Device Overview

A. Features

This embedded system was developed to capture and reconstruct 3D spatial data using the VL53L1X Time-of-Flight (ToF) sensor and a stepper motor. It features 360° scanning capability in the y-z plane, manual advancement in the x-direction, and user control via onboard buttons PJ0 and PJ1. Data is transmitted over UART and visualized live in Python.

The system uses three GPIO LEDs:

- Measurement Status LED: PN1 flashes when a measurement is taken
- UART Transmission LED: PF4 flashes at the start of each UART send
- Additional Status LED: PN0 remains ON while the motor is reversing direction and returning to the home position, and turns OFF when complete

B. General Description

The VL53L1X sensor measures distances based on the time of flight of infrared light. It is mounted on a stepper motor which rotates in increments of 11.25°, collecting one data point every 16 steps of the 512-step motor. This results in 32 points per full rotation. To prevent cable tangling, the motor alternates direction after each rotation.

Button PJ0 rotates the motor and initiates measurement, while PJ1 toggles the scan enable mode. Data, including distance, step number, spad count, and x-depth, is sent to the PC using UART communication at 115200 baud and plotted using Python. The system is designed to be modular, reliable, and easily adaptable for other environments.

C. Block Diagram (Data Flow Graph)

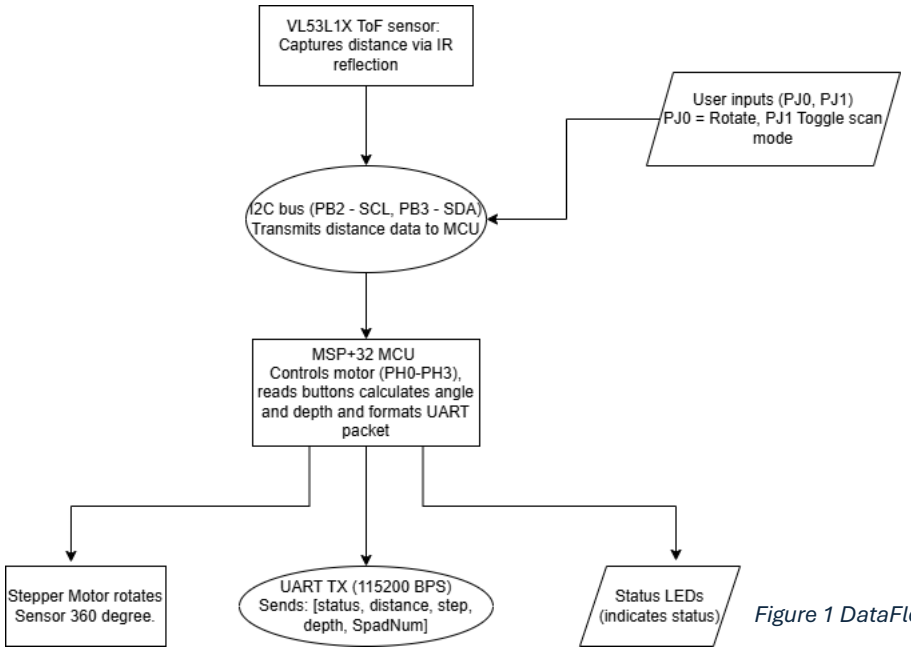

Figure 1 DataFlow Graph

Device Characteristic Table

Table 1 Device Characteristic Table

Component	Specification
Clock Speed	20MHz
UART Baud Rate	115200 BPS
Step angle	0.703º
Sampling Interval	Every 16 steps (11.25º)
Motor Control Pins	PH0, PH1, PH2, PH3
Button Pins	PJ0 (rotate), PJ1 (scan)
LED indicators	PN1 (measure), PF4 (UART), PN0 (status)
I2C Pins	PB2 (SCL), PB3 (SDA)

Detailed Description

A. Distance Measurement

The heart of this project is the VL53L1X Time-of-Flight (ToF) sensor by STMicroelectronics. This sensor emits an infrared laser pulse and measures the time it takes for the pulse to bounce back from a target surface. Using that time delay and the known speed of light, the sensor calculates the distance to the target with high precision and minimal noise.

Sensor Operation

The sensor communicates with the microcontroller using the I2C protocol via PB2 (SCL) and PB3 (SDA). In our implementation, the sensor is configured in short-distance mode for optimal response in indoor environments (like hallways), and polling mode is used rather than interrupt-based operation to simplify timing control.

At the start of each scan cycle, the sensor is initialized and calibrated. Once the device enters active scanning mode (triggered by pressing PJ1), it continuously captures distance values every 16 motor steps (i.e., every 11.25°).

The decision to take a measurement every 16 steps was made to balance spatial resolution and scanning speed:

- Each full motor revolution is 512 steps → 360°
- One data point every 16 steps → 32 samples per sweep
- This yields an angular resolution of 11.25°, which is sufficient for hallway-scale environments.

Stepper Motor Integration

To physically rotate the sensor, a unipolar stepper motor is connected to the microcontroller using GPIO ports PH0–PH3. These pins control the individual coil windings in a sequence, allowing precise angular motion. A delay of 2 milliseconds between step transitions ensures stable motor performance without missing steps.

The motor direction is tracked with a variable (flag), toggled after every 512 steps to reverse the rotation. This was implemented to prevent wire twisting and to allow bidirectional scanning. The total number of steps since the last direction reversal is tracked with a rotate counter. When rotate reaches 512 or more, the depth value (depth) is incremented, and the motor's direction is flipped.

Scan trigger and LED feedback

The scanning process is gated by two user-controlled buttons:

- PJ0: Each press steps the motor by 16 steps (11.25°), checks if a measurement is needed, and toggles scan flags.
- PJ1: Toggles scans enable mode. When scan is enabled, the PF4 LED (UART TX indicator) remains ON. When disabled, the LED turns OFF.

LEDs are also used as visual feedback mechanisms:

- PN1: Flashes during every successful measurement (Measurement LED)
- PF4: Remains ON when scan mode is active (UART LED)
- PN0: Remains ON when rotating 360° in the opposite direction to the direction of the scan (Status LED)

This implementation ensures the system can be operated without needing to look at a terminal or debugging screen.

UART Transmission Protocol

After a measurement is taken, the system formats and sends a UART packet to the connected PC over a baud rate of 115200. Each packet includes five fields:

- status: 1 if successful, 0 if timeout or invalid
- distance: raw millimeter value returned by the sensor
- step: the current step index (0–511)
- depth: the current x-axis depth level
- spadNum: number of active spad return signals (reliability metric)

These values are converted into ASCII strings with delimiters (such as commas or tabs) and terminated with a newline character for easy parsing in Python. Transmission is non-blocking, and we include a short delay after sending each packet to prevent UART buffer overflows.

B. Visualization

On the PC side, a Python script opens the correct COM port and listens for incoming UART packets. Each line of data is parsed, verified, and converted into usable numerical values. The script performs the following key steps:

Angle Calculation

The motor rotates in discrete steps. The angle corresponding to each step is calculated as:

$$\theta = \frac{360}{512} \times step \approx 0.703° \times Step$$

To reduce jitter due to floating-point drift, the angle is rounded to three decimal places.

Cartesian Conversion

The polar data (distance + angle) is converted into 3D Cartesian coordinates using trigonometry:

$$x = depth$$
$$y = distance \times \sin(\theta)$$
$$z = distance \times \cos(\theta)$$

This assumes that depth is constant per sweep and only angle and distance vary.

3D Plotting

Once coordinates are calculated, they are added to a live plot using Python's matplotlib 3D scatter plotting. The plot axes are scaled automatically based on the range of incoming values, and users can zoom and rotate the plot.

Depth Management

After every full 360° sweep (512 steps), the system:

- Toggles direction
- Increments depth by 1
- Resets the step count

This design creates a layered scan structure along the x-axis. The final 3D point cloud includes several layers corresponding to each depth slice.

Application Note

This setup assumes that the user has already installed the required software and dependencies, including Keil uVision for firmware development, Python 3, and libraries such as pyserial, numpy, and matplotlib. The MSP432E401Y microcontroller must be properly configured in Keil prior to flashing the code.

1. Plug the MSP432 board into your PC via USB.
   Open Device Manager, scroll to Ports (COM & LPT), and expand the list.
   Note the COM port number corresponding to XDS110 Class Application/User UART (COM#), this will be needed for the Python script.
2. Open the provided Python script (Data.py).
   In the script, locate the serial port variable and set the COM port to match the one found in Step 1 (e.g., COM3).
3. Build the circuit according to the pin mappings listed in the Device Characteristics Table:
   - ToF sensor over I2C: PB2 (SCL), PB3 (SDA)
   - Stepper motor: PH0–PH3
   - Buttons: PJ0 (rotate), PJ1 (scan toggle)
   - LEDs: PN1 (measure), PF4 (UART), PN0 (rotation complete)
4. Open the Keil .uvprojx project file.
   Click ‘Translate’, then ‘Build’, then ‘Load’ to compile and flash the program.
5. After loading, press the reset button on the board to initialize.
6. To begin scanning, press PJ1 (Scan Enable). This will toggle the status LED (PF4).
   Pressing PJ0 will now rotate the stepper motor by 11.25° and take a measurement.
7. In the Python script, configure how many x-steps (depths) you'd like to scan. This determines how many times the setup will be physically moved forward along the x-axis.
8. Run the Python script. After which the terminal will ask for you to enter the number of frames you want to take (number of x-increments), enter the number of frames and click enter.
9. Each time PJ0 is pressed, the device will take another scan. After one full rotation (512 steps), the direction will reverse, and the depth will increment automatically.
10. After completing the desired number of depths, the resulting 3D scan will available to see by opening the other python Script provided (3D.py). the terminal will ask for you to enter the number of frames you took after which a 3D image will popup. Once that pops up, close the first pop up to get the result.

Expected Output

If the instructions were follow as stated the expected results will be as follows:

Figure 2 Hallway being scanned

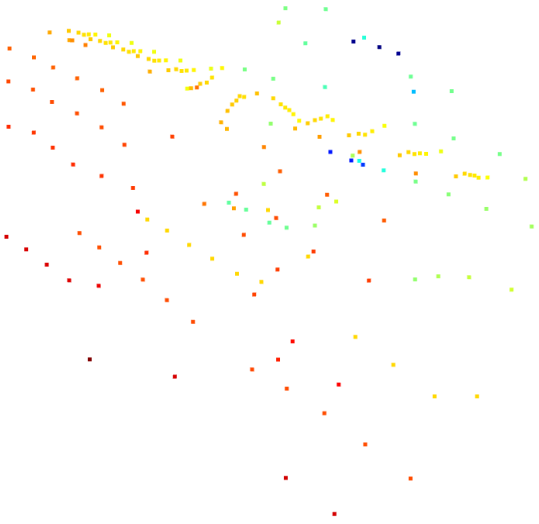

Figure 3 first 3D scan (first pop-up)

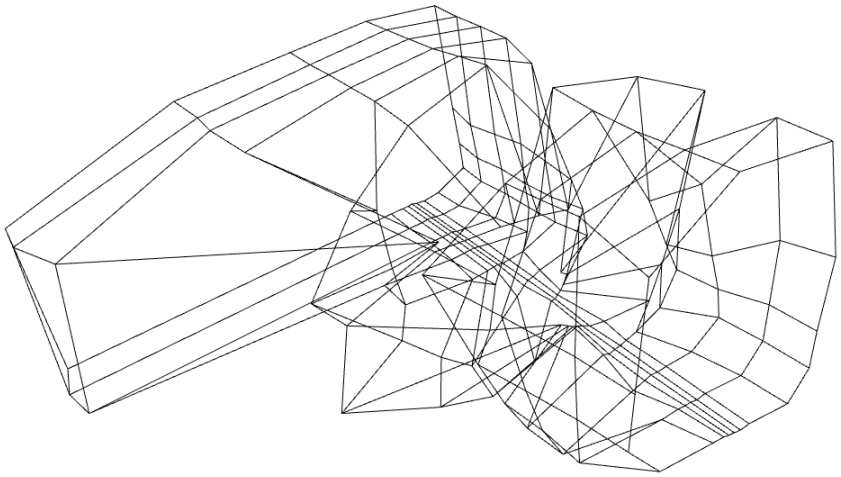

Figure 4 3D scan (second pop-up)

Note that the discrepancy on the right side is because the microcontroller moved during the scan causing the points to be further on one side and closer on the other. Other than that you can see it picked up the feature on the hallway. You can clearly seen the turn on the hallway at the end or the scan and see it's the same as the photo above.

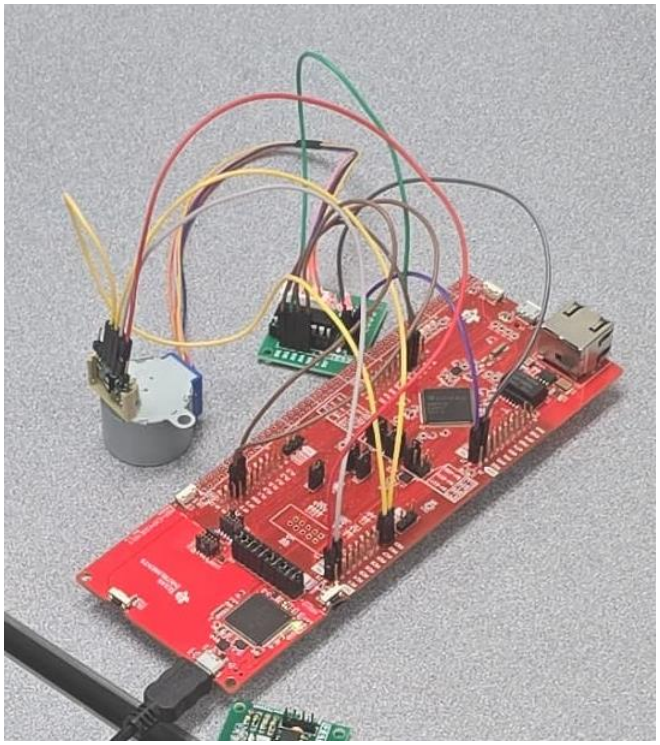

Figure 5 Physical Circuit

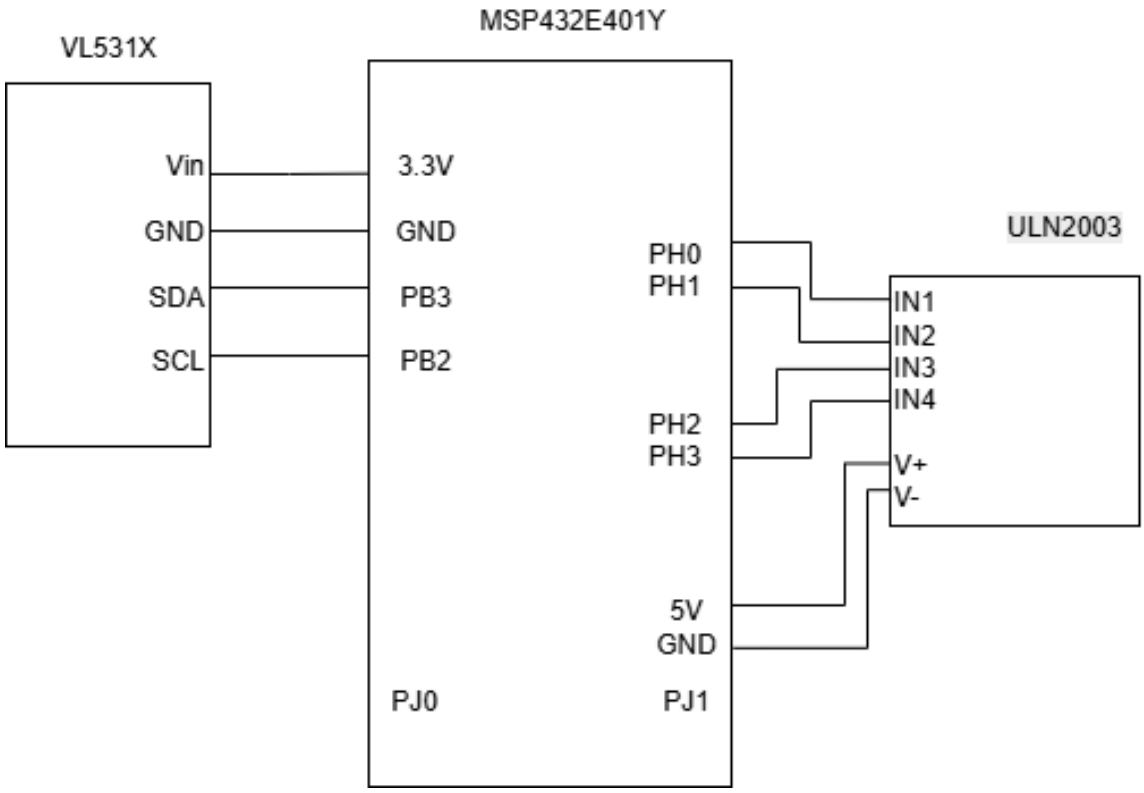

Figure 6 Circuit Schematic

Programming Flow Chart

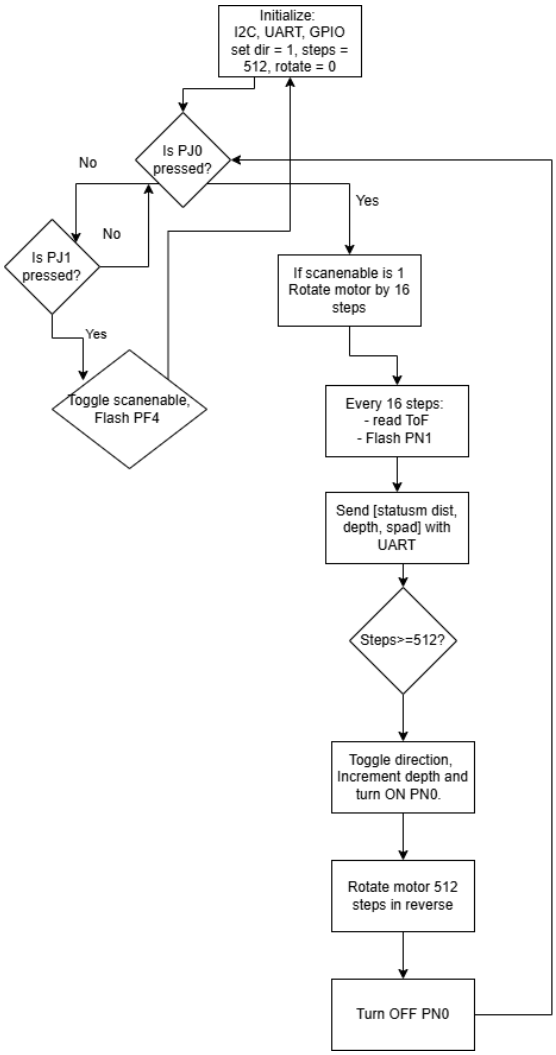

Figure 7 Program Flow Chart