Final Project Report

COMPENG 2DX3: Microprocessor System Project

Taimur Ahmed – ahmedt52 – 400514463

2025-04-09

Contents

Device Overview – Part A: Features

This device combines multiple components, each intended for a particular function, and is controlled by a straightforward interface of push buttons and indicator LEDs which allow for easy usability and overall efficient system. It includes:

- **Processor**: 32-bit Cortex-M4F Processor Core with Floating-Point Unit (FPU)
- **Memory**: 256 KB SRAM, 1024 KB Flash Memory
- **Bus Speed**: 22MHz (in our case)
- **Interfaces**:
    - I2C Interface (Up to 3.33 MHz) update based on the documentation
    - UART Interface (Up to 128 kbps)
- **GPIO Pins**: For connection to other devices
- **User Input**: 1 user button to control device status
- **Indicators**: 3 user LEDs for real-time device status indication

28BYJ-48 Stepper Motor:

- **Resolution**: 4 phases per step, 512 total steps for a full 360-degree rotation
- **Rated Voltage**: 5V DC
- **Indicators**: 4 LEDs on ULN 2003 PCB indicating current phase

VL53L1X Time of Flight Sensor:

- **Function:** Measures distance using LIDAR
- **Distance Measurement**: Up to 4m with high accuracy and minimum 40mm.
- **Field of View:** 15 to 27 degrees (programmable)
- **Operating Voltage (Vin)**: 2.6V - 3.5V

Serial Communication Protocols:

- **I2C**: Used for efficient communication between the ToF sensor and the microcontroller. The ToF has up to 400KHz serial bus programmable address.

- **UART**: Enables communication between the board and PC, as well as data capture from the motor.

3D Visualization:

- **Tools**: Python with PySerial for serial communication and open3d for plotting points and visualization.
- **Functionality**: The Python scripts work along with Open3d. Everything works together to read the data, execute point plotting, perform 3D reconstruction. Python script parses data and collects datapoints which is then read by Open3D for visualization.

General Features:

- **Simple Interface**: 1 external push buttons for easy operation and 3 LEDs indicating the system's operational status. 2 of the LEDs flash every 11.25 degrees. Once 360 degrees is complete, the stepper motor returns home and the third led blinks upon the motor returning home.
- **Push Button Functions**:
    - Button 1 enables the motor to perform a full 360-degree rotation, taking data every 11.25 degrees. Pressing the button again during data collection pauses the process.

Device Overview – Part B General Description

Device Overview - Part B: General Description

This system is a high-performance, cost-effective 3D spatial mapping solution designed to be user-friendly. It utilizes cutting-edge technology, including the MSP432E401Y Simple Link™ Ethernet Microcontroller, the VL53L1X Time-of-Flight Sensor, and the 28BYJ-48 Stepper Motor coupled with the ULN 2003 PCB This combination leverages I2C and UART protocols for robust data transfer and communication.

System Interface:

- **Control Elements**: One user operated button PJ0 to control the system operations and data collection.
- **Status Indicators**: 3 LEDs (D4, D3 and D1 on the MSP432E401Y) provide real-time feedback on the system's status and data transfer processes.

Operation:

- **Initialization**: Button PJ0 triggers the start of each 360-degree scan in the zy-plane, with each scan position spaced to include x-axis data incrementally.
- **Rotation Mechanism**: To prevent wire entanglement and ensure system integrity, the motor returns home after each scan is completed first and then we can press PJ0 again to start a second scan.

Data Processing and Visualization:

- **Data Transfer**: The ToF sensor sends readings to the microcontroller via I2C. These readings are then transferred to a PC using UART, where python processes the data.
- **3D Visualization**: Python scripts transform the polar coordinates into Cartesian coordinates (XYZ), facilitating the plotting of 3D visualizations in real-time.
- **Measurement Calculation**: The ToF sensor calculates distances by measuring the time it takes for a light beam to return, using the formula:

$$D = \left(\frac{inflight\ time}{2}\right) * speed\ of\ light$$

Device Overview – Part C: Block Diagram

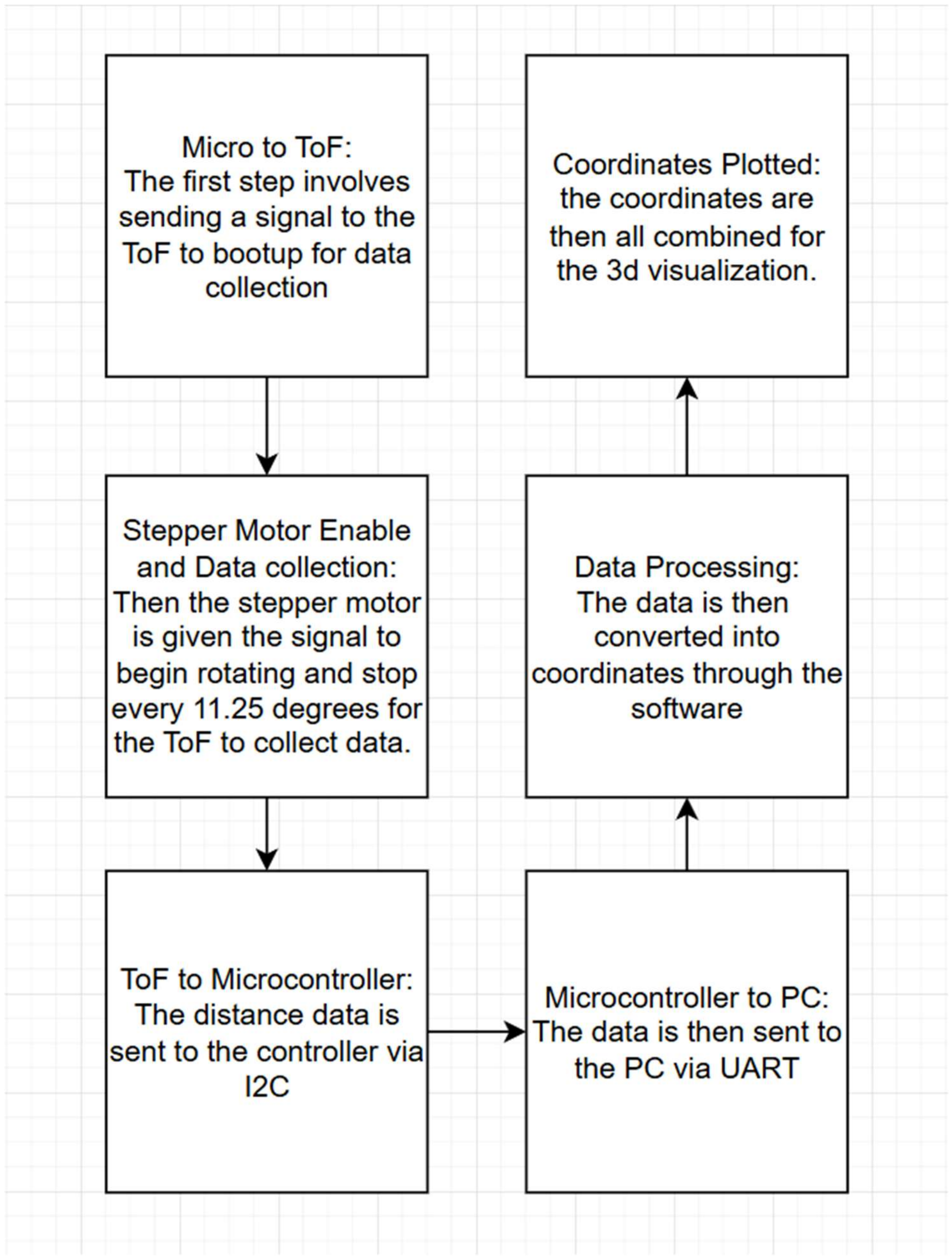

Micro to ToF:
The first step involves sending a signal to the ToF to bootup for data collection
Stepper Motor Enable and Data collection:
Then the stepper motor is given the signal to begin rotating and stop every 11.25 degrees for the ToF to collect data.
ToF to Microcontroller:
The distance data is sent to the controller via I2C
Microcontroller to PC:
The data is then sent to the PC via UART
Data Processing:
The data is then converted into coordinates through the software
Coordinates Plotted:
the coordinates are then all combined for the 3d visualization.

Figure 1: Block Diagram

Device Characteristics

MSP432E401Y Microcontroller	
Device/Feature	**Description**
Clock Speed	22MHz Bus Speed
Status Led (Measurement)	PF4
Status LED (UART Transmission)	PF0
Push Button 1 (start/stop motor)	PJ0
Additional Status LED	PN0
Serial Port	COM7

VL531LX Time of Flight Sensor	
Device/Feature	**Description**
GND	3.3V
GND	GND
SDA	PB3
SCL	PB2

ULN 2003 Stepper Driver Board	
Device/Feature	**Description**
V+	5V
V-	GND
In1	PH0
In2	PH1
In3	PH2
In4	PH3

Detailed Description – Part A: Distance Measurement

The system uses the VL53L1X Time-of-Flight (ToF) sensor, which utilizes infrared light for precise distance measurements. The sensor emits a beam of light that reflects off nearby objects and returns to the sensor's receiver. By calculating the time delay between emission and reception, the sensor determines the distance to the object using the formula:

$$D = \left(\frac{inflight\ time}{2}\right) * speed\ of\ light$$

After the measurement, the sensor processes the analog data through transduction, conditioning, and Analog-to-Digital Conversion to output digital distance data. This information is then sent to the microcontroller via I2C communication.

System Operation: The microcontroller continuously polls for user input via one external push buttons, PJ0. PJ0 initiates the stepper motor to perform a full 360-degree clockwise rotation, taking data points at every 11.25 degrees. Pressing PJ0 again pauses data collection at any during the scanning. The data collected is transmitted to a connected PC for visualization through UART.

Motor Control: Each 360-degree scan is divided into steps, with a total of 512 steps per rotation. At each step, the ToF sensor captures a distance reading, and this information is sent to the microcontroller via I2C, then transmitted to the PC over UART.

Data Capture and Communication:

- **Distance Measurement**: Every time a distance reading is captured, the system stores it and in the end, it is written to a .xyz file.
- **LED Indicators**: The system uses onboard LEDs to provide feedback during the data collection process. these indicate when the system is measuring, and when data is being transmitted via UART.

Control Flow: The ToF sensor is first initialised, after which it goes into an idle state until a button is pressed. The 360-degree scan is carried out by the system after pressing PJ0. Data collection is stopped if PJ0 is pressed once more while the scan is running. A complete data collection is completed with each rotation and sent to the PC for real-time 3D visualisation. UART manages the data transfer, and until the scan is over, the system keeps an eye on and reacts to human input. The system can precisely record depth information using this polling and data gathering mechanism. It then transmits the data to the PC for processing and display using Python, where it is transformed for visualisation.

Detailed Description – Part B: Visualization

Once the data has been collected by the ToF sensor and has been transferred to the computer, through the UART serial communication, the visualization part is done through open3d. Open3d renders the point cloud and connects each of the points with lines. These slices (generated by different scans' points being connected) are then connected to each other to form a 3d surface. This is important because the 3D surface is not just a scatter plot of points, but a continuous structure where points are connected to form a meaningful visualization.

Application: Instructions and Expected Output

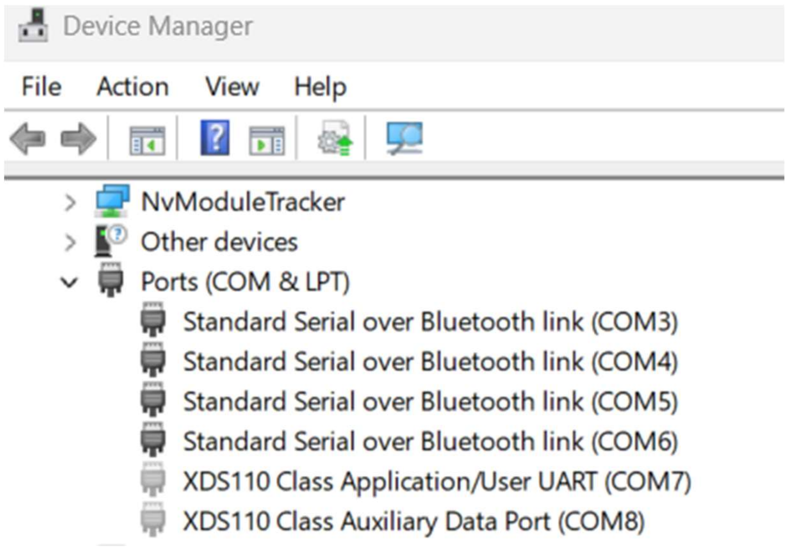
Lidar scanning technology is a useful tool for many applications since it uses laser light to measure distances and produce intricate 3D models of objects and landscapes. It provides high-resolution digital elevation models and is widely used in mapping and surveying, especially for resource management, modelling, and planning. Lidar also provides real-time environmental data for autonomous cars to assist them avoid collisions and stay safe.

Instructions

Assuming that all the software (Kiel Development Environment, any IDE for Python, Python software and the necessary Python libraries such as Open3d, NumPy and serial) has been installed and that all the correct configurations for the microcontroller have been selected, the following instructions will go through on how to use the system:

1. Connect the microcontroller to your computer and identify the COM port by going into the Device Manager. In the Device Manager, click on "View" and make sure that the hidden devices are visible by checking "Show hidden devices". click on the "Ports (COM & LPT)" section and check the COM port number listed next to 'XDS110

Class Application/User UART'. Mine was COM 7



NvModuleTracker
Other devices
Ports (COM & LPT)
Standard Serial over Bluetooth link (COM3)
Standard Serial over Bluetooth link (COM4)
Standard Serial over Bluetooth link (COM5)
Standard Serial over Bluetooth link (COM6)
XDS110 Class Application/User UART (COM7)
XDS110 Class Auxiliary Data Port (COM8)

Figure 2: screenshot of device manager.

2. Open the Python script that reads "TransferFunction" and modify line 7 to your COM #.
3. Set up the circuit according to the schematic in Figure 12 on page 12.
4. In Kiel, open the project with the given code and all the files and navigate to the 'Options for Target' tab, then select the 'C/C++' tab and set the optimization to '-O0'. Next, go to the 'Debug' tab, choose 'CMSIS-DAP Debugger' from the dropdown, click 'Settings', and confirm that 'XDS110 with CMSIS-DAP' is selected. Click 'OK' on both menus.
5. Click 'Translate' → 'Build' → 'Load' in the IDE to build and load the project onto the microcontroller.
6. Press the RESET button on the board.
7. In the Python script called "DV", change the increment if needed to match the desired x-axis step distance (in mm) between scans.
8. Run the Python script and when prompted in the terminal, write the number of scans you want to take, and press enter.
9. Press PJ0 to start a 360-degree rotation scan.
10. Depending on the number of scans needed, move the device by your desired increment and then press PJ0 again to start the scan.

11. Once all the scans are completed, open the other python script (named "3DV") and run it. When prompted in the terminal, enter the number of scans taken.

12. Your data will now be visualized, and the first window will show you all the points captured. Cross out this window to view the full generated 3d scan.

Expected Output:
The expected results from the device are shown in Figures 6 - 9. For these outputs, the x-displacement used was 300mm in the python script, however I believe that in real life, the displacement may have been a bit higher. The assigned scan location was in ITB, near room 236. Figure 3 shows a top view of this location. Then Figures 4 and 5 show real-life images of the hallway.

By comparing the figures, it can be observed that for the most part, the scan does accurately depict the hallway, however it has difficulty picking up the long end of the hallway (The L shape part where the scan starts from). Figure 10 shows a failed scan. Although the beginning of the scan and the ending of the scan are accurate, the wires of the motor came in the way of the ToF sensor around the middle of the hallway, causing the scan to fail.

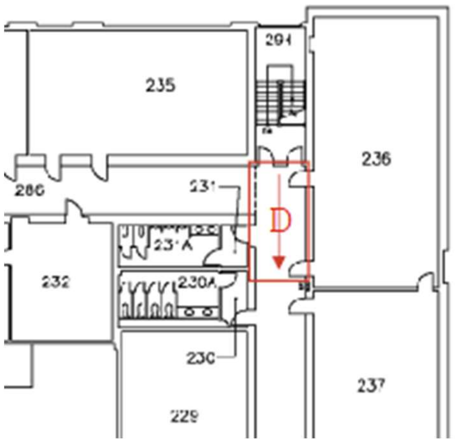


Figure 3: top view of location

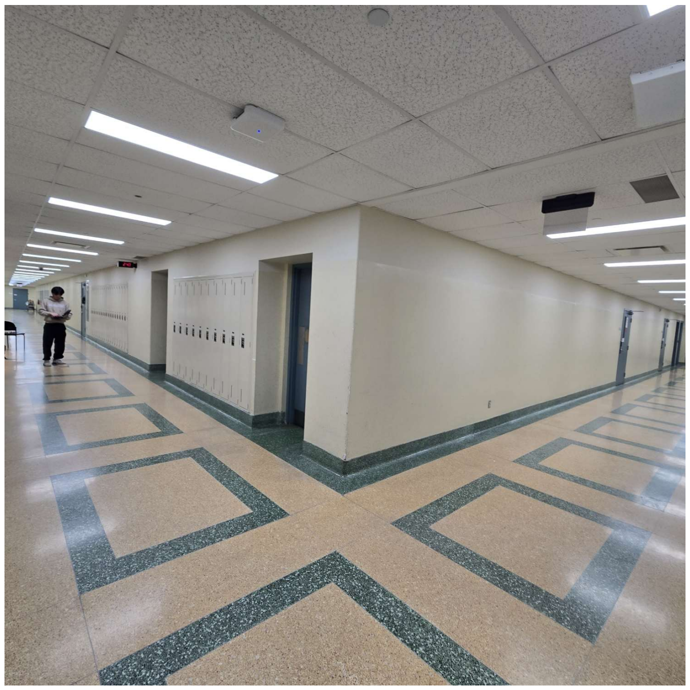


Figure 4: Full view of location

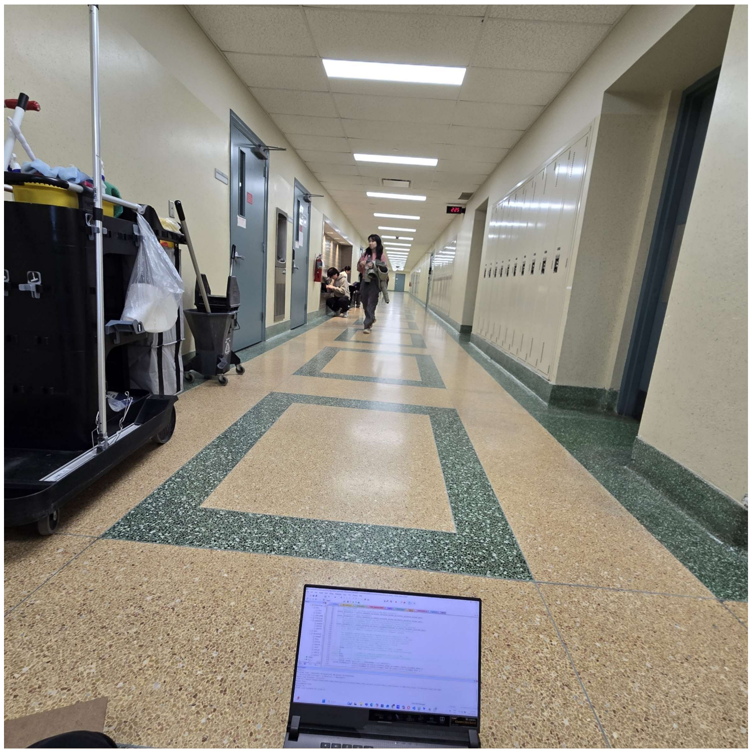


Figure 5: Hallway view of location

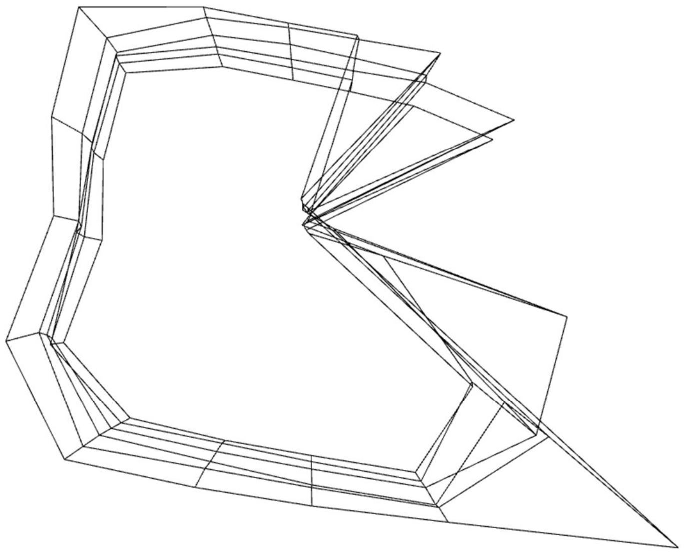


Figure 6: Scan 1 Front View.

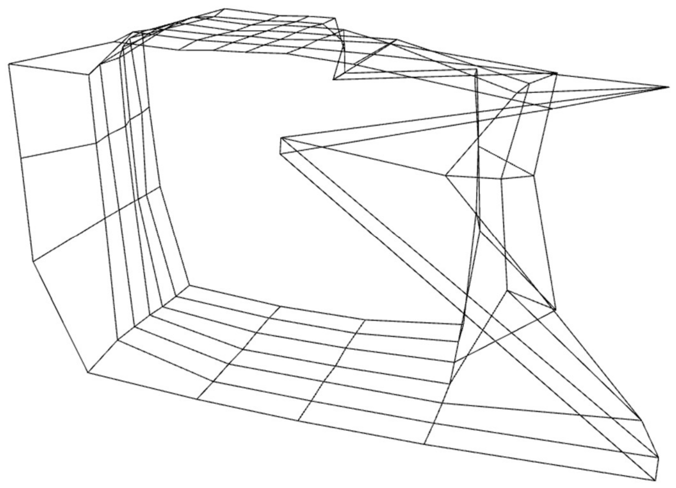


Figure 7: Scan 2 front view (most accurate) .

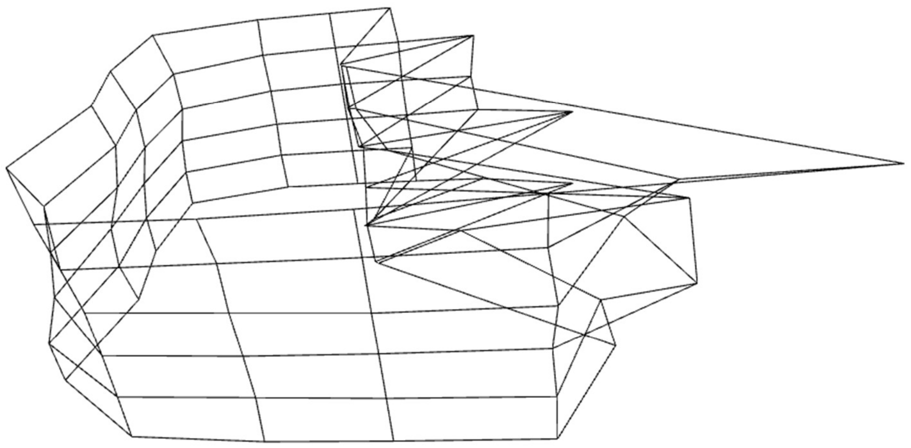


Figure 8: Scan 1 Bottom View

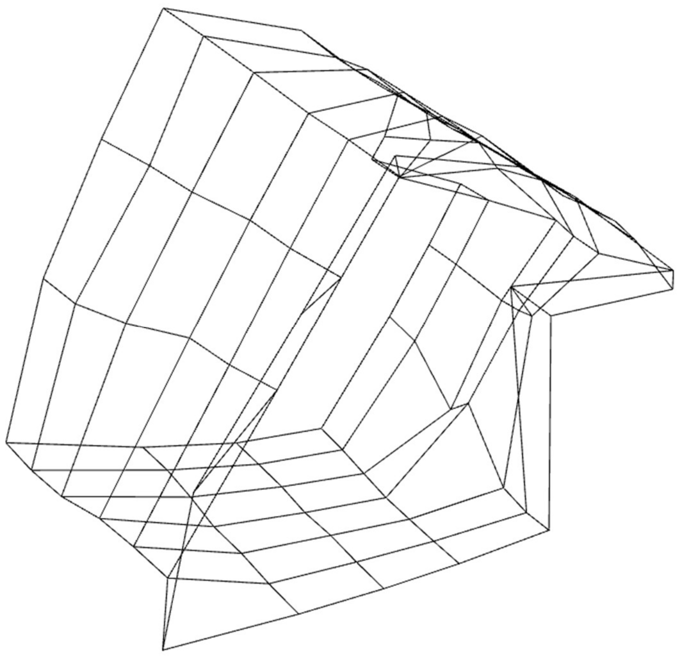


Figure 9: Scan 2 side view.

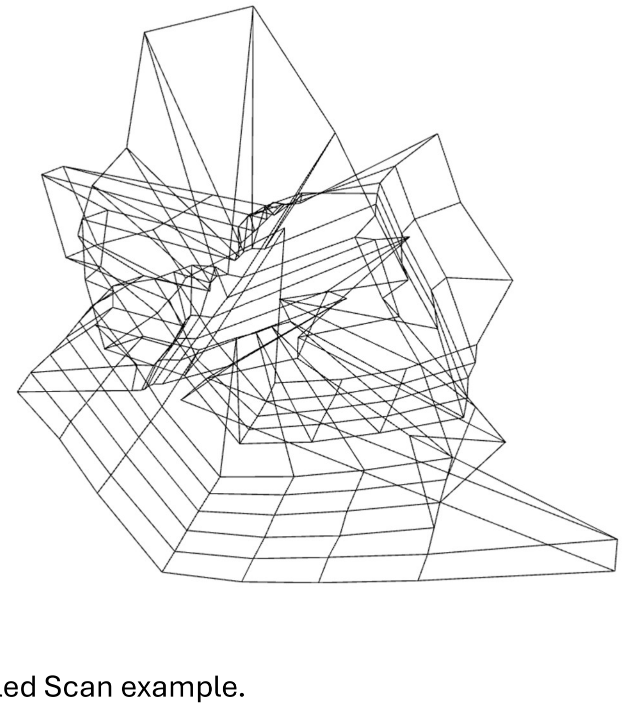


Figure 10: Failed Scan example.

Limitations

The microcontroller used in this system features an Arm Cortex-M4F with a 32-bit Floating Point Unit (FPU) capable of handling single-precision floating-point operations. While this allows for efficient trigonometric calculations, it is better for computations such as converting distance data to Cartesian coordinates to be handled by the PC for faster processing using 64-bit precision, if possible, as this is more accurate.

The maximum quantization error of the system is determined by the max reading of the ToF sensor, which has a maximum range of 4 meters and outputs values in a 16-bit format. This results in a quantization error of 0.06mm.

$Max\ Quantization\ Error = \frac{Max\ output}{2^n}$ (where n = number of adc bits, which is 16 in our case)

$$Max\ Quantization\ Error = \frac{4000mm}{2^{16}}$$

$$Max\ Quantization\ Error = 0.06mm$$

The maximum baud rate for UART communication between the microcontroller and the PC is 128 000 bits per second, verified through the device manager on my PC (check properties of the XDS110 Class Application/User UART port).

The ToF sensor uses I2C for communication, supporting speeds of up to 50Hz for range measurement and up to 400kbps in Fast mode for data transfer.

Two primary factors impact the system's speed: the time required to initialize the ToF sensor and the speed of the motor's rotation. The motor is the more major limitation. The motor's speed is the limiting factor, as it determines how quickly it can complete a full rotation. To mitigate this, various phase delays were tested, finding that the shortest delay was approximately 2ms, which allowed the motor to function at a reasonable speed without issues.

My assigned system bus speed is 22 MHz since the LSD of my student number is 3 (student number: 400514463). This was implemented into the system through the PLL.h and PLL.c files in the Kiel Project. Firstly the crystal (fXTAL) is fixed at 25MHz and the PSYSDIV is defined as 5. Then by setting N to different values and calculating the MINT value that sets the bus speed as close as possible, the bus speed was set to 22MHz. This was all done through the following equations:

$$SysClk = \frac{fVCO}{(PSYSDIV + 1)}$$

$$fVC0 = \left(\frac{\frac{fXTAL}{Q + 1}}{N + 1}\right) * \left(MINT + \left(\frac{MFRAC}{1{,}024}\right)\right)$$

Q and MFRAC are set to 0.

In the final version of the code, N is set to 6 and MINT is set to 37.

$$SysClk = \frac{\left(\frac{fXTAL}{7}\right) * 37}{6}$$

$$SysClk = 22$$

Circuit Schematic

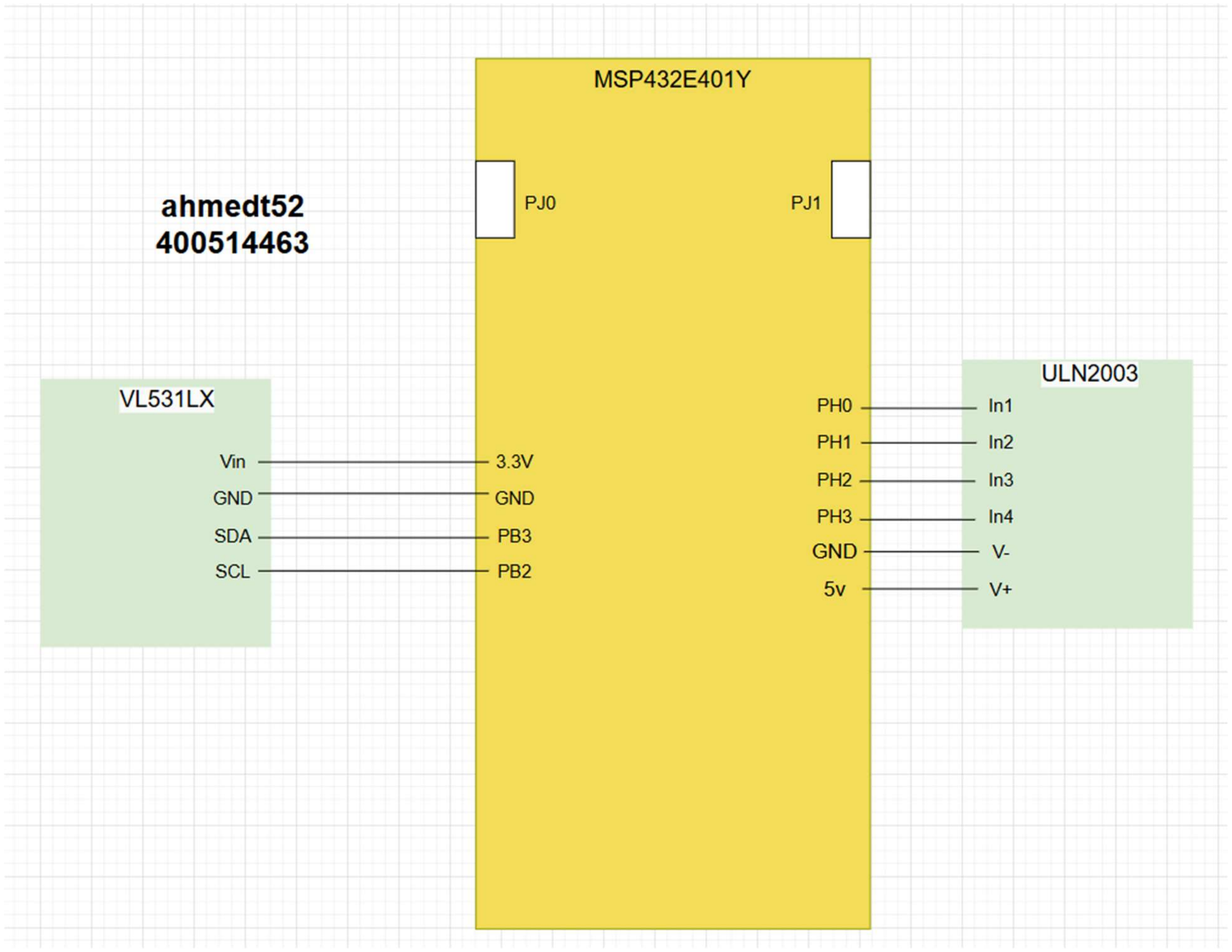


Figure 11: Circuit Schematic

Programming Logic Flowchart

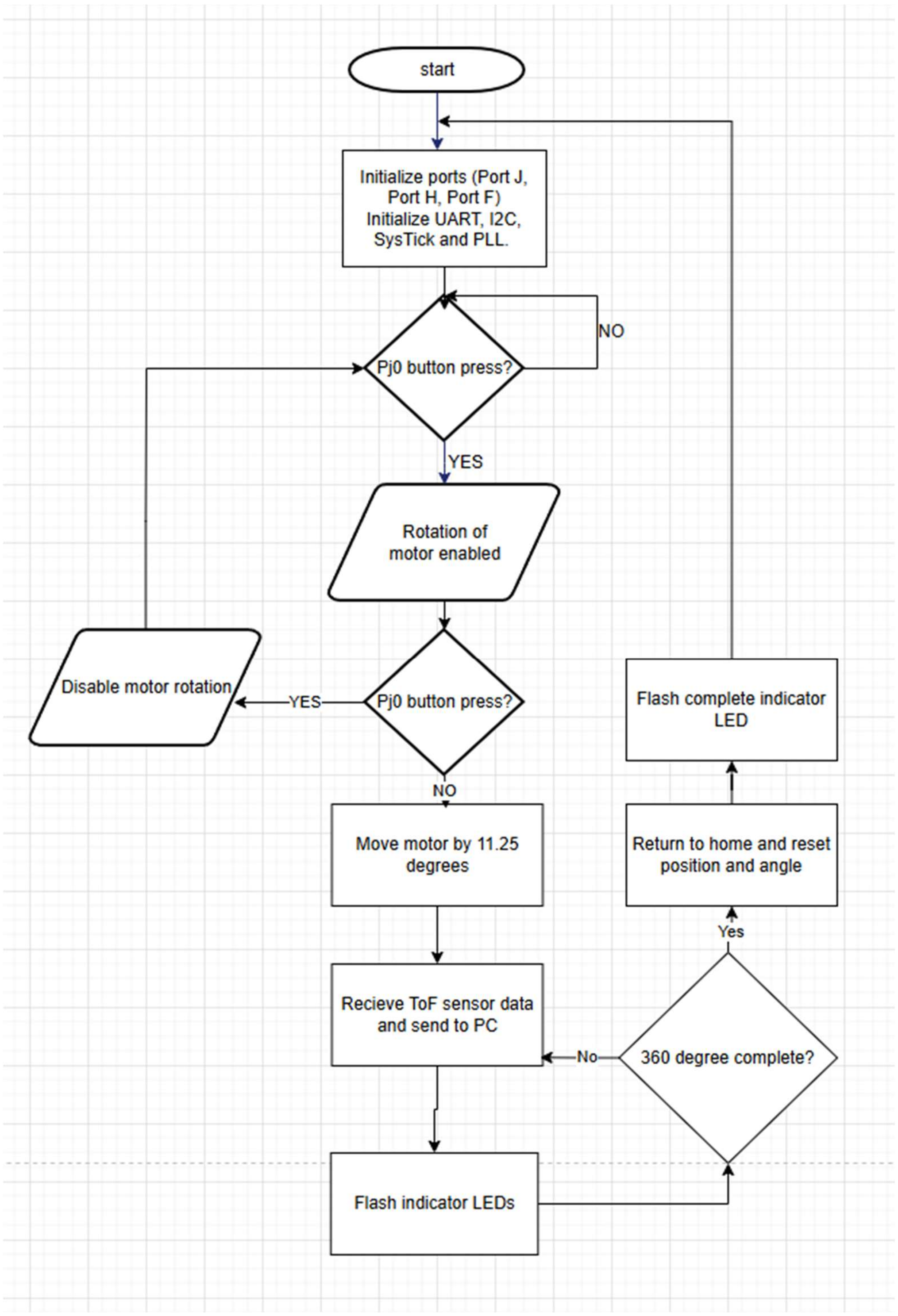

start
Initialize ports (Port J, Port H, Port F) Initialize UART, I2C, SysTick and PLL.
Pj0 button press?
NO
YES
Rotation of motor enabled
Pj0 button press?
YES
Disable motor rotation
NO
Move motor by 11.25 degrees
Recieve ToF sensor data and send to PC
Flash indicator LEDs
360 degree complete?
No
Yes
Return to home and reset position and angle
Flash complete indicator LED

Figure 12: Flowchart showing the programming logic