COMPENG 2DX3

Final Project Report

Course instructor : Dr. Doyle, Dr. Haddara, Dr. Athar

Date: April 7th ,2025

Ali Bandali – 400532826 – bandaa2

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. Submitted by [**Ali Bandali, 400532826**]

Table Of Contents

InSight360 LiDAR Scanner

1 Device Overview

Insight360 is a sophisticated LiDAR scanning system engineered for indoor spatial analysis. The system employs vertical scanning techniques to capture environmental details, which are subsequently transformed into precise three-dimensional visualizations using dedicated software. The device was built using the following components.

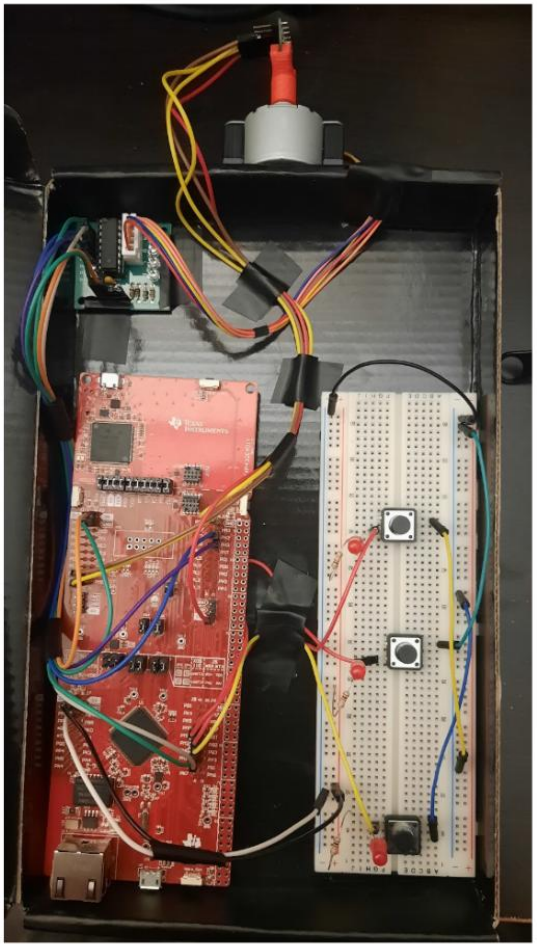


Figure1: Insight360 scanner hardware

Features

- MSP432E401Y Microcontroller Launchpad™
  - 32-bit ARM cortex-M4 CPU
  - Operating Voltage between 2.5V – 5V
  - Bus Speed: 14 MHz (Default speed of 120 MHz)
  - 4 on board LEDS, GPIO pins, ADC
  - 1024KB of Flash memory, 256 KB of SRAM, 6KB EEPROM
- VL53LIX Time of Flight Sensor
  - Distance Range:
  - Operating Voltage: 2.5V – 3.3V
  - Ranging Frequency of 50Hz
  - Maximum Distance measurement: 4 meters
- 28BYJ-48 Stepper Motor and corresponding ULN2003AN driver
  - 512 steps per 360-degree rotation
  - 5V supply
  - 4 on board status LEDS to show phase of the motor
- Serial Communication:
  - UART used to communicate with the microcontroller to users PC
  - I2C communication used between Time-of-Flight sensor and microcontroller

    - Baud rate for UART: 115200 BPS

- Receiver Side Programming Language:
    - MATLAB is used for distance calculations, data plotting and 3D visualization
    - MATLAB also used for serial communication with the microcontroller listening at 115200 BPS
- Component Costs:
    - MSP432E401Y Microcontroller Launchpad™: $72.54 (Digi Key)
    - Stepper motor and driver circuit: $6
    - VL53LIX Time of Flight Sensor: $30
    - Jumper cables, breadboard, buttons and other small electrical components: $40

General Description

The signal acquisition process begins with the VL53L1X time-of-flight sensor, which emits an infrared laser pulse and calculates the distance by measuring the time it takes for the light to be reflected back from an object. This is achieved by the sensor's internal circuitry that pre-processes the incoming analog signal and converts it into a digital distance value using its built-in analog-to-digital conversion method. The distance is calculated by the sensor using the formula:

Measured Distance = (Photon Travel Time / 2) × Speed of Light,

which accounts for the round-trip travel of the light beam. The sensor is initialized and calibrated through API methods, ensuring that each measurement is accurately processed and then communicated over the I2C bus to the microcontroller.

Once the microcontroller receives the digital measurement data via I2C, it formats the data into a comma-separated string that includes parameters such as the range status, the measured distance and the motor step count (which corresponds to the angular position).This formatted string is then transmitted via the UART interface to an external PC. The UART subsystem of the microcontroller sends the measurement data at a fixed baud rate so that the PC reliably receives the information.

On the visualization side, the MATLAB code running on the PC establishes a serial connection and waits for a "START_CYCLE" signal before beginning to read the measurements. As data is received, a custom parse function extracts the numerical parameters from the comma-separated string and calculates the angular position using the known conversion factor (e.g., raw motor steps converted to degrees). The MATLAB code then converts the polar coordinates—comprising the measured distance and the step-derived angle—into Cartesian coordinates using trigonometric functions. The resulting (x, y, z) coordinates are plotted in 3D space, with points

from each full rotation connected into closed loops and connected vertically across cycles to create an accurate three-dimensional model of the scanned environment.

Block Diagram

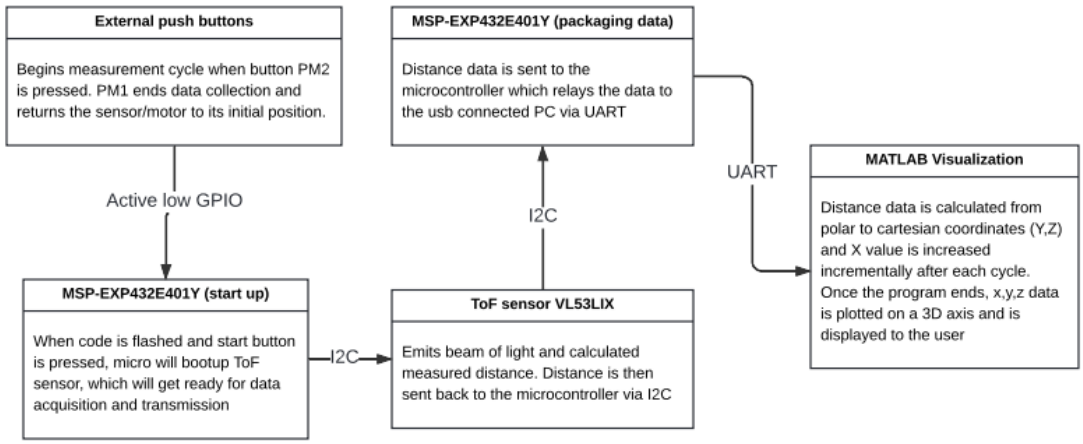


Figure 2: Block Diagram

2 Device Characteristic Table

Microcontroller		Stepper Motor		ToF sensor	
Characteristic	Value	Driver IC Pins	Microcontroller Pins	ToF Pins	Microcontroller Pins
Clock Speed	14MHz	IN1	PH0	Vin	3.3V
Measurement, UART, and status LED	PN0, PN1, PF4	IN2	PH1	GND	GND
Serial Port	COM5	IN3	PH2	SCL	PB2
Baud Rate	115200 bps	IN4	PH3	SDA	PB3
Push Button 1 (start/stop motor)	PM2	V+	5V		
Push Button 2 (end scan and homing)	PM1	V-	GND		
Push Button 3 (return home and continue scan)	PM0				

3 Detailed Description

Distance Measurement: Acquisition

The VL53L1X time-of-flight sensor measures the distance to an object by emitting an infrared laser pulse and timing the return of the reflected signal. Internally, the sensor's circuitry handles the signal conditioning and analog-to-digital conversion of the received analog signal. The sensor computes the distance using the formula:

$$Measured\ Distance\ = \frac{1}{2}\ (Flight\ time)\ \times\ (Speed\ of\ Light)$$

For example, if the sensor detects that the round-trip travel time of the photon is 6.67 nanoseconds, then the one-way travel time is 3.335 nanoseconds. Multiplying 3.335×10^{-9} seconds by the speed of light (approximately 3×10^{8} m/s) gives a distance of about 1 meter, which the sensor converts into millimeters (e.g., 1000 mm).

In the C code for the controller, this measurement process is initiated and managed in the sensor initialization and ranging routines. Specifically, after the sensor boots (using VL53L1X_BootState) and interrupts are cleared (VL53L1X_ClearInterrupt), the main loop waits for data readiness by calling VL53L1X_CheckForDataReady. Once the sensor signals readiness, the code retrieves the distance (using VL53L1X_GetDistance), along with other diagnostic parameters, ensuring that all distance values are expressed in millimeters.

3.1 Distance Measurement

The sensor communicates with the microcontroller over the I2C bus. The I2C_Init() function in the C code configures the I²C0 module on Port B (specifically on PB2 for SCL and PB3 for SDA). This ensures that the digital output from the ToF sensor (now containing the measured distance in mm, range status, and other parameters) is reliably received by the MSP-EXP432E401Y microcontroller. Measurements are taken at fixed angular intervals. In the code, this is achieved by checking whether the motor's step count (tracked by the variable "stepCount") modulo 64 equals zero; 64 steps correspond to an 11.25° rotation. Thus, for every 360° rotation (2048 steps, since 2048/64 = 32), the system collects 32 measurements. After each measurement, the microcontroller formats the data into a string (e.g., "%u,%u,%u,%u,%u\r\n") and transmits it over the UART interface at a baud rate of 115200 bps to the external PC. COM5 is the default port where the USB is to be plugged in although this will be different for differing computers.

Once the microcontroller sends data via UART, MATLAB reads and processes it. In the MATLAB script, after establishing a serial connection (lines 1–10 in the code), the program waits for the "start_cycle" trigger before beginning data acquisition. When data is received, the custom parse() function (located at the end of the MATLAB script) is invoked. This function uses sscanf() to extract five comma-separated values: range status, distance (in mm), raw step count, depth, and SPAD number. It then converts the raw step count into an angular measurement by applying the conversion factor:

$$angle\_deg = mod(angle\_raw, 2048) \times \frac{11.25}{64}$$

For example, if the raw step count yields a value that converts to 45° (i.e., angle_deg = 45°), and if the sensor reports 1000 mm, then the polar coordinates are (1000 mm, 45°). MATLAB's pol2cart() function (called on line 3 in the main loop) converts these polar coordinates into Cartesian coordinates. Using the conversion formulas:

$$x = distance \times sin(\theta)$$

$$y = distance \times cos(\theta),$$

if θ = 45° (or 0.7854 radians), the calculations yield:

x = 1000 × sin(0.7854) ≈ 707 mm,

y = 1000 × cos(0.7854) ≈ 707 mm.

After using pol2cart(), MATLAB re-maps the coordinates so that the final representation of the environment is based on a coordinate system where the x-axis corresponds to depth, the y-axis corresponds to width, and the z-axis corresponds to height. For instance, the MATLAB code reassigns the output with the line "cartesian_data = [-z, -x, -y]," ensuring the measured data is aligned with the design specifications.

Displacement is implemented via a "depth" variable in the microcontroller's C code. Each full 360° rotation of the stepper motor is tracked by a variable ("stepsInCycle"), and when it reaches 2048 (indicating a complete sweep), the depth variable is incremented. This depth value (sent as part of the UART string) simulates physical forward displacement in the scanned environment. This ensures that with each full rotation a new "slice" of the environment is captured. In MATLAB, the depth value is scaled (using depth_scale = 300) to establish a tangible displacement along the x-axis. This means that every layer is offset, simulating movement through a room. The MATLAB script also includes handling for ending a measurement cycle. Inside the main measurement loop, the script checks each line of data received from the serial port. When a line containing "END_CYCLE" is detected—this text is sent by the microcontroller when the button connected to PM1 is pressed. This part of the MATLAB code,

located within the measurement reading loop, causes MATLAB to exit the loop and thereby stops further data acquisition. This “END_CYCLE” signal indicates that the user has pressed the designated homing button (PM1) to complete the scanning process.

3.2 Visualization of the 3D Scan in MATLAB

After converting the polar measurements into Cartesian coordinates, MATLAB uses this data to construct a three-dimensional visualization of the environment. The MATLAB script first accumulates all measurements into a matrix, where each row represents a single data sample in the form [Distance, Angle, Depth]. The pol2cart() function then converts these measurements from polar (angle, distance) to Cartesian coordinates. In our project, the final coordinate mapping is defined as follows:

- The **x-axis** represents “depth” into the environment, derived from the depth variable scaled by 300
- The **y-axis** represents “width”, which is obtained from the cosine component of the polar conversion
- The **z-axis** represents “height”, obtained from the sine component of the polar conversion.

The MATLAB code organizes the data by unique depth layers and then uses scatter3() to display the points. Furthermore, the MATLAB script provides real-time command-line feedback. As the sensor’s measurement data is received, the parse function prints the calculated Cartesian coordinates to the MATLAB command window. This allows users to monitor the incoming data directly. It further enhances the visualization by using plot3() to draw connecting lines between sequential measurements in each layer and vertical lines connecting corresponding points across layers, thereby forming a continuous 3D representation. This results in an interactive visualization that accurately depicts the indoor space by combining the precise distance measurements and the stepper motor’s controlled angular increments.

4 Application Example:

Figure 3: Image of hallway Scanned

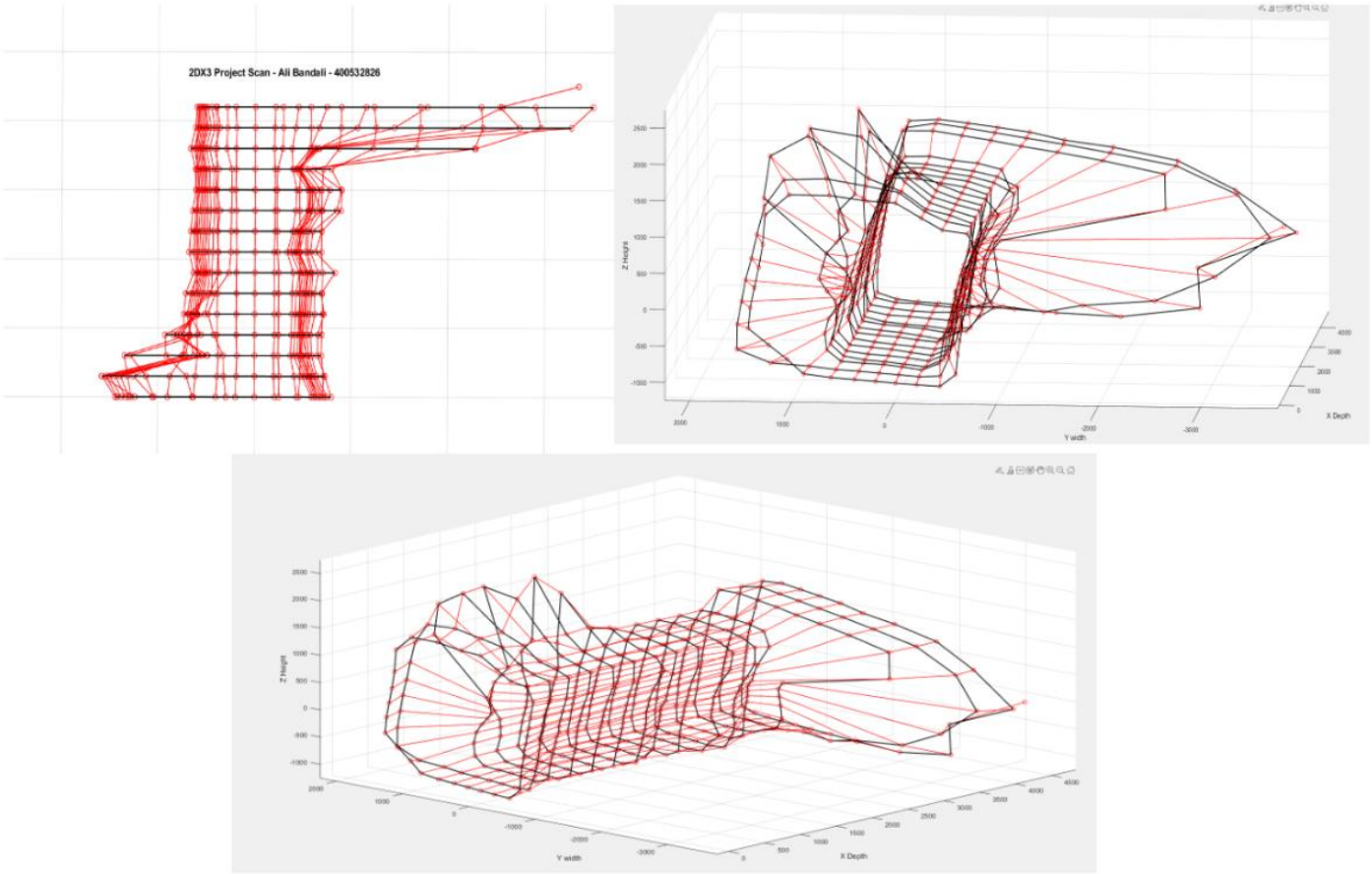


Figure 4: 3D MATLAB Plot from Scan

4.1 Hardware Setup

1. assemble the device as shown in your circuit schematic (refer to the device characteristics for full pin descriptions):
    - **Microcontroller connection:**
        - connect the MSPEXP432401Y via usb to your PC so that it receives power and the debug/uart port becomes active.
        - verify that the bus speed is properly configured
    - **Stepper motor & ToF sensor mounting:**
        - securely attach the tof sensor (vl53l1x) to the mount on the stepper motor using a tape or a 3d printed holder. ensure that the sensor rotates with the motor.
        - attach the motor mount to the opposite end of the microcontroller (this keeps the USB connection free from mechanical strain).
        - connect the stepper motor coil inputs to port h pins (ph0, ph1, ph2, ph3). include a driver circuit
    - **I2C wiring for sensor:**
        - connect the tof sensor to the i2c bus on the microcontroller. wiring is done via port pb2 and pb3 on the controller
    - **Push buttons and indicators:**
        - attach external push buttons to port m (pm0, pm1, pm2) in active low configuration. Each button should have its lower end connected to ground so that internal pull-ups work correctly. these buttons toggle motor scanning, change scanning intervals, and run the homing routines.
    - **Power requirements:**
        - supply 3.3 v to the tof sensor and 5 v to the stepper motor from the controller. All components share a common ground with the microcontroller.

4.2 Software setup and configuration

1. **Keil project setup and code flashing:**
    - open keil and create a new project targeting the tm4c1294ncpdt.
    - add all required source and header files to your project – for example:
        - main (containing your device code)
        - pll.c and pll.h
        - systick.c and systick.h
        - uart.c and uart.h
        - vl53l1x_api.c and vl53l1x_api.h
        - any other supporting files.
    - compile and flash the keil code to your microcontroller. after programming, reset the device; observe that the onboard leds flash to indicate that initialization is complete.
2. **MATLAB configuration:**
    - on your windows system, open the matlab script that handles data collection and 3d visualization.
    - check the device manager to determine the correct com port that your microcontroller is using (for example, com5). update the matlab script's portName variable accordingly.
    - confirm that the baud rate (115200) and timeout settings match the configuration in your uart_init() function.
3. **Synchronization between keil and matlab:**
    - verify that uart messages from the microcontroller (such as "start_cycle", "new_cycle", and "end_cycle") remain unchanged so that matlab can correctly identify cycle start, cycle end, and intermediate events.

4.3 Operational flow and testing

1. **Initial system start-up:**
    - power on the microcontroller and ensure that all onboard leds flash twice; this indicates that the pll, systick, and uart subsystems are active and that the tof sensor has been properly initialized.
    - open matlab and run the script; matlab will set up the serial port and wait for the "start_cycle" command from the microcontroller.
2. **Starting the scanning cycle:**
    - press the reset button on the microcontroller so that the system begins its initialization sequence.
    - once matlab prints "start_cycle detected. now reading measurements…" in the console, press the dedicated push button (button 0 on port m) to toggle motor scanning on. this will begin the automatic stepping sequence of the motor.
    - as the motor rotates, note the following:
        - every 64 steps (or 11.25° rotation), the tof sensor takes a measurement.
        - the microcontroller blinks the measurement led on pn0 and the uart led on pf4 to provide visual feedback for each measurement.
3. **Data collection and communication:**
    - each measurement produces a uart string with the following comma-separated data: range status, distance, current step count (or angle), depth, and spad number. an example output might be:
        - "incoming string: 0,8,54,0,1"
        - followed by coordinate data printed by the matlab script, for example: "[x = 0.00, y = -8.57, z = -1.47]"
    - after one full rotation (2048 steps) the microcontroller sends a "new_cycle" message, which matlab interprets as an indication to start a new layer (depth). this requires the user to move the device forward as needed.
4. **Using homing and cycle termination routines:**
    - if the scanning cycle needs to be terminated, press the homing button (for example, button 1 on port m configured for homing1). this will command the microcontroller to run a homing routine, returning the step count to zero.

- the microcontroller then prints “end_cycle detected. stopping read loop.” and matlab automatically proceeds to generate the final 3d model based on the collected data.

4.4 Coordinate definition and data visualization in MATLAB

- in our system, the conversion from polar to cartesian coordinates assigns:
  - the **x axis** for displacement (representing the forward movement between successive scan layers),
  - the **y and z axes** for vertical distances (width and height) (or slices).
- MATLAB uses the pol2cart function to convert the tof sensor’s data, and further reassigns the axes so that the final 3d model correctly reflects the physical structure of the scanned environment.
- after the scan cycle is completed and the “end_cycle” command is received, matlab automatically generates a 3d plot of the data, showing:
  - a scatter3 point cloud display for each layer,
  - connected closed loops for each full rotation, and
  - vertical lines connecting corresponding points between layers.

5 Limitations

1. **Limitations of the microcontroller's FPU and the use of trigonometric functions**
   The MSPEXP432401Y incorporates a 32×32-bit single precision floating point unit that is capable of handling basic arithmetic and trigonometric operations. however, this single precision inherently limits the accuracy of calculations when using functions such as sin and cos. due to the repeating decimal nature of these functions, rounding errors can accumulate during successive computations. while the fpu accelerates operations compared to software-based routines, these accumulated inaccuracies mean that for high precision tasks (for example, in detailed data plotting) it is preferable to offload these computations to a pc, whose double precision arithmetic in matlab yields greater overall accuracy.

2. **Maximum quantization error for the tof module**
   The tof sensor provides a digital reading based on a 12-bit representation. assuming a maximum measurement range of 4000 mm, the maximum measurement quantization error is given by:
   $maximum\ quantization\ error\ =\ maximum\ range\ /\ (2^{12})$

   $4000\ mm\ /\ 4096\ \approx\ 0.9766\ mm$

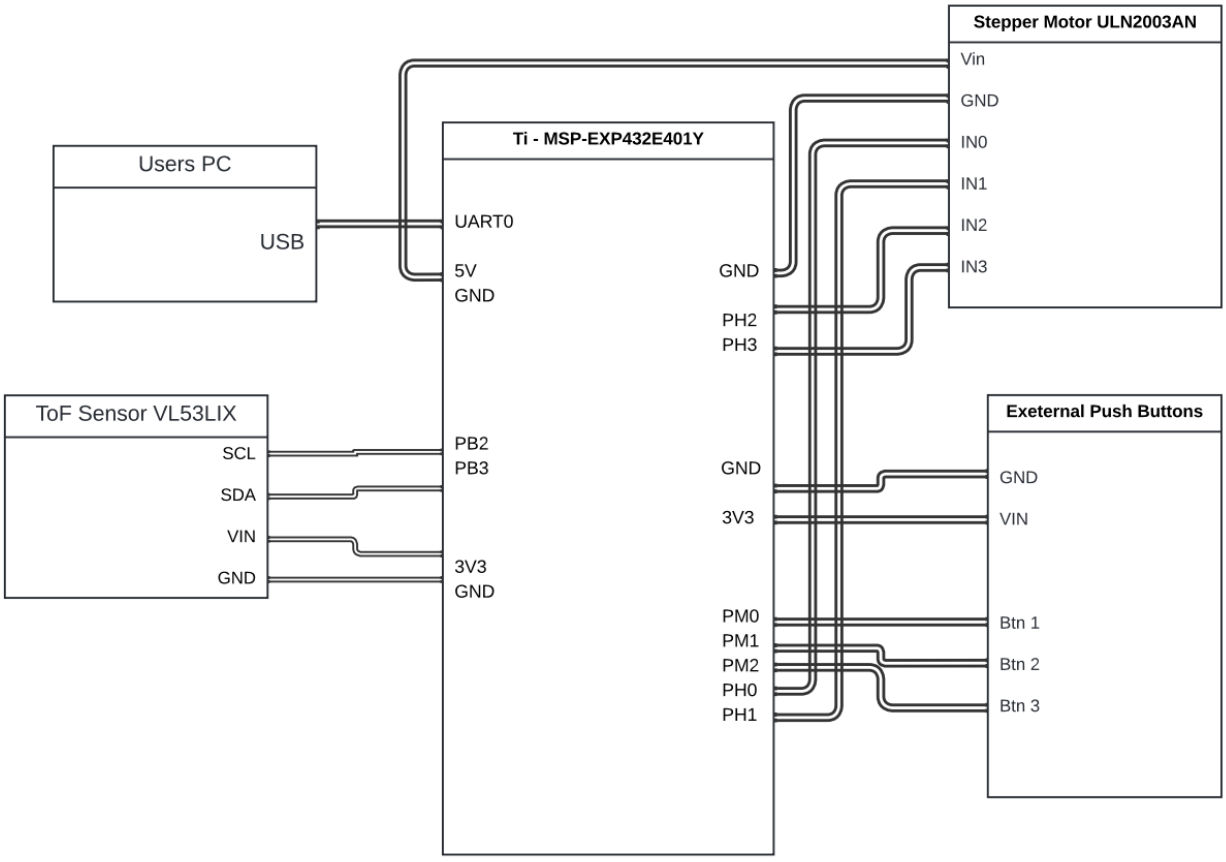
   thus, each digital step represents roughly 1 mm in sensor resolution. this defines the finest granularity the system can reliably distinguish in distance readings.

3. **Maximum standard serial communication rate and verification**
   The maximum baud rate available on the pc for serial communication is approximately 115200 bps, although this does depend on hardware. For this system, the implemented rate was set to 115200 bps for robust and reliable data transmission. verification of the maximum rate was performed by inspecting the device manager on the pc under the relevant com port settings, specifically, checking the bps values listed for the xds110 class application/user uart port confirmed these rates.

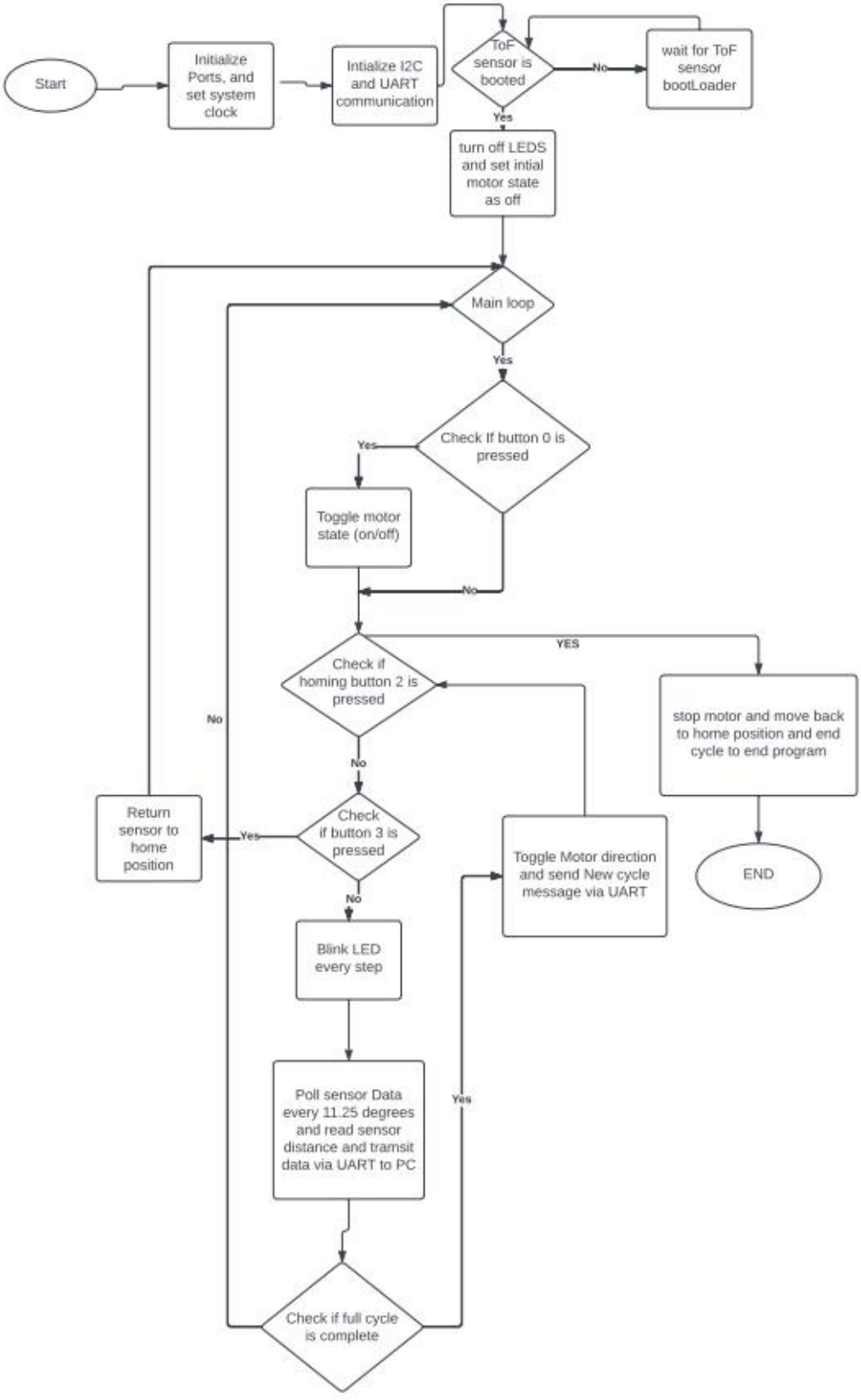
4. **Communication method and speed between the microcontroller and tof module**
   The system uses the i2c protocol to interface the tof sensor with the microcontroller. in the keil code, the i2c_init() function configures the communication at 100 kbps. this standard-mode i2c setting is well within the tof sensor's capabilities and ensures that the sensor data, refreshed at roughly 50 hz, is transmitted reliably.

5. **Primary limitations on system speed and testing methodology**
   The stepper motor's rotational speed is the primary bottleneck in the system's operation. while the tof sensor and serial communications also contribute to overall speed constraints, the motor's performance is limited by the required time delays between the four discrete phases needed to perform each step. to test this, delays were gradually reduced in the keil code until skipped steps or mechanical instability were observed; the minimum delay that still ensured reliable motor operation was found to be approximately 2 ms per phase. this testing confirms that the motor's mechanical limitations are the most significant factor affecting the maximum operational speed of the system.

6 Circuit Schematic



7 Flow chart (controller and MATLAB)



MATLAB Flow chart