2DX4: Microprocessor Systems Project

Final Project

Instructors: Dr. Bruce, Dr. Haddara, Dr. Hranilovic, Dr. Shirani

Makeish Raveendran – L01 – raveenm - 400230606

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 **[Makeish Raveendran, raveenm, 400230606]**

Google Drive Links

Final Project Demonstration Video

<https://drive.google.com/file/d/13fN4tuz6u6VZWeh1sQHSMHCXnW_sn4TI/view?usp=sharing>

Answering Question Video

<https://drive.google.com/file/d/13iFHdAwgG-iZ4g6yiPlL2QwHjaZwXiCT/view?usp=sharing>

Circuit Setup and ScreenShot of Data

<https://drive.google.com/file/d/13h4d7k76YoojJZawq97d6RhqjeFkTb76/view?usp=sharing>

Circuit Setup

<https://drive.google.com/file/d/13ftkLqSu4ai0sitfw0Q3bt0XJZttSYXr/view?usp=sharing>

Screenshot of Data

<https://drive.google.com/file/d/1p5NU7p_T3KOQ-PeD6PBommmGXKs7Vdvo/view?usp=sharing>

Device Overview

Features

* The project re-creates a 3D representation of the area around one.
* It measures the distance in the y-z plane, by controlling a motor that rotates 360 degrees with a horizontal displacement of 10 cm for every rotation.
* The microcontroller used is the MSP432E401Y.
* The Time-of-Flight (ToF) sensor used is the VL53L1X.
* The motor used is a stepper motor.

Hardware

* The bus speed is 12 MHz.
* The voltage used for the stepper motor and ToF sensor are 5V. The voltage used for the two switches is 3.3V.
* The data was measured every 11.25° for a total of 32 data points per rotation.
* The total cost of the equipment totalled $ 179.99 without tax.
* The language used to control the microcontroller is C. The language used to graphically display the area is Python.
* The serial communication utilized PySerial and read 3 bytes per data transfer.
* The UART port for this project was COM4 and used a baud rate of 115.2 kbps. The data bits are set to 8 and the stop bit is set to 1.

General Description

The project primarily utilizes a stepper motor, ToF sensor and a microcontroller to create a functioning system that is able to scan the area around itself and process the information to graphically create a 3D representation of itself. The ToF sensor will measure distance in the y-z plane and return a value in millimeters. The stepper motor will rotate the ToF sensor 360°, when the start switch is pressed. The push buttons start and stop the system when either is pressed. The onboard LED PN1 (D1) will blink when the ToF sensor scans the area. The microcontroller allows the user to configure the ports to input and output, to receive and control data. This is done through I2C communication.

The ToF sensor measures distance by emitting pulses of infrared light that determines the distance based on the amount of time before the pulse is detected. The ToF sensor measures a distance every 11.25° and will transmit that information to the python shell through simplex serial communication. This is done through using the UART port COM4 for this specific computer and transmits 3 hexadecimal characters, as the program reads 3 bytes. These values are then processed into y and z values. The x values are manually created through displacements of 100 mm. The x, y and z lists of values are then able to be graphically presented as a 3D scatter plot.

Block Diagram

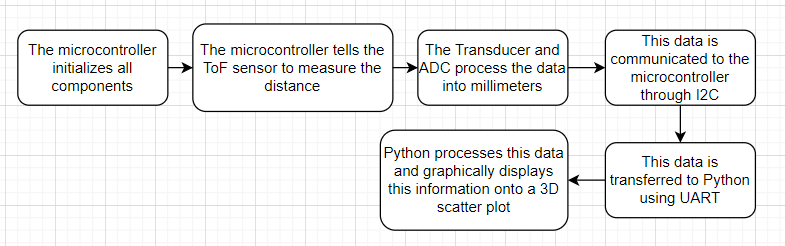


Figure 1: Block Diagram of the system.

Image of Setup

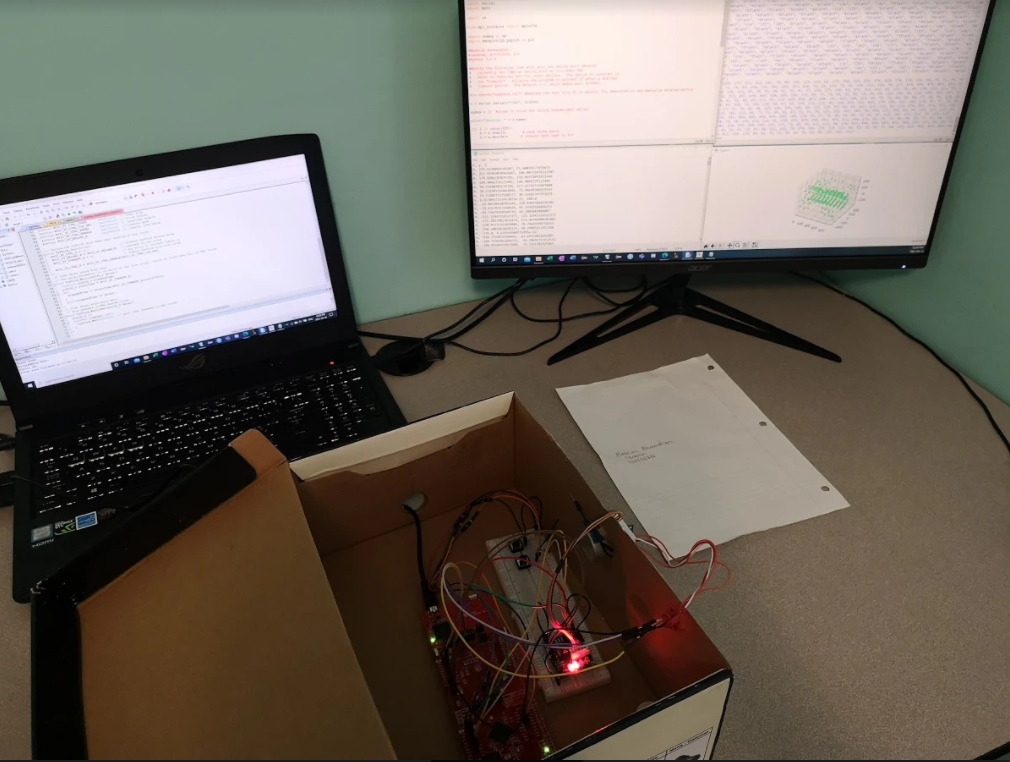


Figure 2: Final Project setup with data on PC screen.

Device Characteristics

Table 1: The specifications of the project.

| Devices | Details | |
| --- | --- | --- |
| Device Component | Microcontroller |
| VL531X Time-of-Flight Sensor | VIN | 5V |
| GND | GND |
| SDA | PB3 |
| SCL | PB2 |
| Stepper Motor | IN1 | PK0 |
| IN2 | PK1 |
| IN3 | PK2 |
| IN4 | PK3 |
| -ve | Ground |
| +ve | 5V |
| Onboard LED D1 | PN1 | |
| Bus Frequency | 12 MHz | |
| Serial Port | COM4 | |
| Baud Rate | 115.2 kbps | |
| Python Version | 3.8.9 | |

Detailed Description

Distance Measurement

The final project acquires a distance from the time-of-flight (ToF) sensor that has been rotated 360° and processes that data into a 3D area. The project is composed of the physical components as detailed in Table 1 and the keil and python code that controls and processes data. The steps can be found in the logic flow chart (refer to Figure 6 and 7). The project first initializes all necessary ports and functions. The VL53L1X ToF sensor can measure up to a maximum distance of 4000 mm and uses a 940 nm infrared laser to determine the distance in the y-z plane. This is done by the ToF calculating the time it takes for the infrared laser to travel the unknown distance and back to the ToF sensor.

The distance measured is then transferred to the microcontroller through I2C protocol and stores the value as a hexadecimal value into an array. This data is then sent to the python program through UART at a baud rate of 115.2 kbps. The values stored in the new array within python are in string form and are converted into decimal integer form. This is done through executing the command “int(“hexadecimal value in string form”,16)”. This data is the y-z vector data and will be processed into y and z values with their respective angles.

The final project utilizes a stepper motor that rotates 360° each time the start push button is pressed. Also, each rotation takes a total of 32 scans at 11.25°. This is important when processing the y-z vector values, because we will utilize python’s math library and use the functions sine and cosine. For the following formula the y-z values will be known as V. The following formula is used to calculate the y values and z values.

The y and z values are stored into their respective y and z lists. The displacement can be determined manually, as the demonstration utilized 100 mm displacements for every rotation. The x values are stored in their respective x lists and every 32 values the x value is incremented by 100 mm. For the demonstration, 10 scans were used and therefore the x values will start at 0 mm and will increase till 900 mm. At the same time these values are stored into their respective lists, they are also stored into a separate text file in the form of “x, y, z”.

Visualization

The visualization of the surrounding area was done through a windows 10 and Intel i7 7th general processor. The software used to graphically represent the surrounding area was python 3.8.9.

The 3D representation was done through utilizing python libraries such as: math, mplot3d, numpy and matplotlib.pyplot. These libraries were used to create a 3D figure and plot the data into a 3D scatter plot in the form of x,y and z values. The data received during demonstration can be seen visually in Figure 3. This data was saved as a png file and the raw data was saved as a text file.

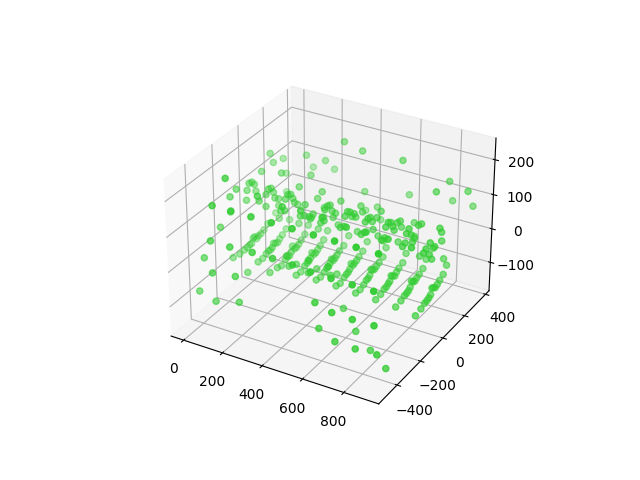


Figure 3: 3D representation of data.

The 3D representation looks a little curved near the corners and seems to have a larger radius on the ends. Overall, the 3D represented data is not accurate to the real space. The values seem to be too small as the dimensions of the x,y,z are less then the expected dimensions. The displayed dimensions state that x increases to 900 mm, y increases to 800 mm and z increases to 400 mm. The space that was supposed to be recreated, has dimensions that exceed 1000 mm. The space that was attempted to be recreated can be seen in Figure 4.

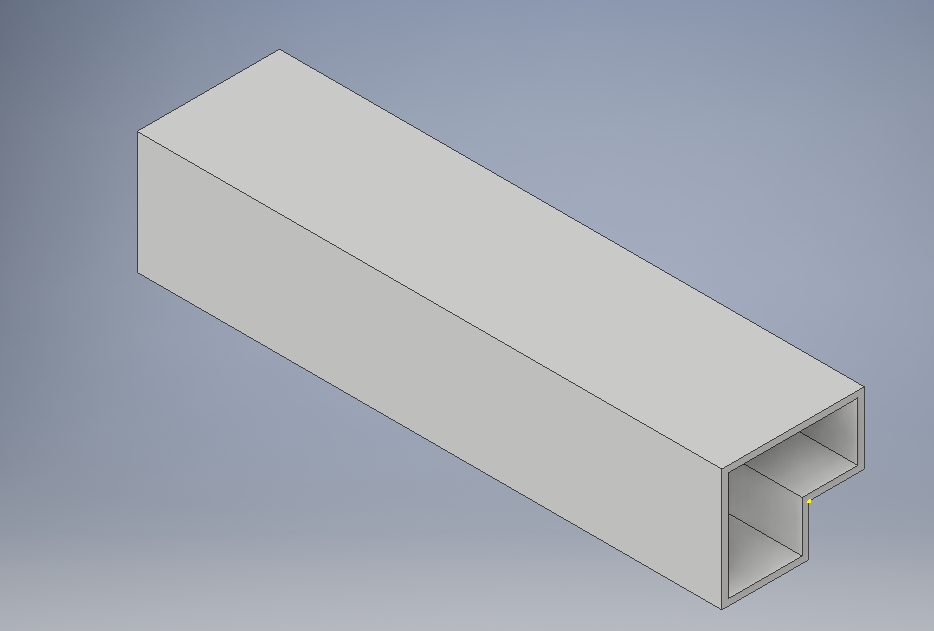


Figure 4: Rough diagram of the room.

\*The room had the project demonstrated on a table.

Application Example

The final project acquires a signal and transmits that to the microcontroller through I2C and then transmits the information to python through UART. Then python processes that data and plots the data onto a 3D scatter plot. The data measured will be in the y-z plane and the x values are manually determined through the code by displacing the project 100 mm to the left after each 360° rotation.

Steps used to setup Python and Keil

1. Download Keil and python 3.8.9. Ensuring that python 3.8.9 has been added to “Path” when the executable is run.
2. After keil has been downloaded, obtain the keil program code and create a project with the pll.h, systick.c, UART.h and the ToF sensor files. Save the obtained code into a main.c file.
3. After python has been downloaded, open the command line and install pyserial using “pip install pyserial”. Then type the command “python -m serial.tools.list\_ports -v” and check to see which port is used for UART. Import the pyserial library “import serial”, create the serial object using “s = serial.Serial(‘the port from earlier’, 115200).
4. Obtain python code and save it within a .py file. Ensure the serial port is used for UART.

Steps used to setup microcontroller

1. Obtain the necessary physical components to assemble the project. The components needed are MSP432E401Y microcontroller, the VL531X Time-of-Flight sensor and the stepper motor. Then obtain male to male wires, female to female wires, 2 push buttons and 2 resistors with a 10kOhm resistance value.
2. Using Figure 5 the circuit schematic, assemble the project.
3. Connect the micro usb to the microcontroller and plug it into the usb port in your PC.
4. Using Keil, translate, build and then load the code onto the microcontroller. If there is an error, ensure the microcontroller is plugged in and the debug settings are correct.

Steps to demonstrate

1. Attach the Time-of-Flight sensor onto the stepper motor.
2. Run the python code from earlier.
3. Push the start push button PM0 and the stepper motor should rotate 360°. Every 11.25° the LED onboard D1 pin PN1 should blink and the python shell should display hexadecimal values.
4. If there is an error on the python shell, ensure the UART port is correct.
5. After waiting a single rotation displace the project 10 cm or 100 mm to the left
6. Push the start button 9 more times after each rotation has finished and displace the project 10 cm or 100 mm to the left after each rotation.
7. The values will be displayed onto the python shell and the 3D scatter plot will appear.

Limitations

1)

The microcontroller stores the values measured as hexadecimal values. Therefore, the smallest measurement is 1 mm. This does not take in decimal values and thus is not accurate. For the final project I processed the hexadecimal values using python and did not test the limitations on through the microcontroller. Given that the Time-of-Flight sensor returned values within hexadecimal form, values used with trigonometric functions would be rounded to the nearest non-decimal digit.

2)

The maximum quantization error for the Time-of-Flight sensor can be calculated by the formula:

Qmax = ADC Resolution = .

The maximum distance measured is 4000 mm and the Time-of-Flight sensor when using I2C protocol is 16 bit, thus we get 15.625.

3)

The baud rate used in this final project is 115.2 kbps. The maximum standard serial communication rate that can be implemented is 128 kbps. This was checked by going to the device manager and viewing the communication port located there labelled XDS110 (COM4). By checking the properties of this port, it can be seen that the maximum communication rate is 128 kbps.

4)

The communication between the microcontroller and the Time-of-Flight sensor uses I2C protocol. The speed used between the microcontroller and the Time-of-Flight sensor is up to 400 kHz and is a maximum transfer rate of 400 kbps.

There were other limitations during the demonstration that could result in slight variations in results. Primarily, the setup that attaches the Time-of-Flight sensor to the stepper motor can cause the wires to tangle when rotating. This results in the wires falling off during demonstration and potential misreadings where the Time-of-Flight sensor detects and measures the distance of the hanging wire.

Circuit Schematic

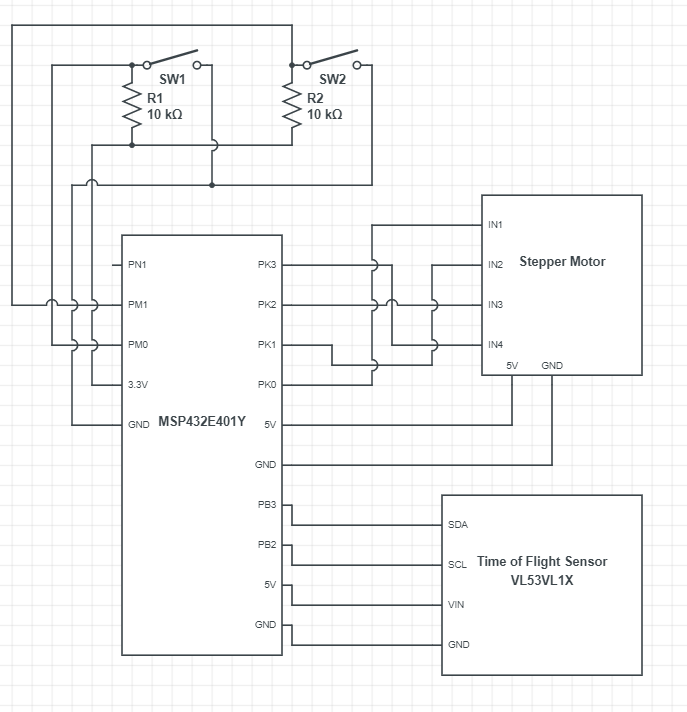


Figure 5: Circuit schematic of the final project.

\*PN1 represents LED D1

Logic Flowcharts

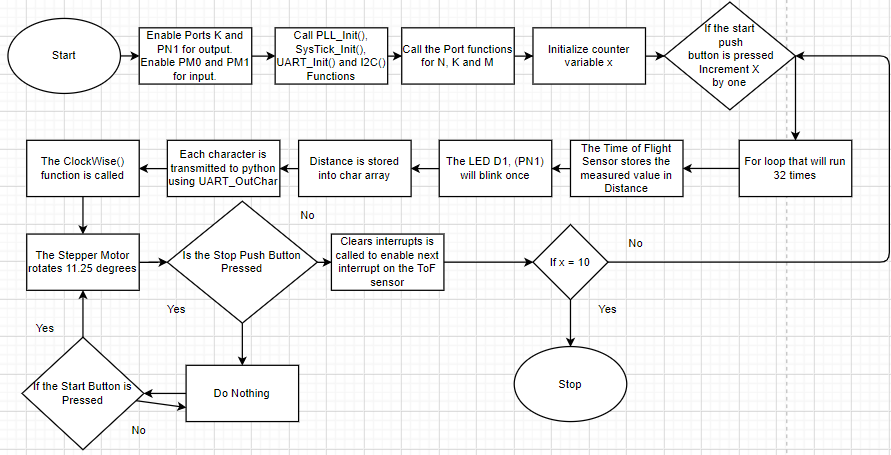


Figure 6: The flow chart for the microcontroller.

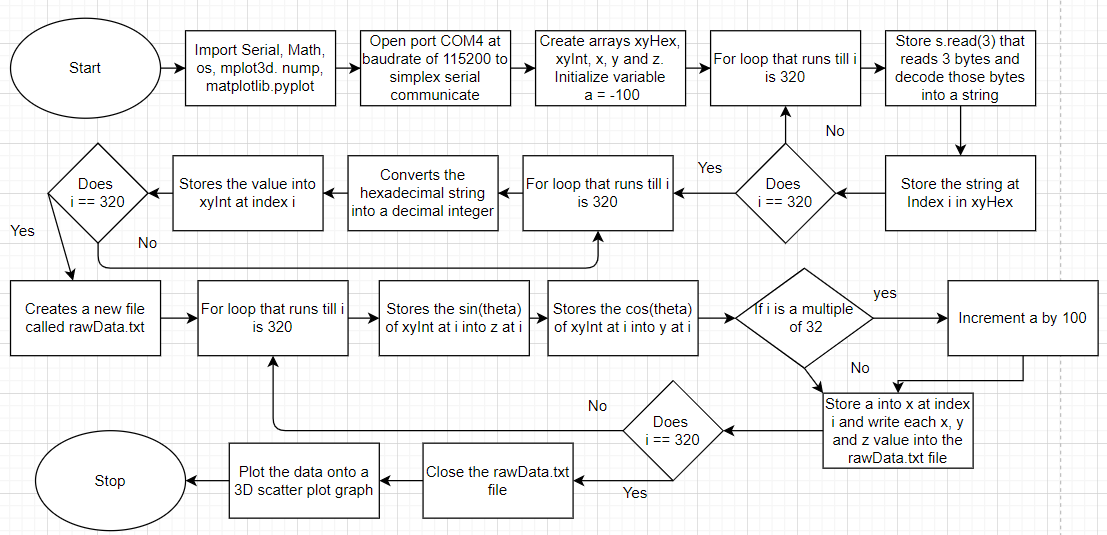


Figure 7: The flow chart for the python program.