOF HS 2023 utilizes a transducer which provides a digital output. Spatial distance measurements are acquired via the ToF sensor using $\mathrm{I^2C}$, while displacement is fixed at 30 cm. At each angle a measurement is taken, the D3 LED flashes. Together these measurements allow the device to measure a 3-D area. These signals are converted to electric signals, then conditioned into digital form and are finally converted to discrete digital values during the ADC process. The system communicates this to the user PC via UART asynchronous serial communication. A Python program is run on the PC and interfaces via their UART COM port and a baud rate of 115200, which then transfers measurement data to the PC and allows it to be stored in a .xyz file. This collected data is then used to generate a 3-D plot of the area that has been scanned via a Python program using open3d. The points for this plot are generated using trigonometric properties based on the angle and y,z coordinates generated, along with the fixed x-coordinate increment. If at anytime the button is pressed again during the scanning process, the data acquisition will be stopped.

1.3 Block Diagram (Data Flow Graph)

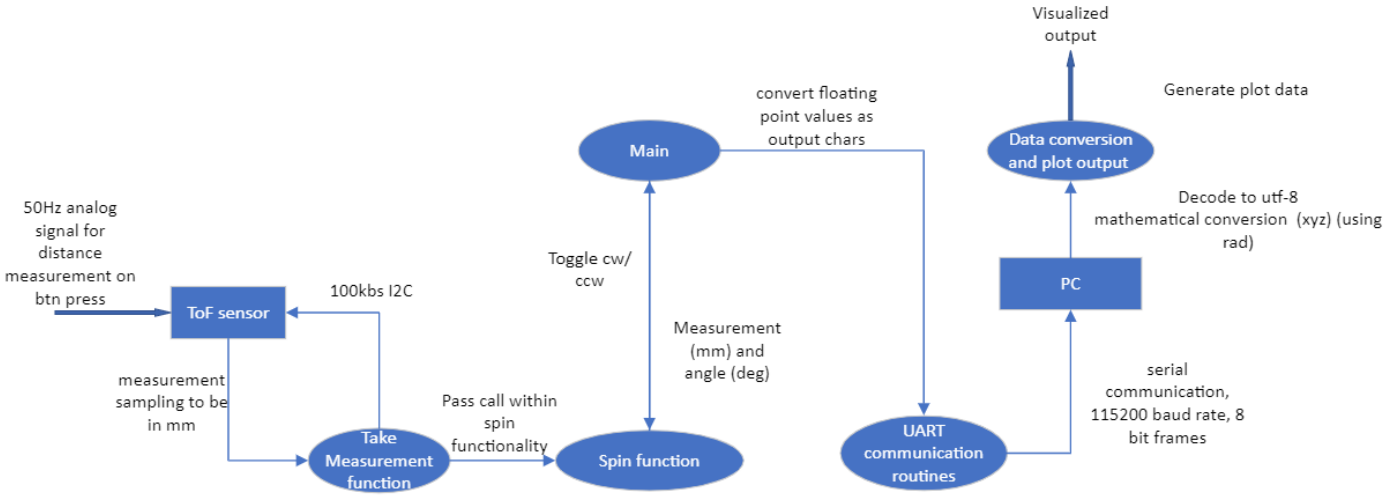


Figure 1: Flow graph

2 Device Characteristics Table

Characteristic	Description
Bus Speed	20MHz
Serial Port	COM4
Communication Speed	115200 bps
Measurement Status	D3

Table 1: Device Characteristics

Stepper Motor Pin Assignments

ULN2003 Driver Board	MCU
+ (5V)	5V
- (5V)	GND
IN1	PH0
IN2	PH1
IN3	PH2
IN4	PH3

Table 2: Stepper Motor Pinout

Time-of-Flight Sensor Pin Assignments

VL53L1X	MCU
VDD	-
VIN	VCC (3.3V or 5V)
GND	GND
SDA	PB3 (I2C0 SDA)
SCL	PB2 (I2C0 SCL)
XSHUT	-
GPIO1	-

Table 3: Time-of-Flight Sensor Pinout

3 Detailed Description

3.1 Distance Measurement

Acquisition:
The VL53L1X sensor utilizes Light Detection and Ranging (LIDAR) technology to accurately measure distances. It emits a pulse of light with a wavelength of 940 nm, which is then reflected back to the sensor by an object (maximum 4 m away). The sensor measures the time it takes for the emitted light pulse to reach the object and be reflected back to the detector. By utilizing the known speed of light and the measured time, the VL53L1X calculates the distance to the object with high accuracy as follows:

$$Measured\ Distance = \frac{Photon\ Travel\ Time}{2} \times Speed\ of\ Light$$

The sensor is mounted to a motor that completes 360 degree rotations (512 steps). At every 64 steps, or 45 degrees, a user LED is flashed to indicate a measurement (in mm) has been taken by the sensor. This is achieved through interfacing with the sensor and micro controller via $\mathrm{I^2C}$ serial communication. In total, this allows for 8 measurements to be taken in a single rotation, and the program can be modified to measure at smaller number of steps to take more measurements for more accuracy at the cost of time (use-case dependent configuration).
Data Processing:
Data is processed and converted via mathematical computation. When given the motor's angle θ, the program increments by the measurement angle rotation each time and computes the coordinates using trigonometric properties. This is computed as $angle = steps/(TOTALSTEPS) \times 2\pi$ where steps is incremented by 45 degrees and TOTAL STEPS is set to 512 - these reset each time a rotation is complete i.e. when 512 steps have completed. For a given distance measurement r, we obtain that $y = r\cos\theta$ and $z = r\sin\theta$ for the vertical plane. X coordinate is computed through configuring a physical step increment taken for each successive measurement rotation in the program and incrementing the coordinate for the next spatial scan based on this value (350mm default configuration). This is the displacement for each scan. Together these coordinates can be used for plotting and mapping.
Example Calculation:

Suppose a measurement r is measured as 3 m at an angle of 45 degrees. Then,

$$\begin{aligned} y &= r\cos 45 \\ &= 3\cos 45 \\ &= 2.12132034356 \text{ m} \end{aligned}$$

and

$$\begin{aligned} z &= r\sin 45 \\ &= 3\sin 45 \\ &= 2.12132034356 \text{ m} \end{aligned}$$

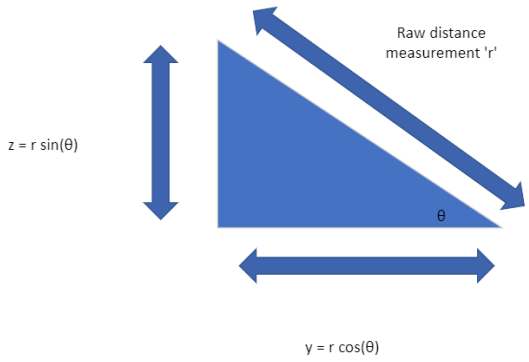


Figure 2: Trigonometric Conversion: Solving for y and z using the sine and cosine functions.

3.2 Visualization

The visualization is achieved through Python using the NumPy, open3d, and PySerial libraries. Data is communicated using UART and a specified PORT which is accessed via PySerial. The measurement data from the micro controller is then communicated to the PC and is processed as points following the previously outlined conversion method. The points are then written to a .xyz file in the format of "x y z" per line, and is outputted on the console in the same manner for the user to view measurements printed in real time for each measurement. Next, using open3d, the points are viewed numerically by conversion using NumPy.
Here is an example of how to read a point cloud using the Open3D library:

```
pcd = o3d.io.read_point_cloud("pointdata.xyz", format="xyz")
```

In this code snippet, the variable `pcd` stores the point cloud data read from the file `pointdata.xyz`. The `format` parameter specifies the format of the file, which in this case is XYZ format. These are then visualized using the following function call:

```
o3d.visualization.draw_geometries([pcd])
```

Next, each vertex from the data is appended and the coordinates to connect lines in each yz slice is defined. The lines are then mapped to the 3D coordinate vertices as a line set and are then outputted to the GUI for visualization with the following function calls:

```
line_set = o3d.geometry.LineSet(
    points=o3d.utility.Vector3dVector(np.asarray(pcd.points)),
    lines=o3d.utility.Vector2iVector(lines)
)

o3d.visualization.draw_geometries([line_set])
```

In this code snippet, the `LineSet` class is used to create a line set from a set of points and lines. The `points` argument is a vector of 3D points, and the `lines` argument is a vector of pairs of point indices that define the line segments. The resulting line set is then visualized using the `draw_geometries` function from the Open3D visualization module.

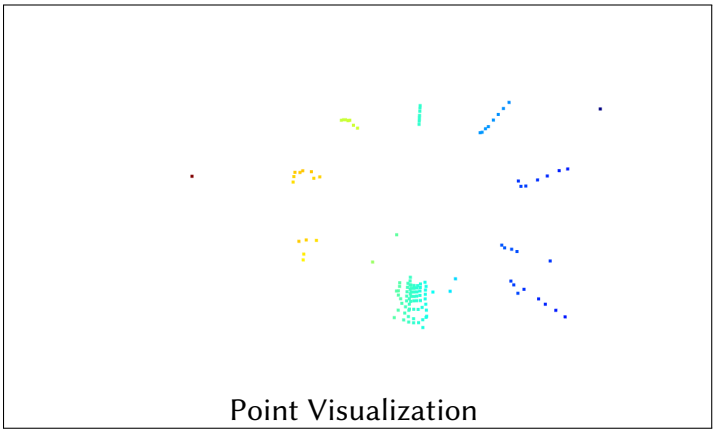
4 Application Example with Expected Output

A spatial scan can be taken of a room with a set number of displacement increments. An example room with an expected output can be seen below:

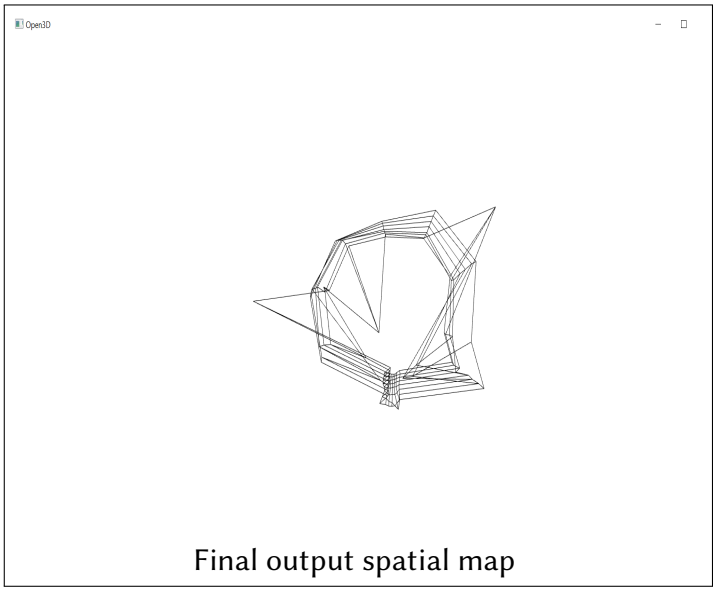
Hallway POV 1

Hallway POV 2

Figure 3: Hallway scanned



Point Visualization



Final output spatial map

Figure 4: Expected output of scan

5 User's Guide

Setting up:
Build the circuit based on the schematic provided in section 7. Ensure the ToF is mounted to the stepper motor. The Keil project should be flashed on the micro controller by default, so pressing the reset button is sufficient to load the program. If this is not the case: Open the Keil Project src from the provided code and $Translate \rightarrow build \rightarrow load$.

Software Installation:

1. Download Python 3.9, which can be found at https://www.python.org/downloads/ under specific release downloads.
2. Install NumPy, open3d, and Pyserial by opening terminal and typing:

```
pip install numpy
pip install open3d
pip install pyserial
```

   Refer to their documentation for any issues:

   - NumPy documentation: https://numpy.org/doc/1.21/
   - Open3D documentation: http://www.open3d.org/docs/release/
   - Pyserial documentation: https://pyserial.readthedocs.io/en/latest/pyserial.html

How to use the device after setting up:

1. Open `measurement_data.py` and modify the PORT variable to that of your device, found from UART port listed in device manager.
2. Run `measurement_data.py` by opening terminal, navigating to the directory of the file, and entering `python measurement_data.py`
3. Enter how many displacement steps the scan will use when prompted by the program.
4. Press the peripheral push button to enable measurements to set up the ToF. Pressing this again will toggle this back off.
5. Press the the onboard PJ1 button each time a measurement rotation has to be made. Pressing this again will end the rotation process.
6. For each number of physical steps taken, wait for the motor to complete one full rotation and return back to the home position. Once this is complete, move forward by the amount of x-displacement increment set (350 mm in the program but modifiable by user in the STEP_INCREMENT variable)
7. Press the button again for the next measurement.

8. The program will then print to you the measurements at each point taken as the program runs from the scan and also output this to a file in the same directory called `pointdata.xyz`. D3 LED will flash on the micro controller for verification purposes.

9. Run the `spatial_visualization.py` file by typing `python spatial_visualization.py` in terminal.

10. When prompted, enter the number of scans taken. The program will use this and then output the numerical value to the user in the console and output the visualization points on the screen in a GUI. Closing this screen will show a new screen with final spatial map in a GUI.

As explained earlier, the scan is of a yz plane with x being the displacement taken for each scan. y and z are seen as the vertical plane that the motor rotates and scans for, and x is the displacement after physically moving the motor by the step increment defined. Refer to section 3.1 for more detailed information with example calculations.

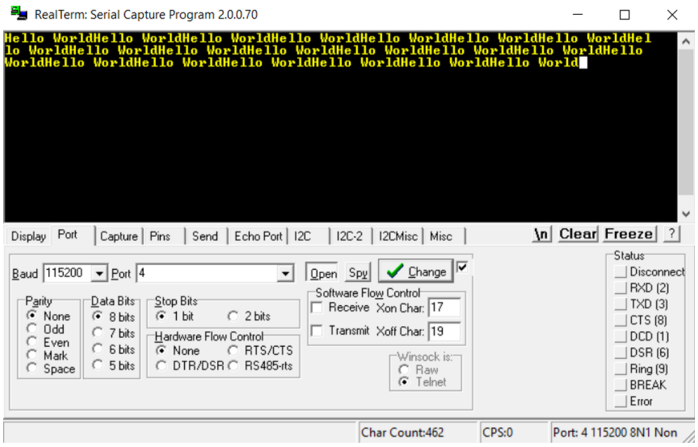
6 Limitations

- **FLoating Point Capability:** The micro controllers features a 120-MHz Arm® Cortex®-M4F Processor Core With Floating Point Unit (FPU). The FPU fully supports various single-precision operations such as add, subtract, multiply, divide, multiply-and-accumulate, and square root. It also provides conversions between fixed-point and floating-point data formats, and floating-point constant instructions. It has 32 dedicated 32-bit single-precision registers, which are also addressable as 16 double-word registers. With the limitation of bits, the computations communicated to the PC are given with truncation. The trigonometric computations done by Python's math library add to this as the infinite series are accurate to limits of double-precision floating-point arithmetic, 15 to 17 decimal digits. Together the accumulation of these truncations result in an imperfect calculation of points, however minimal. Furthermore the trignometric functions assume fixed angle inputs, which may be subject to error by upto a few degrees, hence the point is not computed at the exact angle the sensor was oriented at. This allows for the points to still be accurate to a certain reasonable precision without being entirely perfect.

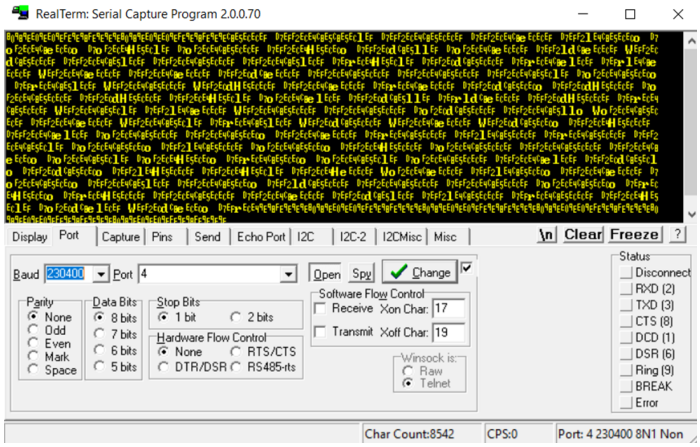
- **ToF Quantization Error:** The ToF module includes a quantization error in the ADC process. Quantization error is the difference between the analog signal and the closest available digital value at each sampling instant from the A/D converter. It is equivalent to resolution. It is computed using $m = 8$ from the ADC module bits and $V_{FS} = 3.3$ V.

$$\begin{aligned} resolution &= \frac{V_{FS}}{2^m} \\ &= \frac{3.3}{2^8} \\ &= 0.0129 \text{ V} \end{aligned}$$

- **Serial Communication:** The maximum serial communication rate of the PC can be found from device manager and is seen as 128000 bps as seen in the UART port. The rate is limited when using the micro controller as the rate is capped at a baud rate of 115200 bps. This can be verified by interfacing with UART at different baud rates until receiving an error. An example can be seen below:



RealTerm output at 115200 baud rate



RealTerm output above 115200 bps

Figure 5: Verification of maximum standard serial communication rate

The microcontroller and time-of-flight modules communicate via I2C serial communication. The I2C ports on the microcontroller are configured with a 100 kbps clock (in standard mode), as specified in the datasheet. In this setup, the microcontroller assumes the role of leader while the time-of-flight sensor acts as the follower and operates based on the microcontroller's clock speed.

- **Speed Limitations:** The speed of the stepper motor is accounted for with 512 steps and delays using `SystickWait(4000)` with a 20MHz bus speed. This gives a motor speed of 0.4096 s per rotation without measurement delays between. The ToF also had a speed limitation with a very quick default speed from the overall calculation formula mentioned, however the delays within the program outweigh this speed and become the limitation. The delay called was of 50ms. This was verified by toggling a GPIO output port before and after spinning, and before and after measuring, then visualizing this using oscilloscope probes from an Analog Discovery 2 board with a generated waveform, and visualizing the pulse generated for each.

7 Circuit Schematic

Below is the circuit schematic for the device:

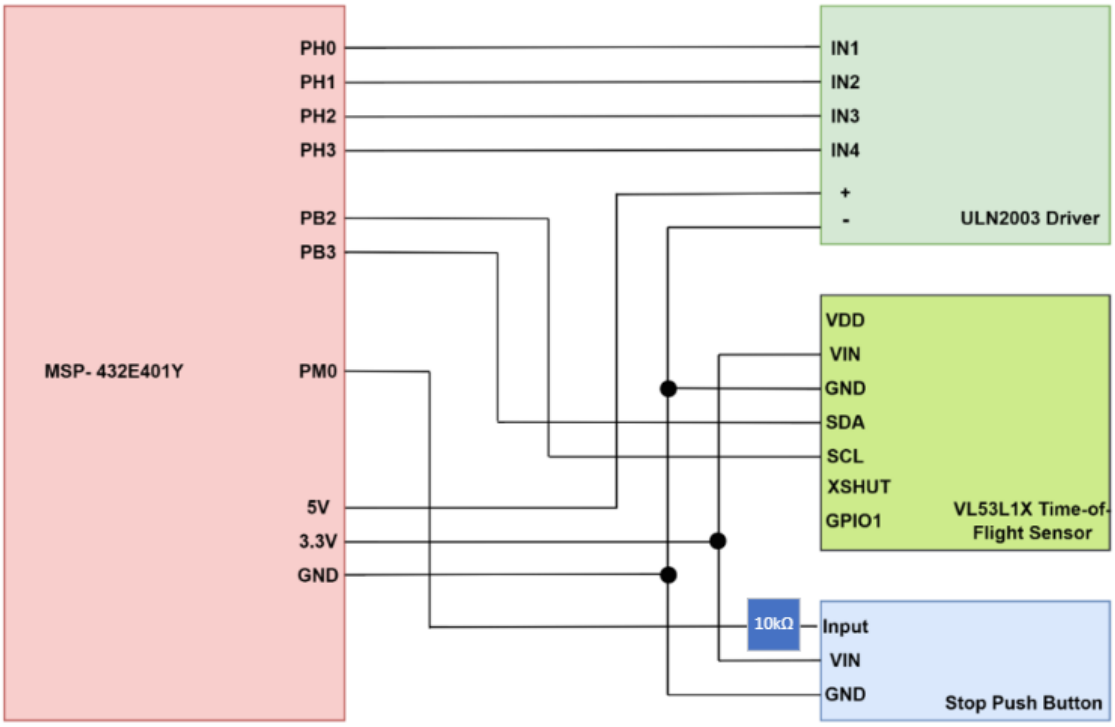


Figure 6: Circuit Schematic

8 Programming Logic Flowchart(s)

Below is the programming logic flowchart of the system:

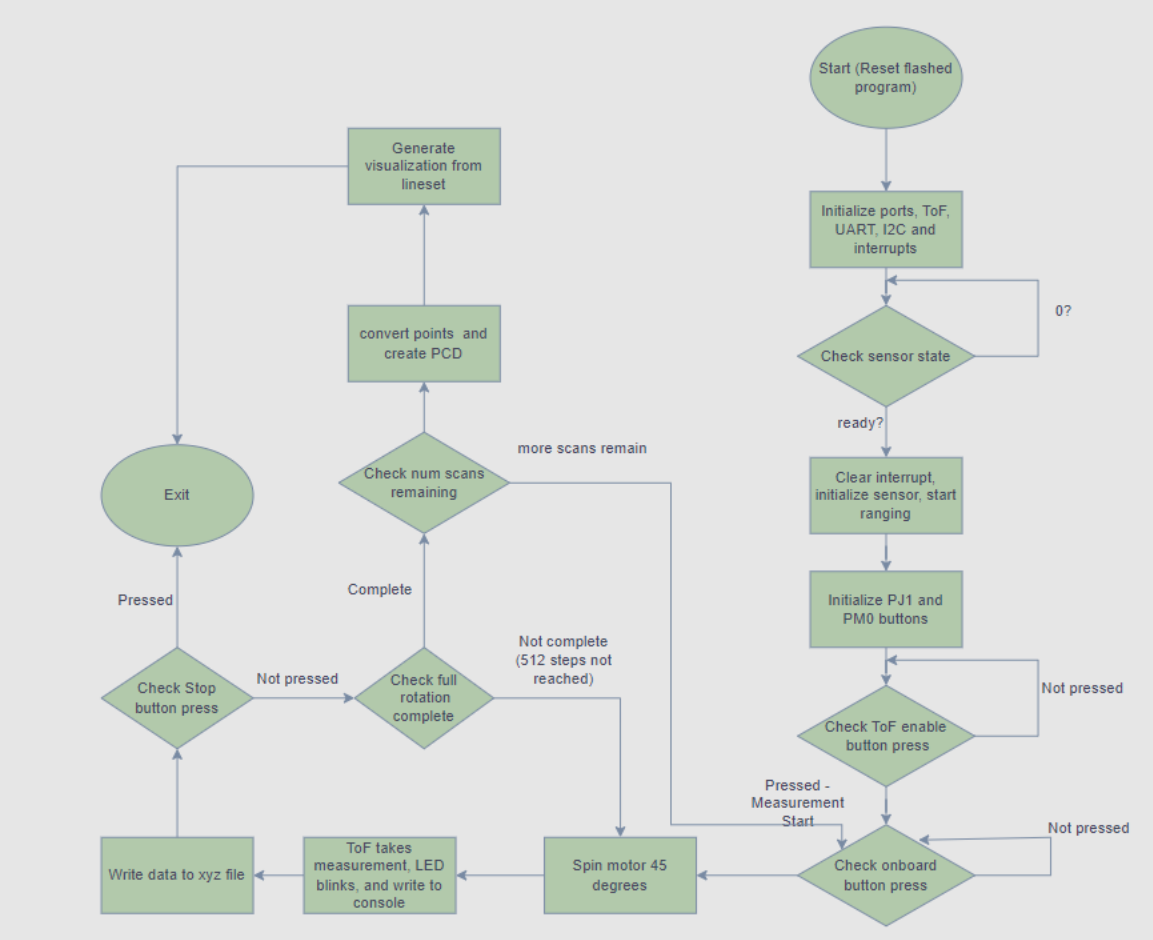


Figure 7: Logic Flowchart

9 Additional Diagrams

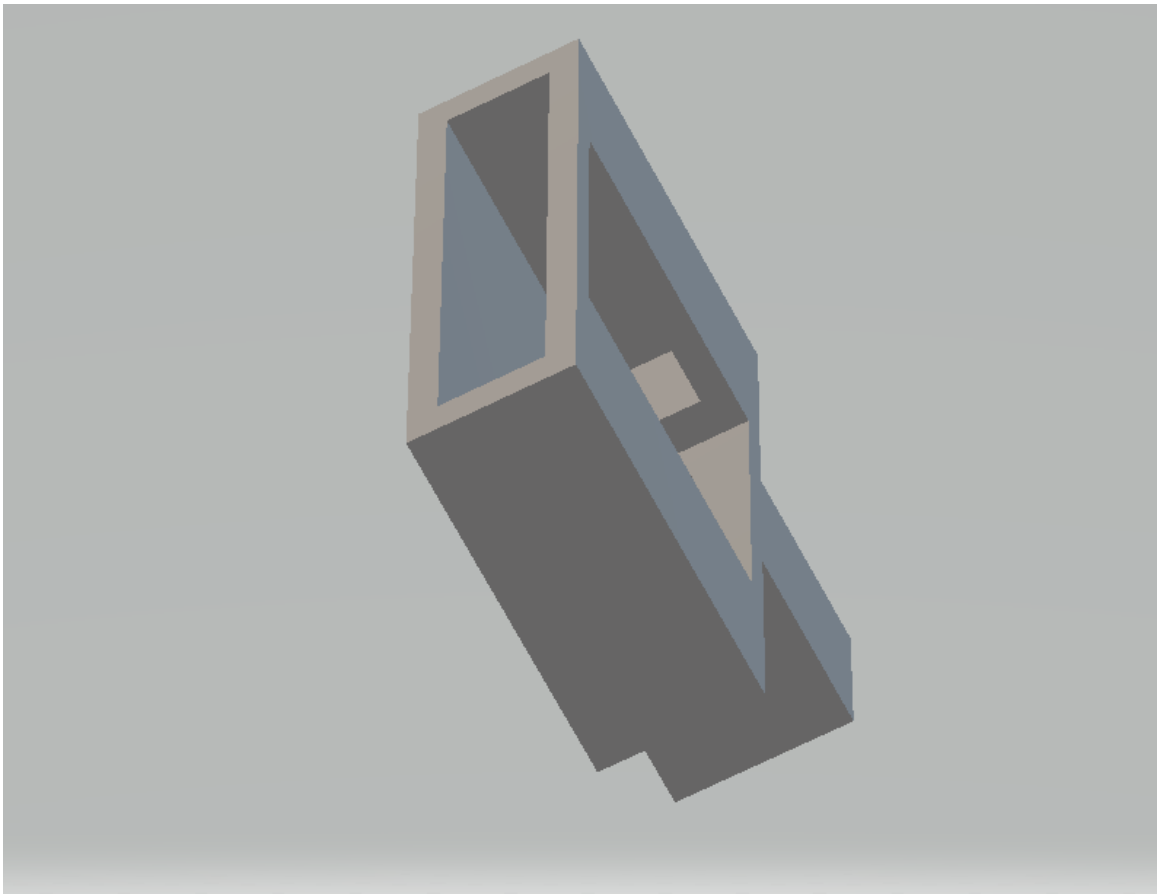


Figure 8: ToF Mount STL