2021-2022 2DX3 Project Specification – Observe, Reason, Act: Spatial Mapping using Time-of-Flight

Device Overview

There are many systems available to acquire data, however, equipment such as Commercial Light Detection and Ranging (LIDAR) are found to be expensive and bulky [1]. Fortunately, there are applications less expensive and smaller capable of acquiring the same set of data. The embedded spatial measurement system using a time-of-flight sensor can easily acquire the necessary information of any space.

Features:

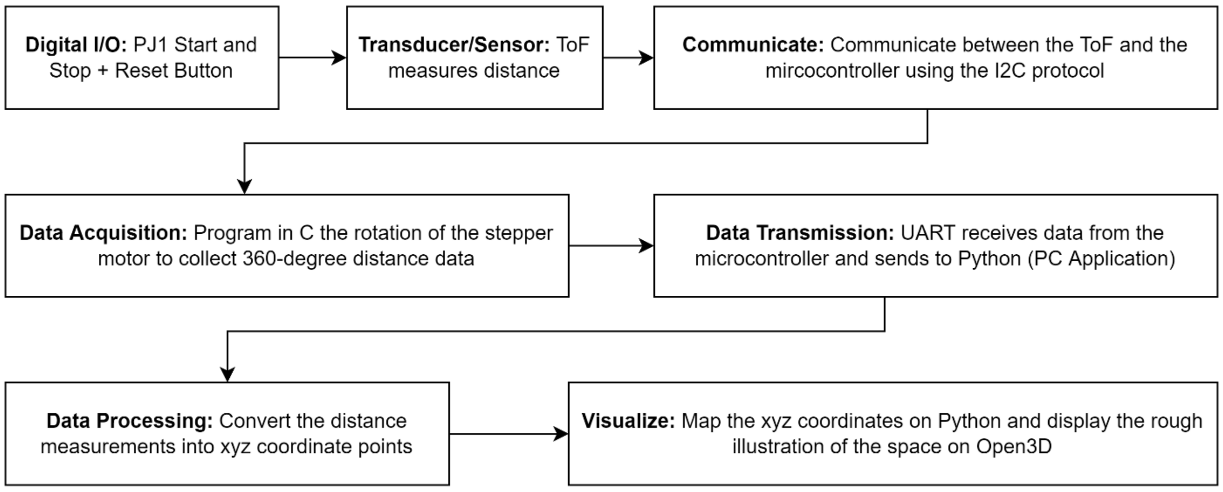
- MSP432E401Y Microcontroller – A microcomputer operating at 120MHz used to process, store, and execute information to perform programmed tasks. 32-bit Cortex M4F
- VL53L1X Time of Flight Sensor (ToF) – A device used to measure the distance of nearby objects
- Stepper Motor & Driver– A rotary device mechanism with parameters that can be controlled such as speed, angle, and direction
- Time of Flight Stepper Motor Mount – 3D printed mount to attach the ToF sensor to the stepper motor
- Onboard Microcontroller Push Buttons (Reset & PJ1) – Push buttons built onto the MSP432E401Y microcontroller to reset, start, and stop the device
- Keil Microcontroller Project Software in the C programming language – Programming software to code instructions for the MSP432E401Y Microcontroller to follow.
- AD2 + Waveform Software – A circuit device analyzer with the software to display the results of the measured frequency and bus speed of the system
- Python IDLE + Open3D – A PC Application to extract data from the ToF with the use of serial communication (UART & Pyserial). Open3D to graphically map the distance coordinates and display on the PC
- Universal Asynchronous Receiver/Transmitter (UART) communication – A communication protocol that the MSP432E401Y Microcontroller uses to send and receive date from 2 devices such as the ToF and a PC
- 60MHz Bus Speed
- ADC Sampling Rate: $f_{\text{sample}} >= f_{\text{signal}}$
- 3.3V Operating Voltage
- 115200 Baud Rate (bps)
- LED D2 Status Tracker
- Approximate cost of $200 CAD (AD2 not included)

General Description:

A student designed 3D space scanner. This device/product is an embedded spatial measurement system that gathers data from its nearby surroundings using a time-of-flight (ToF) sensor to gather data from any given space and a stepper motor to provide a 360-degree measurement of distance within a single vertical geometric plane. The data is then analyzed using a PC application and graphically mapped and displayed.

Digital I/O are used to start/stop the system. Once the system begins, data can begin to be collected by the ToF sensor. While the sensor acquires data, it is being rotated 360-degrees by the stepper motor, collecting distance measurements at every 45-degree angle. The angle can be changed for more precision. Once data has been allocated, the data can now be sent and processed using Python by serial communication. UART is an inter-integrated circuit protocol that sends received data from the microcontroller to Python via Pyserial. A Python program is used to read the encoded data and writes it to a text file for further analysis. Now that data had been stored to the PC, a program can extract the distance from the text file, calculate the yz coordinates, manually generate the x coordinate and save theses mapping points into a xyz file allowing the Open 3D to graphically map these coordinates and display a rough illustration of the space.

Block Diagram:


Digital I/O: PJ1 Start and Stop + Reset Button
Transducer/Sensor: ToF measures distance
Communicate: Communicate between the ToF and the mircocontroller using the I2C protocol
Data Acquisition: Program in C the rotation of the stepper motor to collect 360-degree distance data
Data Transmission: UART receives data from the microcontroller and sends to Python (PC Application)
Data Processing: Convert the distance measurements into xyz coordinate points
Visualize: Map the xyz coordinates on Python and display the rough illustration of the space on Open3D

The 3D Space Scanner Process

Product Image & PC Display:

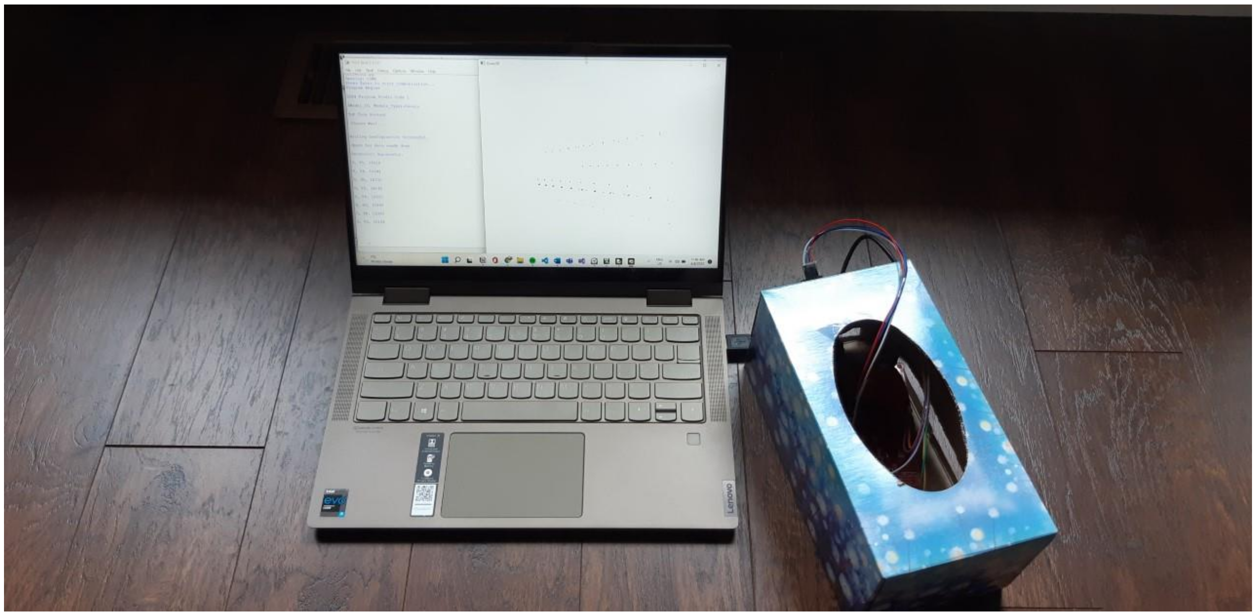


Figure 1 - Complete Product Image

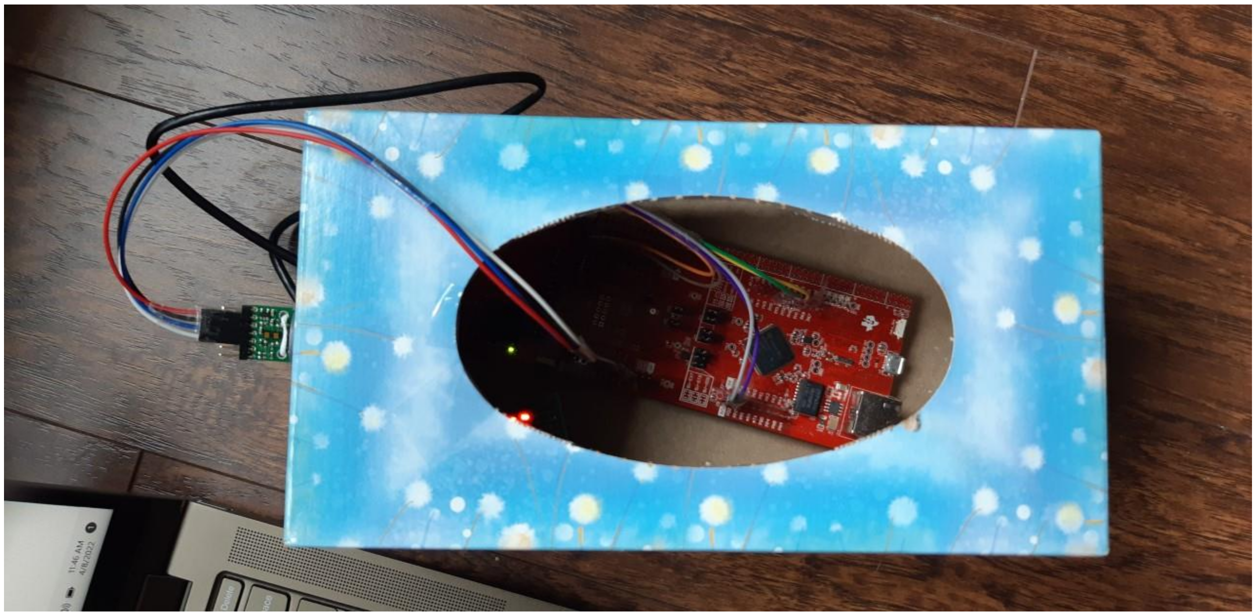
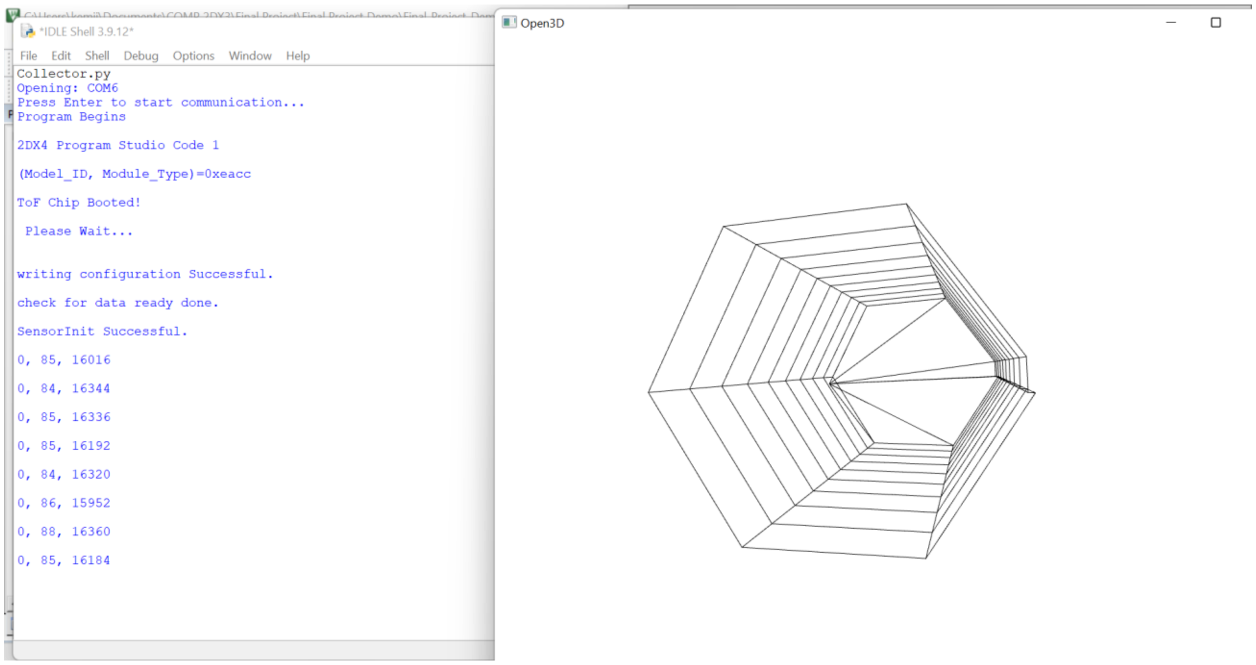


Figure 2 - 3D Space Scanner



```
Collector.py
Opening: COM6
Press Enter to start communication...
Program Begins

2DX4 Program Studio Code 1

(Model_ID, Module_Type)=0xeacc

ToF Chip Booted!

 Please Wait...


writing configuration Successful.

check for data ready done.

SensorInit Successful.

0, 85, 16016

0, 84, 16344

0, 85, 16336

0, 85, 16192

0, 84, 16320

0, 86, 15952

0, 88, 16360

0, 85, 16184
```

Figure 3 - PC Display

Device Characteristics Table

Device Characteristic	***Specified Value***
Bus Speed	60MHz
Digital I/O LED Status	PN0 (LED D2 on microcontroller)
Microcontroller Pins	PH0-PH3, PB2-PB3, 3.3V and GND
Stepper Motor Driver Pins	IN1-IN4, + and -
Operating Voltage	3.3 V
Baud Rate	115200 bps
Python Software	v.3.6-3.9 (v3.10 doesn't work with Open3D)
Python Libraries & Packages	Pyserial, Open 3D

Detailed Description

Distance Measurement:

To collect distance data, a ToF sensor is a device required to achieve this. The VL53L1X ToF sensor is a laser-ranging sensor [2]. How the ToF works is that it measures how long it takes for emitted pulses of light to reach the nearest object and be reflected to the detector. It emits pulses of infrared laser light and signal reflects the nearest object in its path. Reflection is collected by the ToF receiver and there is a time delay between the emitted and received periodic signal that is measured [3]. Real applications for the ToF sensor include drones, service robots, vending machines, and vacuum cleaners [2].

The 3-step process collecting and mapping the distance measurement includes acquisition, transmission, and conversion. The first step, acquisition is where data is collected by the ToF sensor. The components of this first stage are the microcontroller, the ToF sensor, the I2C protocol as well as digital I/O buttons to start the system. The Inter-Integrated Circuit (I2C) protocol is used to help 2 external devices communicate with and transfer data between each other. These 2 devices are the microcontroller and the ToF. As the ToF collects data, the microcontroller is receiving signals from the ToF. To begin data collection, an input is required which is represented by the PJ1 push button. This button starts and stops the system using interrupts. Once the microcontroller receives the 'flag' to begin running the program, it collects all the data which for this device is 8 distance points for one 360-degree cycle. It does this until another flag is raised which is where it will stop the program. This is implemented through interrupts.

The second step is transmission. Now that data has been collected and it has been received by the microcontroller, this data now must be transferred to the PC application for coordinate conversion. To do this, a serial communication interface (SCI) is needed. The Universal Asynchronous Receiver/Transmitter (UART) interface is implemented to receive and send data. Many devices have the UART interface imbedded in their systems and so does the MSP432E401Y microcontroller. Data coming from the ToF can be transmitted from point A to B. Serial transmission involves sending one bit at a time. The total number of bits transmitted per second is called the baud rate (bps) [4]. The baud rate for this system is 115200.

Lastly, once the data has been sent to a PC application (Python, Matlab, RealTerm etc), now the data (distance measurements) can be converted into xyz coordinate points. Below are key equations used to breakdown the distance measurement from the sensor into yz coordinates. Based on the angle step, the y and z components are determined by taking the sine and cosine of the distance as show in equations 1 and 2 below.

$$y = distance * \cos\theta \qquad (1)$$

$$z = distance * sin\theta \qquad (2)$$

Now that, the distance has been broken into a 2D plane, the x coordinate is manually determined to map a 3D model of the space. An example of the use of the formulas above is stated below.

Suppose a distance measurement was determined to be 355 mm at 135 degrees. To determine the yz coordinate points in space, simply substitute 355 for the $distance$ and 135 for θ and solve for each of the 2 equations above. The answer yields that y = -251.02 mm and z = 251.02 mm. Next, the x coordinate points can be chosen and set to repeat every 10 mm, generating multiple yz slides and mapping a 3D model of the space. Therefore, a possible coordinate point for this example could be (10, -251.02, 251.02).

Visualization:

To visualize the distance measurements collected, a PC and programming software is required. For this product, a Yoga 7 Series Lenovo Laptop, Windows 11 operating system is used to compute and process all the data received from the microcontroller and ToF using Keil and Python v.3.9. programming software.

Obtaining visualized data requires certain packages to be installed into PC system. Once Python has been installed, simply import the Pyserial package using the following command,

```
pip install pyserial
```

to give the PC application the ability to communicate with the ToF based on serial communication. Specified Python code is used to decode and extract the data sent by the UART which is then stored into an empty text file. From that, the distance can be extracted from the text file and analyzed to determine y-z coordinate points using trigonometric equations. For the x coordinate, this can be manually specified.

Next, the xyz coordinates must be transferred into an xyz file, which the functions for 3D mapping imported with Open3D can be used to visualize the data. The command to import Open3D is

```
pip install open3d
```

Rough Dimensioned Sketch:

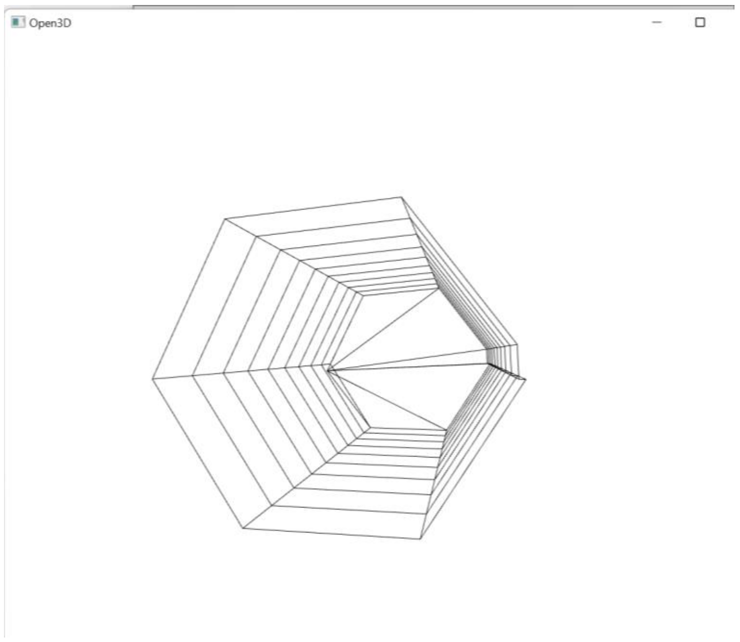
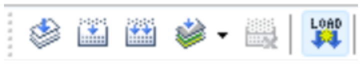


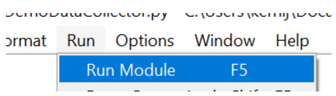
Figure 4 - 3D Space Visual

Application Example with Expected Output

1. Begin by loading the C program code on to the microcontroller



2. Run the module of the Python code



3. Reset the microcontroller and hit 'Enter' on the PC
4. A "Please Wait" message will appear and once "SensorInt Successful" is displayed, push the onboard PJ1 button
5. The ToF will begin to rotate and pause at every 45 degrees to collect the distance measurement. Press the PJ1 button to stop the rotation if desired.
6. As data is being collected, data is being displayed. 8 distance measurements in total should be displayed at the end of one full rotation. Simultaneously, this data is being written to a text file (captureDistance.txt).
7. Next, each line of the text file is read and analyzed using strip() and split() Python functions to extract the distance measurement only.
8. Taking the distance, the y and z coordinates are determined in Python using trigonometry (see equations 1 & 2) and is written to an xyz file.
9. For the x coordinate, the values are manually determined, increasing by 40 mm all 8 distance points 10 times.
10. Once xyz coordinates are determined and written to an array in the xyz file, the values are plotted using the open3d visualization function.
11. The function should graphically plot all 80 points (8*10) in 3D space
12. Finally connect the lines off all coordinate points and a rough dimensioned illustration of the space is displayed as shown in *Figure 4*.
13. The program ends and can be restarted again, starting from step 2.

Brief Users Guide

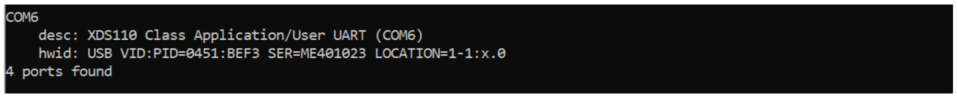
To get this project up and running, begin by acquiring all components (listed in features) needed for the project. Assemble these components and install the necessary software used in this project.

To assemble the physical components, refer to the circuit schematic for all the pin/port connection between all device components of this product (*Figure 5 & 6*). Ensure the USB cable along with the microcontroller is plugged into your PC. Have pins PH0-PH3 connected to the IN1-IN4 respectively on the stepper motor driver. Supply power to the stepper motor module by connecting 3.3V and GND from the microcontroller to pins on the stepper motor driver labelled '+' and '–' (5-12V). Then, proceed to connect pin PB2 on the microcontroller to SCL on the ToF and PB3 to SDA. Supply a voltage of 3.3V from the microcontroller to VIN on the ToF. Any pin labelled 3.3V will equally work. Ground the ToF by connecting GND to any pin labelled GND on the microcontroller. Once this is complete, there should be 10 wires connected to the microcontroller, 6 wires connected to the stepper motor module and a USB cable connected to the microcontroller, plugged into your PC. Finally, connect the grouped wires of the stepper motor to the stepper motor driver (*Figure 6*).

Assuming the correct code is provided, begin to set the device specific characteristics values in the Python file. For the microcontroller to communicate with Python, the port for which the microcontroller is connected to your PC needs to be acknowledged by Python in order to receive the encoded data. Using the command below in command prompt will return the user UART port code for which will need to be defined in the Python code.

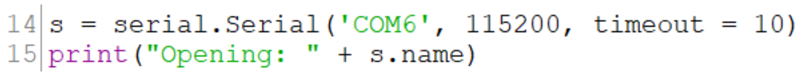
```
python -m serial.tools.list_ports -v
```

The screenshot below should be the expected output. Look for what reads "XDS110 Class Application/User UART (COM#)" and the COM port should be displayed.



```
COM6
    desc: XDS110 Class Application/User UART (COM6)
    hwid: USB VID:PID=0451:BEF3 SER=ME401023 LOCATION=1-1:x.0
4 ports found
```

Record your PC specific port code and return to the Python code and update the port code on line 14 seen below. Set the baud rate to 115200 and a timeout of 10 in the same line.



```
14 s = serial.Serial('COM6', 115200, timeout = 10)
15 print("Opening: " + s.name)
```

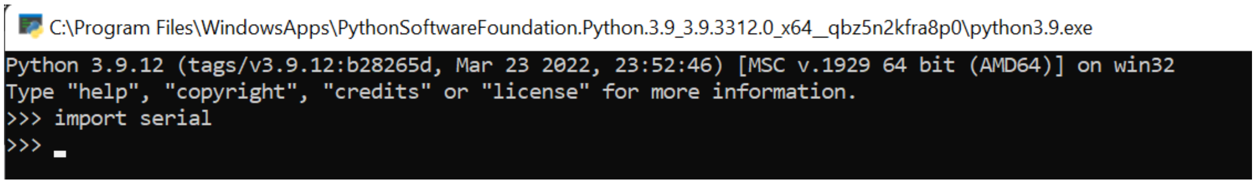
If not done already, import or install all necessary packages. Below are the required packages to install and import and their respective commands to type into your command prompt.

<u>Command Prompt Commands</u>

Pyserial (import serial) → pip install pyserial

Open3D (import open3d) → pip install open3d

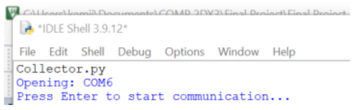
To check you have installed these packages correctly, open your installed Python version (shell) and type 'import serial' and 'import open3d' and press enter. The packages have successfully been installed if nothing follows the command as shown below.



```
C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.9_3.9.3312.0_x64__qbz5n2kfra8p0\python3.9.exe
Python 3.9.12 (tags/v3.9.12:b28265d, Mar 23 2022, 23:52:46) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import serial
>>>
```

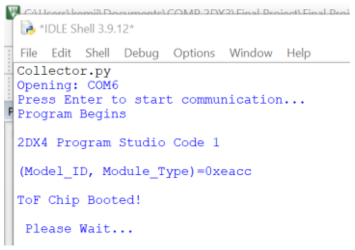
Now, the device is ready to be used. Be sure to acknowledge where the reset and PJ1 buttons are on the microcontroller. Refer to the microcontroller circuit schematic (*Figure 6*) for where to locate the buttons.

Open your Keil Project and load the program onto your microcontroller. Go over to Python at the top and click 'Run' then on the drop down, click 'Run Module'. A new shell window should open, and the following message should be displayed.



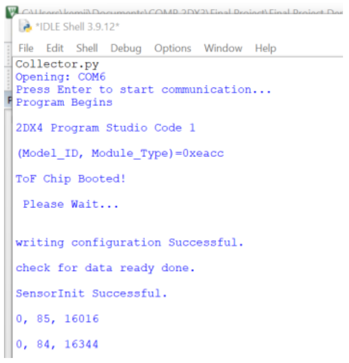
```
Collector.py
Opening: COM6
Press Enter to start communication...
```

Before hitting enter, reset your microcontroller at this time then proceed to hit enter on your PC. The following should be displayed.



```
Collector.py
Opening: COM6
Press Enter to start communication...
Program Begins

2DX4 Program Studio Code 1

(Model_ID, Module_Type)=0xeacc

ToF Chip Booted!

 Please Wait...
```

At this point, patiently wait for the ToF to be ready and once 'SensorInt Successful' is displayed, go ahead, and push the PJ1 button on the microcontroller and the rotation of the stepper motor and the collection of data by the ToF sensor should begin and synchronously display the distance on the Python console.

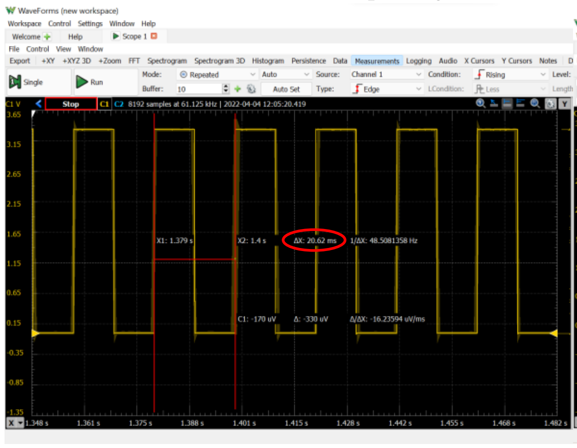


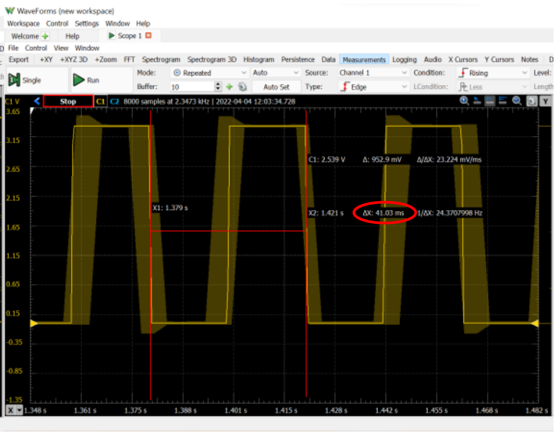
```
Collector.py
Opening: COM6
Press Enter to start communication...
Program Begins

2DX4 Program Studio Code 1

(Model_ID, Module_Type)=0xeacc

ToF Chip Booted!

 Please Wait...


writing configuration Successful.

check for data ready done.

SensorInit Successful.

0, 85, 16016

0, 84, 16344
```

Limitations

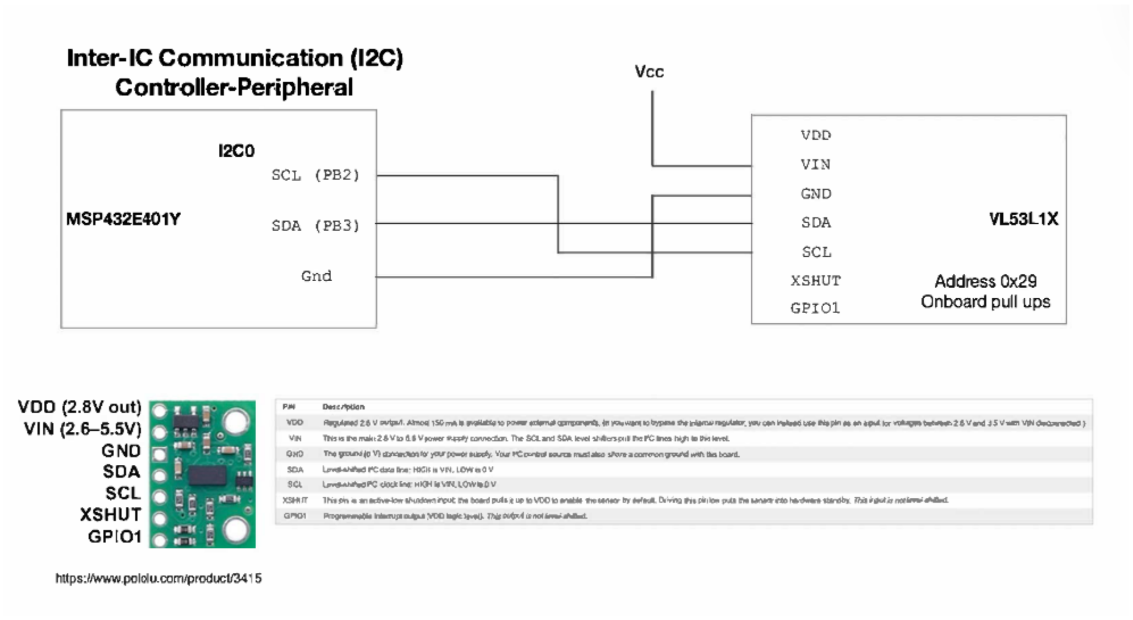
1. **Summarize any limitations of the microcontroller floating point capability and use of trigonometric functions**: The Floating-Point Unit (FPU) provides conversions between fixed-point and floating-point data formats, and floating-point constant instructions, more specifically, 32-bits instructions for single-precision (C float) data-processing operations [5]. Python receives distance values as integers. Trigonometric functions were implemented in the Python file by importing the math Python library (import math). Performing trigonometry by using sine and cosine functions from the math library did not cause any limitations since casting an integer as a float is possible in Python.

2. **Calculate your maximum quantization error for each of the ToF module:** The maximum quantization error is the ranging error from the datasheet with a value of 20mm to 25mm.
3. **What is the maximum standard serial communication rate you can implement with the PC. How did you verify?** The maximum standard serial communication rate to implement with the PC is 128000 bps. To verify this, navigate to the PC Device Manager and select 'Ports (COM & LPT)'. Right click on the 'XDS110 Class Application/User UART COM#' and select 'Properties' and then click 'Port Settings'. There you can see the 'Bits per second' drop down. All the available baud rates are displayed, and the largest baud rating is identified as the maximum standard serial communication rate you can implement with the PC.
4. **What were the communication methods and speed used between the microcontroller and the ToF modules?** I2C (UART) serial communication protocol with a speed of 100kbps to transfer data.
5. **Which element is the primary limitation on speed? How did you test this**? The element that primarily limits the speed is PSYSDIV variable. Below is a waveform of the 120MHz and 60MHz (device specific clock speed) bus frequency. PSYSDIV was changed from 3 (120MHz) to 7 (60MHz) and the period was reflected in the output on the waveform. Since the frequency decreased by 50% the period is expected to double.





Circuit Schematic



PIN	Description
VDD	Regulated 2.8 V output. Almost 150 mA is available to power external components. (If you want to bypass the internal regulator, you can instead use this pin as an input for voltages between 2.6 V and 3.5 V with VIN disconnected.)
VIN	This is the main 2.6 V to 5.5 V power supply connection. The SCL and SDA level shifters pull the I²C lines high to this level.
GND	The ground (0 V) connection for your power supply. Your I²C control source must also share a common ground with this board.
SDA	Level-shifted I²C data line: HIGH is VIN, LOW is 0 V
SCL	Level-shifted I²C clock line: HIGH is VIN, LOW is 0 V
XSHUT	This pin is an active-low shutdown input; the board pulls it up to VDD to enable the sensor by default. Driving this pin low puts the sensor into hardware standby. *This input is not level-shifted.*
GPIO1	Programmable interrupt output (VDD logic level). *This output is not level-shifted.*

Figure 5 - ToF to Microcontroller Circuit Schematic

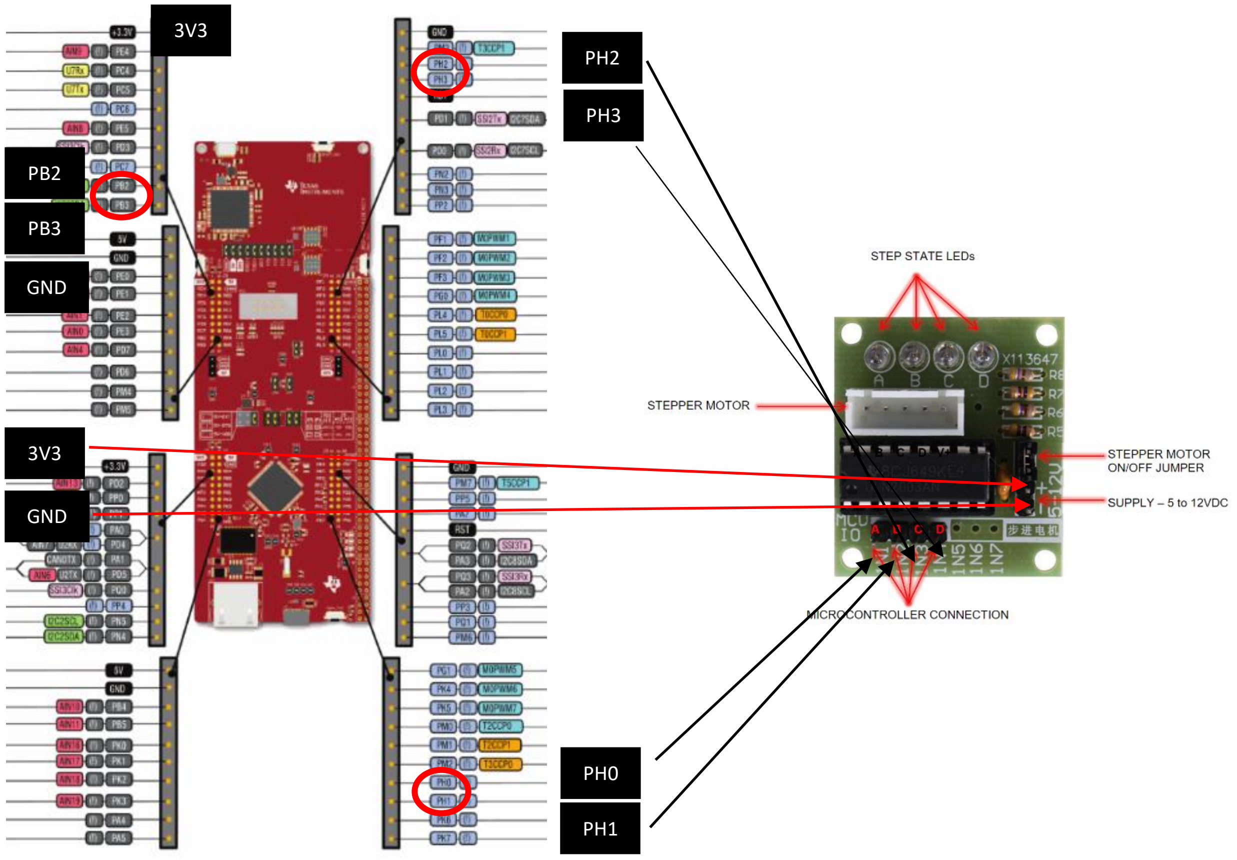


Figure 6 - Microcontroller to Stepper Motor Circuit Schematic

Programming Logic Flowcharts

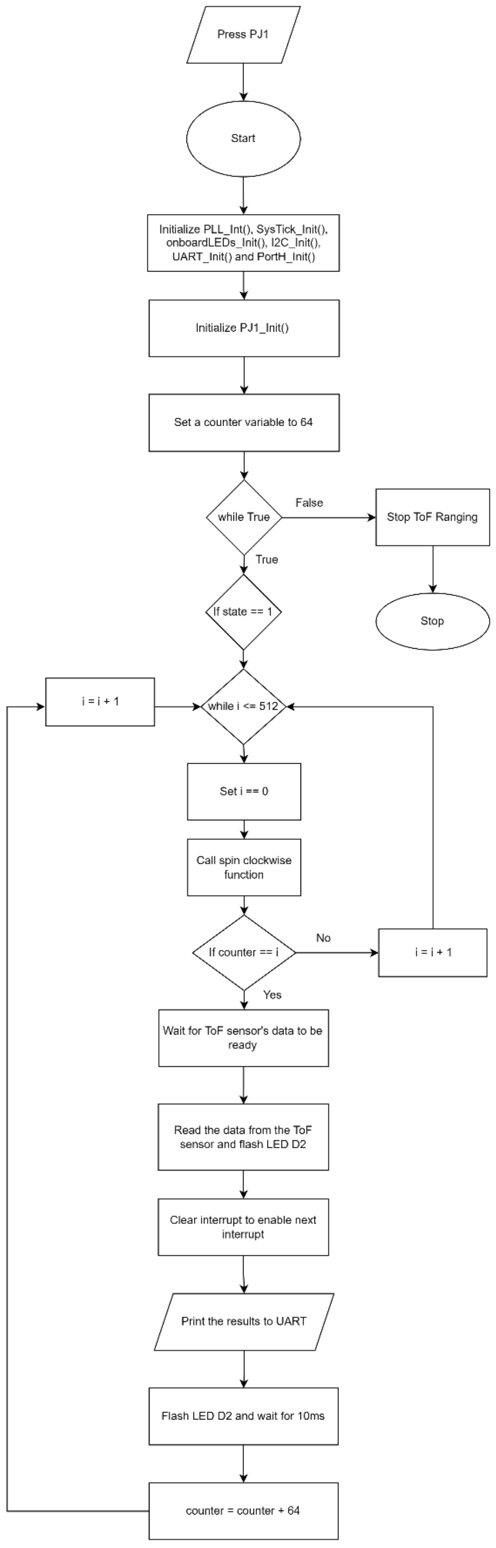

Press PJ1
Start
Initialize PLL_Int(), SysTick_Init(), onboardLEDs_Init(), I2C_Init(), UART_Init() and PortH_Init()
Initialize PJ1_Init()
Set a counter variable to 64
while True
False
Stop ToF Ranging
Stop
True
If state == 1
i = i + 1
while i <= 512
Set i == 0
Call spin clockwise function
If counter == i
No
i = i + 1
Yes
Wait for ToF sensor's data to be ready
Read the data from the ToF sensor and flash LED D2
Clear interrupt to enable next interrupt
Print the results to UART
Flash LED D2 and wait for 10ms
counter = counter + 64

Figure 6 - Keil C Program Flowchart

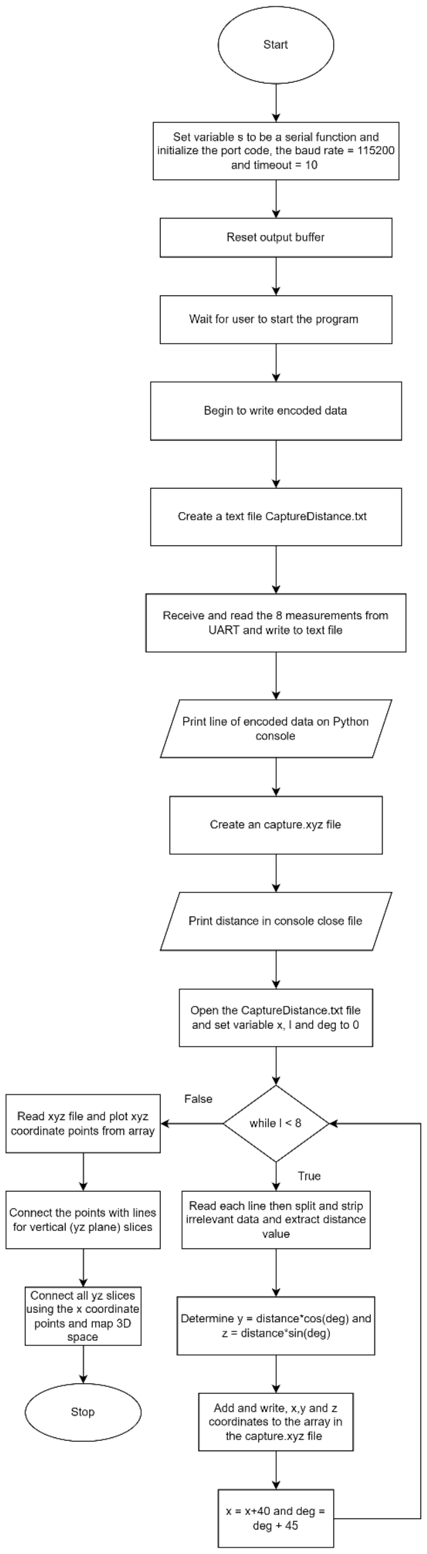

Start
Set variable s to be a serial function and initialize the port code, the baud rate = 115200 and timeout = 10
Reset output buffer
Wait for user to start the program
Begin to write encoded data
Create a text file CaptureDistance.txt
Receive and read the 8 measurements from UART and write to text file
Print line of encoded data on Python console
Create an capture.xyz file
Print distance in console close file
Open the CaptureDistance.txt file and set variable x, l and deg to 0
False
while l < 8
Read xyz file and plot xyz coordinate points from array
True
Connect the points with lines for vertical (yz plane) slices
Read each line then split and strip irrelevant data and extract distance value
Connect all yz slices using the x coordinate points and map 3D space
Determine y = distance*cos(deg) and z = distance*sin(deg)
Stop
Add and write, x,y and z coordinates to the array in the capture.xyz file
x = x+40 and deg = deg + 45

Figure 7 - Python Program Flowchart

References

[1] “2021_2022 _2DX3_2DX4_Project_Specification” class notes for 2DX3, Department of Engineering, McMaster University, Winter, 2022.

[2] “vl53l1x” class notes for 2DX3, Department of Engineering, McMaster University, Winter, 2022.

[3] “Week 6 – Implementing Signal Transfer Functions & Calibration” class notes for 2DX3, Department of Engineering, McMaster University, Winter, 2022.

[4] “Week 7 – Communication Protocols” class notes for 2DX3, Department of Engineering, McMaster University, Winter, 2022.

[5] “MSP432E401Y datasheet | TI.com,” *Ti.com*, 2017. https://www.ti.com/document-viewer/MSP432E401Y/datasheet/device-characteristics#t95624825 (accessed Apr. 12, 2022).