Comp Eng 2DX3

Final Deliverable Report

Instructors: Dr. Doyle, Dr. Haddara, Dr. Athar

Timothy Luo - L05 – luot26 – 400460844

Device Overview

Features

Texas Instrument's MSP432E401Y microcontroller

- Allows for communication between the ToF sensor and the PC
- 32-bit Cortex M4F architecture with Floating-Point Unit
- 256 KB SRAM, 6KB EEPROM, 1024 KB Flash Memory
- 24MHz Bus Speed
- 2 Onboard LEDs, one that turns on while scanning process is occurring and one that blinks every time a measurement is taken
- GPIO pins

VL53L1X time-of-flight sensor

- 940 nm invisible laser emitter
- 400cm max ranging distance
- 50Hz max ranging frequency
- 400kHz max I2C interface

28BYJ-48 Stepper Motor

- 5V supply voltage
- Has 4 phases per step, with 512 steps required for a complete 360-degree rotation
- Has 4 onboard LEDs that turn on to show the current phase

External Push Button

- A 3.3V supply voltage
- Pushing the button activates the motor for a 360-degree rotation, taking measurements every 11.25 degrees

Serial Communication

- I2C communication protocol is used to transmit data from the ToF sensor to the microcontroller
- UART communication protocol is used to transmit data from the microcontroller to the PC
- UART communication uses a 115200 BPS baud rate

3D visualization

- Python is utilized for serial communication between the microcontroller and PC
- Python is used to read data, plot points, and create a 3D visualization

General Description

This project utilizes Texas Instrument's MSP432E401Y microcontroller and the VL53L1X time-of-flight (ToF) sensor to build a LiDAR scanner that scans and plots the surrounding area. The system is started by pressing a push button to begin rotating a stepper motor. Mounted to the stepper motor is the ToF sensor that begins taking measurements once the button is pressed and the motor begins to rotate. Every 11.25 degrees, a data measurement is taken along the yz plane, and an onboard LED will flash to signify that a measurement has been taken. After a full 360-degree rotation, the motor will complete another rotation in the opposite direction to untangle the wires, not taking any scans while doing so. Then the user can take a step forward and press the button again to commence the next scan.

For every sensor reading, the data will be transmitted to the microcontroller via I2C communication. It is then sent to the user's PC through UART communication, at which point it is read by a Python script. The data is converted to cartesian coordinates and saved onto an xyz file, which is then plotted using Open3D. The result will be a 3D diagram of the area that has been scanned.

Block Diagram

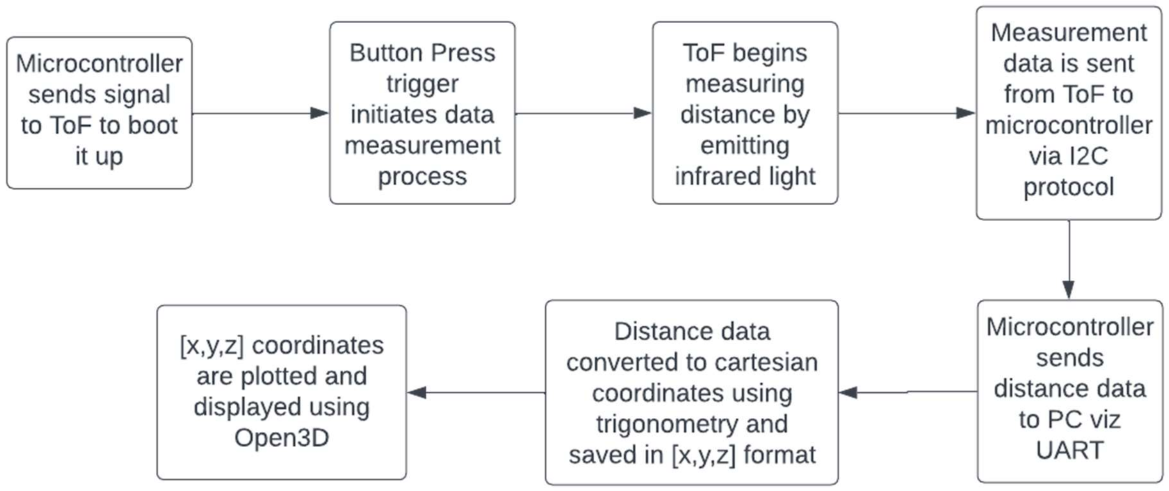

Microcontroller sends signal to ToF to boot it up
Button Press trigger initiates data measurement process
ToF begins measuring distance by emitting infrared light
Measurement data is sent from ToF to microcontroller via I2C protocol
Microcontroller sends distance data to PC viz UART
Distance data converted to cartesian coordinates using trigonometry and saved in [x,y,z] format
[x,y,z] coordinates are plotted and displayed using Open3D

Figure 1: Block Diagram of Data Transfer

Device Characteristics

Microcontroller	**Bus Speed**	24 MHz
	Serial Port	COM 4
	Baud Rate	115200 BPs
	Board Architecture	Cortex-M4F
	Memory	SRAM – 256 KB EEPROM – 6KB Flash Memory – 1024 KB
	Operating Voltage	5V
	LEDs	PN1, PF4
	I2C Serial Bus Speed	400KHz
Stepper Motor and Driver	**V+**	5V
	V-	GND
	Inputs (1-4)	PM0, PM1, PM2, PM3
Time-of-Flight Sensor (VL53L1X)	**Input Voltage**	3.3V
	GND	GND
	SDA	PB3
	SCL	PB2
Push Button	**Input Voltage**	3.3V
	GND	GND
	Input	PL0
Software	**Python Version**	3.8.4
	Libraries	serial, math, open3d, numpy

Detailed Description

Distance Measurement

In this system, distance is measured through the VL53L1X ToF sensor. This sensor uses its emitter to emit a beam of infrared light, in this case a 940nm laser. The beam will reflect off objects within the max ranging distance of 400cm and return to the sensor's receiver. To get the distance measurement, the time delay between emitting and receiving the beam is measured and the distance between the sensor and object is calculated using the equation $Distance = \frac{flight\ time}{2} \times speed\ of\ light$. Next, the sensor converts the value from analog to digital and transmits it to the microcontroller through I2C serial communication.

The ToF sensor is activated and initiates ranging through the ToF sensor boot function that is called in the Keil Program. The microcontroller then polls for a trigger from the button peripheral. Once the button is pressed, the motor begins to rotate, and this commences the distance measuring.

Inside the spin_motor function of the C code, we control both the motor rotation and the distance measurements. First, we check if the number of steps is divisible by 16. If it is, this means that the motor has rotated 11.25 degrees (360 degrees/11.25 degrees = 32 measurements, 512 steps/16 steps = 32 measurements). If the motor has rotated 11.25 degrees, the measurement LED is flashed, and a scan variable is set to take a distance measurement. Using the TOFs API functions, RangeStatus and Distance are taken. If RangeStatus does not equal 0, that means the scan failed and Distance is set equal to 9999 as a means of error handling. The two variables are then transmitted from the ToF sensor to the microcontroller via I2C and then to the PC via UART. This then repeats for each rotation of 11.25 degrees until one full rotation has been completed.

Visualization

After the data has been transmitted to the PC, the visualization process begins. After the Python program is run, it begins receiving the data by connecting to the UART COM port, in this case COM4, and opening it. An xyz file is created and this is where the data points are stored for Open3D to access. Once the button press initiates the scanning process, the PC receives a message telling it that the program will be receiving valid plotting data. Then, the program enters a loop to take the data points. In each loop iteration, firstly, the distance value is checked to see if it is equal to 9999. If not, that means the scanned values are accurate and each distance value is converted to cartesian values and then printed out and stored in [x, y, z] format. As the ToF sensor scans the yz plane of the hallway, it uses the following formulas and math library to calculate the y and z coordinates:

$$y = distance \times \cos(angle)$$

$$z = distance \times \sin(angle)$$

Angle is a variable that is incremented by 11.25 degrees after each data measurement. Next, the x coordinate is calculated by setting it equal to the variable depth. After each 360-degree rotation, depth is incremented by a constant step size, which is set to 500mm. This results in a complete 32 measurements for each step. If distance is equal to 9999, the yz values are set equal to the previously measured values to account for unsuccessful scans. The rest of the process remain the same. After the desired number of scans has been taken, the program breaks out of the loop and closes the COM port, ending data transmission. The xyz file is then saved and Open3D is used to generate the plot. The resulting image is created through a cloud point data set where each point in a scan is connected by a line and each point at the same angle is connected by a line as well. This should result in a plot that resembles the surrounding area.

Application Note

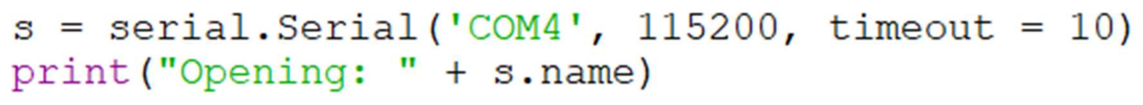
The following sections act as a manual for how to use the LiDAR scanner and the expected output. It is important to note that there are limitations to the accuracy of the scanner and human error can result in irregular scans. The following recommendations can help ensure an accurate scan.

- It is recommended that you begin scanning with the sensor facing upwards for consistent scanning results.
- The device should remain still and levelled during the scan to avoid scanning issues, as in the case of the expected output. This means that handheld scans are subject to more imprecision and not recommended.
- The ToF sensor has difficulty recognizing clear or shiny surfaces. In these cases, the distance has been modified so that the maximum ranging distance will be returned.
- In the case of scanning errors such as the motor stopping, ensure that all wires are properly connected and not twisted. Additionally, ensure that the wires do not cover the ToF sensor and do not impede the motor's rotation.

Instructions

These instructions are written under the assumption that all software necessary for the proper execution of this scanner are installed and configured properly. For proper function, Keil Development Environment and Python IDLE (versions 3.6-3.9 only) must be installed and TI's MSP432E401Y microcontroller must also be properly setup in Keil.

1) The user must plug the microcontroller into the computer. Then, open the Device Manager on their computer. Inside the device manager menu, scroll down until the 'Ports (COM & LPT)' section is found. Expand this section and record the number value in the following line: XDS110 Class Application/User UART (COM#). This number is the COM port of the computer.
2) Assemble the hardware based on the pins specified in the device characteristics table and the circuit schematic.
3) Open the python script and scroll past the functions until the following line is found. Then edit the COM value to match the COM port of the computer.



```
s = serial.Serial('COM4', 115200, timeout = 10)
print("Opening: " + s.name)
```

Figure 2: Line of python script where the COM value is edited

After this, the serial communication should be ready.

4) Inside the python script, edit the scans variable to the number of scans you would like to take. In the Keil code, update the 'scan_num' variable to the same value.
5) Open the Keil project and click 'Translate' – 'Build' – 'Load' in that specific order.
6) Click the onboard reset button the flash the project onto the microcontroller.
7) Run the python script and click enter on your keyboard when prompted.
8) Click the button to rotate the stepper motor and repeat until the desired number of scans have been completed.
9) The Open3D scan should appear on the computer after all scans have been completed.

Expected Output

To test the LiDAR scanner, a hallway on the 1st floor of ETB was scanned and modelled. While the hallway was mainly just a simple, straight hallway, at the back end of the hall was wider opening that exceeded the max distance of the ToF sensor. This was necessary for the LiDAR scanner to account for in order to generate an accurate scanner.

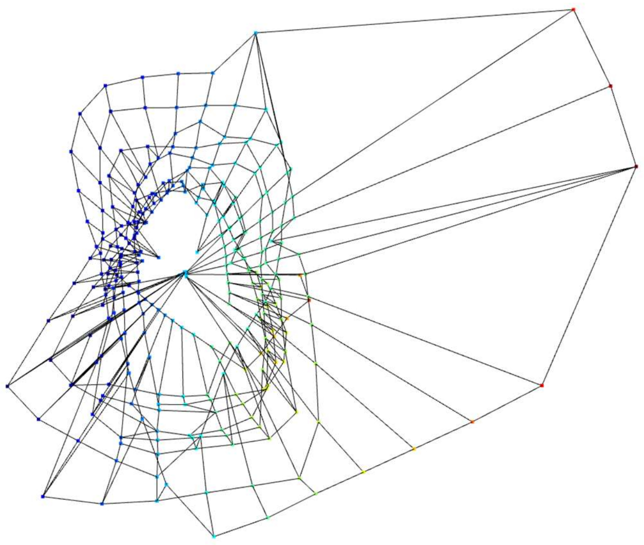


Figure 3: Scan of the Hallway

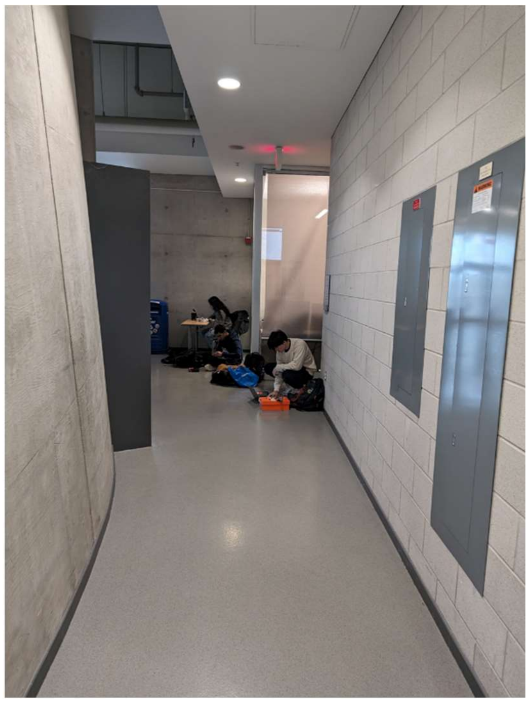


Figure 4: Picture of the Actual Hallway

As seen in the scan, the system can accurately recognize the extended opening, which is where the scan began. After passing this starting point, the hallway becomes narrower which is also captured in the scan, as the rest of the hallway remains a uniform width and height. Considering this, the scan is relatively successful. However, there are evidently some issues with the scan. For example, in the third step, the hallway scan is tilted about 20 degrees which results in a distinct irregularity compared to the actual hallway. This issue can be explained by human error, as the scanner was slightly tilted while taking this scan which resulted in this angled result. To address these issues, the scanner should be placed on a flat surface and remain completely level during future scans.

Limitations

1. The TI's MSP432E401Y utilizes the Cortex-M4F 32-bit processor and includes a Floating-Point Unit (FPU). As a result, all calculations and numbers are limited to the 32 bits that are available. This is notable for the LiDAR scanner as the data plotting process utilizes trigonometric functions to convert from polar to cartesian coordinates. Since calculations involving these functions often end up with many decimal points, the microcontroller limits these numbers to the 32 bits available.
2. To calculate the maximum quantization error of a system, we use the formula $Max\ Quantization\ Error = \frac{Maximum\ Reading\ Distance}{2^{\#\ of\ ADC\ bits}}$. Given we know the max reading of the ToF sensor is 4000mm and the ToF stores values in 16 bits format, we can calculate that $Max\ Quantization\ Error = \frac{4000mm}{2^{16}} = 0.0610$.
3. The maximum standard serial communication rate that could be implemented with the PC is 128000 bps. This value was verified by opening the device manager and finding the XDS110 Class Application/User UART port and then confirming the BPS information of this port. However, this project implemented a communication rate of 115200 bps.
4. The microcontroller and ToF module communicate through I2C. For this project, the I2C transmission speed was 400kHz. As the ToF sensor has a maximum transmission speed of 50Hz, the I2C transmission is not considered a significant limitation on the design.
5. The primary limitation on system speed is the speed of the stepper motor and the speed that the ToF sensor can transmit data. As stated earlier, the ToF has a max transmission speed of 50Hz. Since the I2C transmission speed is much higher at 400kHz, much more data could be sent to the microcontroller if the ToF has a faster transmission speed. To address this, you could add multiple ToF sensors so they can take multiple measurements in the same amount of time, effectively speeding up the data transmission. For the stepper motor speed, a delay needed to be placed between phases or else the motor would not rotate. The shortest possible delay was 2ms, as when I decreased it past this point, the motor stopped rotating. This resulted in longer measurement times and an overall slower system.
6. To configure the system to the assigned Bus Speed, I had to go into the PLL.h file inside the Keil project and modify the PSYSDIV value to the corresponding number for my bus speed. In my case, I needed to achieve a 24MHz bus speed, so PSYSDIV was set to 19. Next, I needed to edit the SysTick.c file to change the wait times to match my new bus speed. For SysTick_Wait10ms, the wait time was set to 240000. For SysTick_Wait10us,

the wait time was set to 240. For SysTick_Wait1us, the wait time was set to 24. We can calculate the time it takes for 1 clock cycle at 24MHz with the equation $T = \frac{1}{f} = \frac{1}{24\times10^6} = 4.167 \times 10^{-8}$s. This means that to get 10ms, we can calculate $0.01\text{s} = (4.167 \times 10^{-8})x$, where x is the wait time value to achieve 10ms. This means for SysTick_Wait10ms, we can conclude that $x = 240000$. These same calculations were used to solve for 10us and 1us, which as previously stated, resulted in 240 and 24.

Circuit Schematic

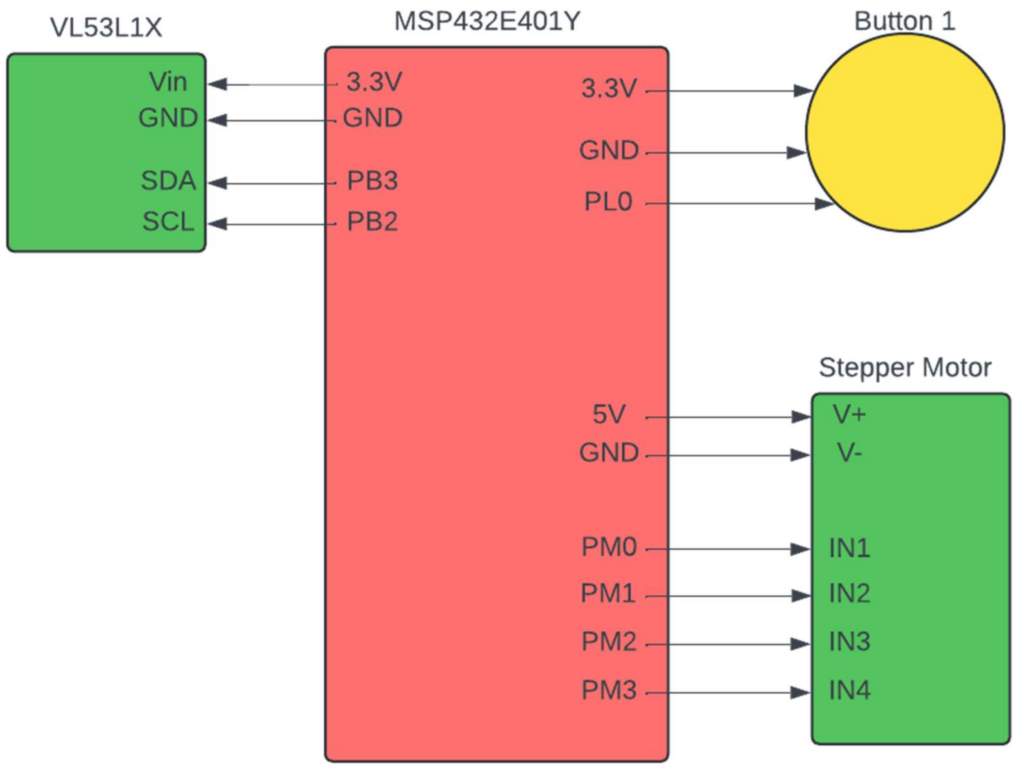


Figure 5: Circuit Schematic

Flowchart

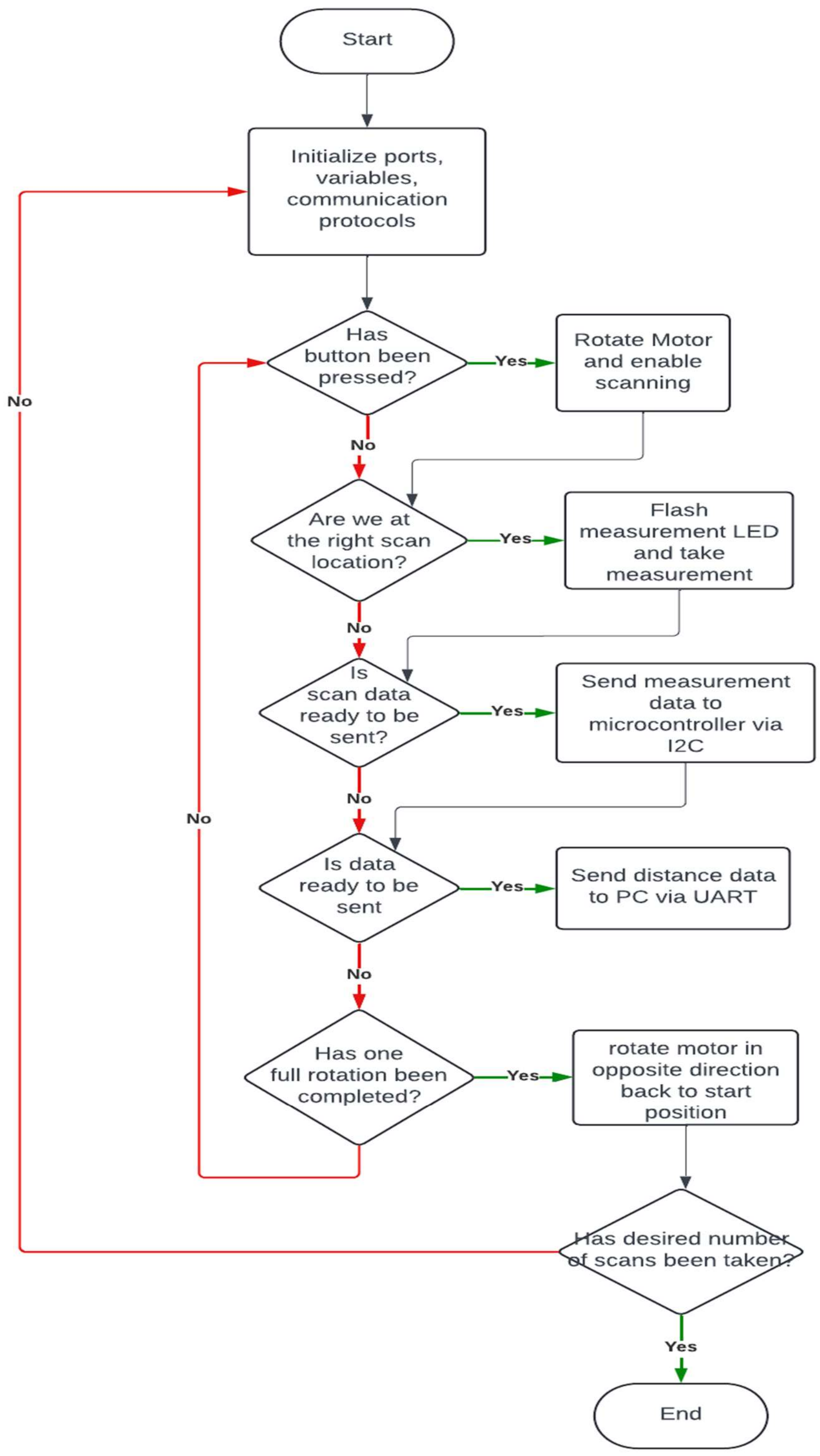

Start
Initialize ports, variables, communication protocols
Has button been pressed?
Yes
Rotate Motor and enable scanning
No
Are we at the right scan location?
Yes
Flash measurement LED and take measurement
No
Is scan data ready to be sent?
Yes
Send measurement data to microcontroller via I2C
No
Is data ready to be sent
Yes
Send distance data to PC via UART
No
Has one full rotation been completed?
Yes
rotate motor in opposite direction back to start position
No
Has desired number of scans been taken?
No
Yes
End

Figure 6: Programming Logic Flowchart