2DX4: Microprocessor Systems Project
Final Project

Instructors: Dr. Doyle, Dr. Haddara, Dr. Shirani

Shiv Thakar – L04 – thakas4 – 400247588

Submitted By: Shiv Thakar, thakas4, 400247588

Table of Contents

Device Overview

Features

- Configured with a Texas Instruments MSP-EXP432E401Y microcontroller
    - ARM Cortex-M4 Central Processing Unit
- Uses a Bus Speed of 30 MHz
- Operating Voltage ranges from 2.6-5 Volts
- Programmed with C Assembly Language and Python 3.8
- UART Simplex Serial Communication with Python 3.8 and the pySerial Library
    - Communication Port: COM6
    - Baud Rate: 115200 bits per seconds
- VL53L1X Time-Of-Flight Sensor
    - FlightSense ranging technology
    - Fast ranging with three different distance modes
        - Short
        - Medium
        - Long
    - Up to 4000 mm distance measurement
    - Up to 50 Hz ranging frequency
- ULN2003A-PCB Stepper Motor
    - 5VDC operating voltage
    - 5.625° Step Angle
- This device takes full horizontal scans of the surrounding area and determines the distance of the nearest object to the sensor. It will then create a three-dimensional mesh of the area.

General Description

The system is designed to take three-dimensional measurements using the VL53L1X time of flight sensor which receives an XYZ plot of the surrounding area. The entire design uses the Texas Instruments MSP-EXP432E401Y microcontroller which configures a series of ports to interface with peripheral devices. Firstly, a push button is used to initiate the program as well as resetting the program once a horizontal scan has been taken. After the button is pushed, the motor and time of flight sensor can be used. The sensor is mounted on top of the stepper motor makes a full rotation (360 degrees) to determine the measurements of the area the device is in. The device receives a non-electric signal, and it is transduced to convert it into an electrical signal. The signal is preconditioned and then analog to digital conversion is executed. Now the signal can be transmitted using I^2C and UART communication protocols and then is visualized using the Open3D Library in Python. The VL53L1X sensor measures distance by determining how long it takes pulses of infrared laser light to reach the closest object and is reflected to the sensor. These measurements are taken in millimeters with 8-bits within one each measurement.

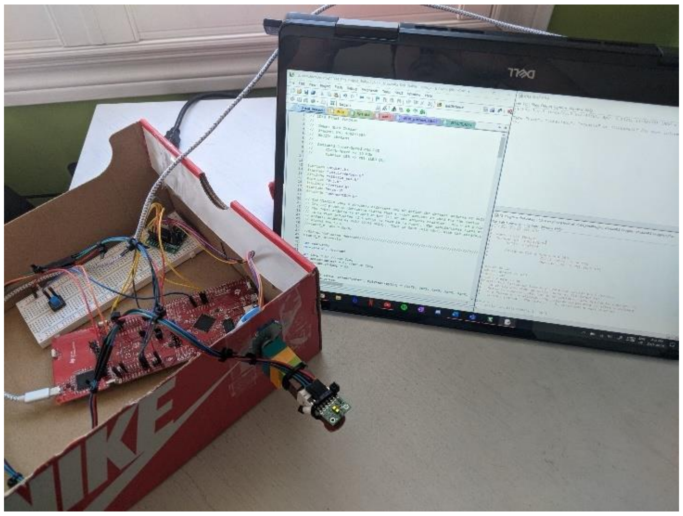
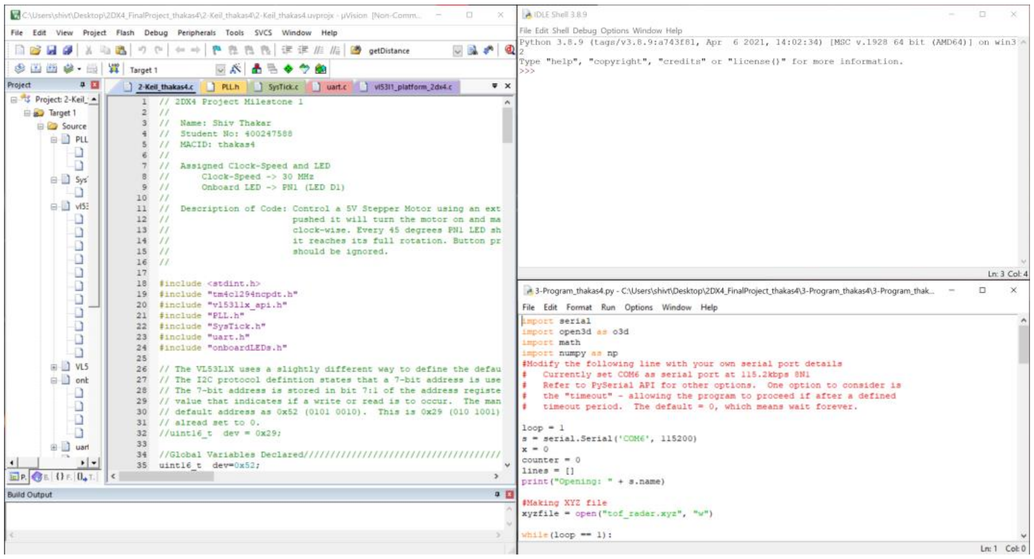


Figure 1: Picture of Microcontroller, Time-of-Flight Sensor, and PC



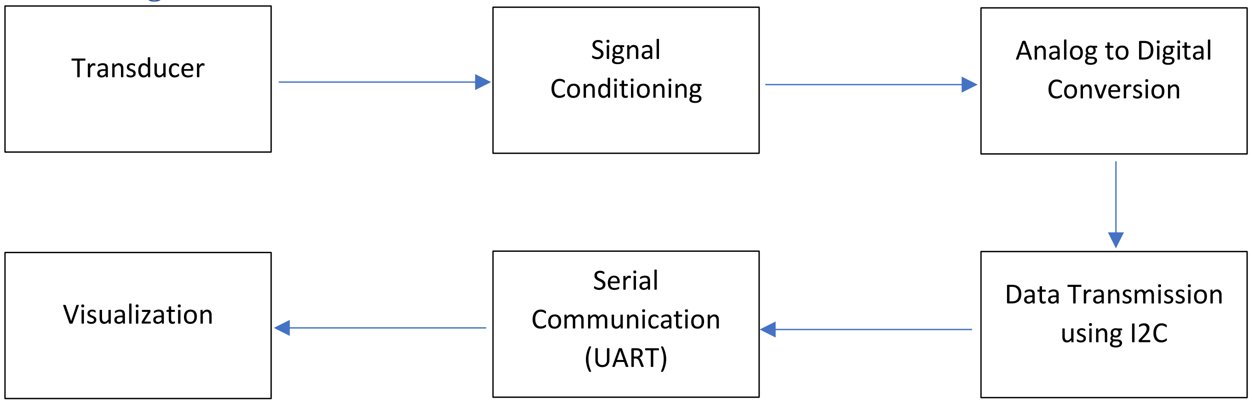
```
// 2DX4 Project Milestone 1
//
//  Name: Shiv Thakar
//  Student No: 400247588
//  MACID: thakas4
//
//  Assigned Clock-Speed and LED
//      Clock-Speed -> 30 MHz
//      Onboard LED -> PN1 (LED D1)
//
//  Description of Code: Control a 5V Stepper Motor using an ext
//                       pushed it will turn the motor on and ma
//                       clock-wise. Every 45 degrees PN1 LED sh
//                       it reaches its full rotation. Button pr
//                       should be ignored.
//

#include <stdint.h>
#include "tm4c1294ncpdt.h"
#include "vl53l1x_api.h"
#include "PLL.h"
#include "SysTick.h"
#include "uart.h"
#include "onboardLEDs.h"

// The VL53L1X uses a slightly different way to define the defau
// The I2C protocol defintion states that a 7-bit address is use
// The 7-bit address is stored in bit 7:1 of the address registe
// value that indicates if a write or read is to occur.  The man
// default address as 0x52 (0101 0010).  This is 0x29 (010 1001)
// alread set to 0.
//uint16_t  dev = 0x29;

//Global Variables Declared////////////////////////////////////
uint16_t  dev=0x52;
```

```
Python 3.8.9 (tags/v3.8.9:a743f81, Apr  6 2021, 14:02:34) [MSC v.1928 64 bit (AMD64)] on win3
2
Type "help", "copyright", "credits" or "license()" for more information.
>>>
```

```python
import serial
import open3d as o3d
import math
import numpy as np
#Modify the following line with your own serial port details
#   Currently set COM6 as serial port at 115.2kbps 8N1
#   Refer to PySerial API for other options.  One option to consider is
#   the "timeout" - allowing the program to proceed if after a defined
#   timeout period.  The default = 0, which means wait forever.

loop = 1
s = serial.Serial('COM6', 115200)
x = 0
counter = 0
lines = []
print("Opening: " + s.name)

#Making XYZ file
xyzfile = open("tof_radar.xyz", "w")

while(loop == 1):
```

Figure 2: Screenshot of PC

Block Diagram


Transducer
Signal Conditioning
Analog to Digital Conversion
Data Transmission using I2C
Serial Communication (UART)
Visualization

Device Characteristics

Table 1: Device Characteristics

Component	Specification	Description
Bus Speed	30 MHz	Bus Speed and Clock Speed
GPIO Port N	Pin 1 (onboard LED D1)	Port N Pin 1 is used to flash the onboard LED when a distance measurement is taken.
GPIO Port B	Pin 2	Pin 2 is used for SLC for I2C
	Pin 3	Pin 3 is used for SDA for I2C
GPIO Port H	Pins 3:0	This port is used to control the stepper motor's rotation. This stepper motor uses full-step mode and uses each of the pins to control its movement and direction.
GPIO Port E	Pin 0	Pin 0 in Port E is used of button scanning and checks if its being pushed. When it is pushed, it can start the program.
GPIO Port M	Pin 0	Pin 0 in Port B is used of button scanning and checks if its being pushed. When it is pushed, it can start the program.
GPIO Port A	Pin 1:0	UART Initialization
GPIO Port G	Pin 0	Time of Flight Sensor XSHUT
Serial Communication Port and Communication Speed	COM6 Baud Rate of 115200 bps	Data Transmission

Detailed Description

Distance Measurement

The distance measurements are first received by Signal Acquisition with the sensor and the microcontroller. Signal Acquisition begins by taking a real-world measurement, or a non-electric signal and it is transduced which converts this signal into an electrical one. Within the time-of-flight sensor, a measurement of time is first taken, this signal is converted into an electrical signal which is then pre-conditioned. Signal conditioning is done to prepare the signal for analog to digital conversion. Once the signal is converted, we can use serial communication protocols to transmit data taken from the slave device. In the case of the time-of-flight sensor, a distance measurement it returned to the user.

The VL53L1X Time-of-Flight sensor measures distances using LIDAR (light detection and ranging). The sensor can acquire information about an object without physically touching it. It uses pulses of laser infrared light to measure multiple different ranges. The short range can measure up to 1360 mm in the dark and 1350 mm when under strong ambient light. The medium range can measure up to 2900 mm in the dark and up to 760 mm under strong ambient light. Finally, the long range can measure up to 3600 mm in the dark and up to 730 mm under string ambient light. This is summarized below:

Table 2: Distance Modes for the VL53L1X Time-of-Flight Sensor

Distance Mode	Maximum Distance under low-light conditions/dark (mm)	Maximum Distance Under Strong Ambient Light (mm)
Short Range	1360	1350
Medium Range	2900	760
Long Range	3600	730

These distances are measured using the time-of-flight principle, the emitter will first send out pulses of light (photons) and once it hits the closest target object, it is reflected back to the sensor. The measured distance is then calculated, in millimeters, within the sensor with the following formula:

$$Measured\ Distance = \frac{travel\ time\ of\ light/photon}{2} \times (Speed\ of\ Light)$$

Equation 1: Measured Distance from the Time-of-Flight Sensor

After the time-of-flight sensor has received and transmitted the data/measurement, it is outputted with UART Simplex Communication using pySerial. These measurements are manipulated to determine the XYZ coordinates in a three-dimensional space and are stored with a file created through Python in the *.xyz* file format. The XYZ coordinates are determined using trigonometric formulas:

$$Y = (Measured\ Distance) \times \cos(\theta)\,, angle\ is\ measured\ in\ radians$$

$$Z = (Measured\ Distance) \times \sin(\theta)\,, angle\ is\ measured\ in\ radians$$

Equation 2 & 3: Coordinates of YZ on the Plane

These equations are used after the distance is measured. The measured distance is multiplied by the cosine or sine of the current angle in the motor's rotation. X is determined by incrementing by a defined delta (20 cm) within the Python program. When a new measurement is taking out of the 10 needed, the X coordinate will be incremented to show that the device has moved within the space.

The program will begin when the user pushes the button which will show that a new input has been transmitted. Button Scanning is used to check for inputs. Referring to the circuit schematic, the button is connected to Port M Pin 0 and Port E Pin 0. When there the button is not pushed, both ports have value of 0 in its respective data registers. The program will sense that a button is pushed when both data registers carry a value of 1. This will trigger the components of the device to initiate. When the button is pushed, the time-of-flight sensor will get booted and get ready to start ranging for data, once the user is notified through the PC, the motor will start its rotation. The stepper motor is using a full-step mode (with 10 millisecond delays in between each step) moving in the clockwise direction. In full-step mode, the bits within the data register within Port H (Pins 3:0) are changed in each step, so that the motor can move. Every 11.25 degrees, the motor will take a small pause, and the time-of-flight sensor will start emitting and will receive the distance and output the data, using UART communication, to the user. The UART serial communication port used in this program is COM6 with a baud rate of 115200 bps. The distance is manipulated to determine the XYZ coordinates which are then outputted to a file (of *.xyz* format). Once the measurements for one horizontal scan is taken, the button is pushed to reset the components to take another set of measurements, which will be repeated 10 times. After all the 10 horizontal scans are completed and stored within the file, the Open3D Library within Python will visualize the data on a three-dimensional plot.

The time-of-flight sensor is using a communication protocol called I^{2}C. This protocol allows peripheral digital devices to communicate with the microcontroller. I^{2}C consists of two signals, the serial data line (SDA) which sends and receives data, and the serial clock line/signal (SCL) which synchronizes data transfer. Data transmission begins with SDA transitioning from a high to a low, and then the SCL also transitions from a high to a low. Data is transmitted through messages and are further broken down to frames of data. Each of the messages has a frame that holds the address of the slave device. The time-of-flight sensor has an address of 0x29 hexadecimal. Furthermore, data frames contain the data that is being transmitted. The data transmission ends with the SCL line transition from a low to a high, and then the SDA line transition from a low to a high.

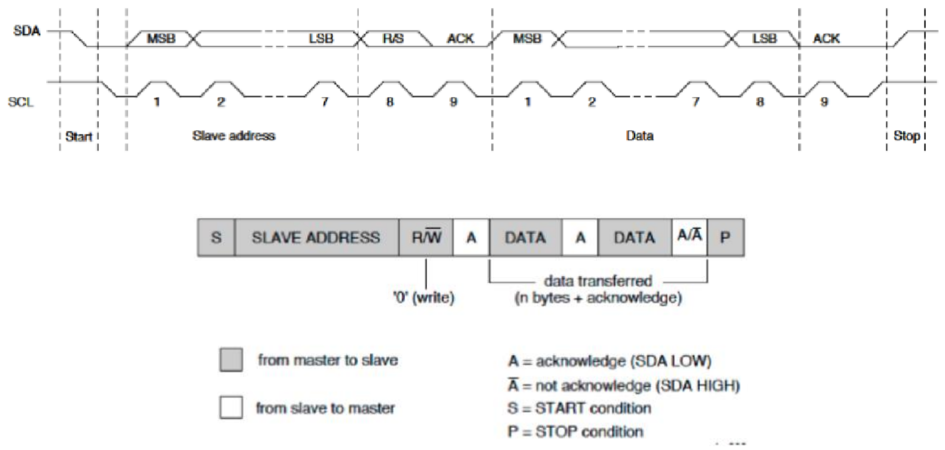


Figure 3: I^{2}C Diagram

Visualization

The computer that this program is running on is a Dell Inspiron 7586, using an Intel Core i7-8586U CPU, with 16 GB of RAM and 512 GB of storage. The computer is currently running a 64-bit operating system, Windows 10 Home. This program is using Keil uVision5 coded in C and Python 3.8.

The Python program uses many different libraries. The first library used is the pySerial Library, which gives users access to the serial ports and provides backends for Python running on Windows. The library is used to connect to serial communication ports for UART. It will open the ports when a new object of the class is declared with the port name and baud rate given. This program uses communication port COM6 and a Baud Rate of 115200 bits per second.

This program also employs the use of Python's Math library. Which allows you to use mathematical operations such as powers, logarithmic, trigonometric, and more. This program uses the trigonometric function to determine the YZ coordinates. The trigonometric functions use radians, so the degrees that are used are converted into radians before the YZ coordinates are determined, using Equation 2 and 3. This process is repeated until all ten scans are taken. On the flowchart, once the coordinates are determined, the *.xyz* file is closed and now the program can visualize these points.

Python's Open3D Library is imported to use for visualizing the coordinates in a three-dimensional space. When the data is received from the time-of-flight sensor, the coordinates are first determined, and they are stored in an *.xyz* file. This file can be read by the point cloud function (io.read_point_cloud()) from Open3D. Various point variables are created so they can be connected together and these points can be visualized with a series of points on a given plane using the point cloud function and the numpy library, specifically the asarray() function. The points are connected within a repetition structure to append the lines to a new list. After these points have been plotted, thirty-two new variables are declared that represent each of the points from the measurements. These points are then connected to form lines and they are then outputted in an Open3D window showing the three-dimensional representation of the surrounding area.

The program will recreate the surrounding area in a three-dimensional plot. To test the functionality of the program and sensor, my bedroom was recreated. In the drawing below, it shows a multi-view layout of the room as well as an isometric view. The highlighted area in the box shows the exact area that was tested.

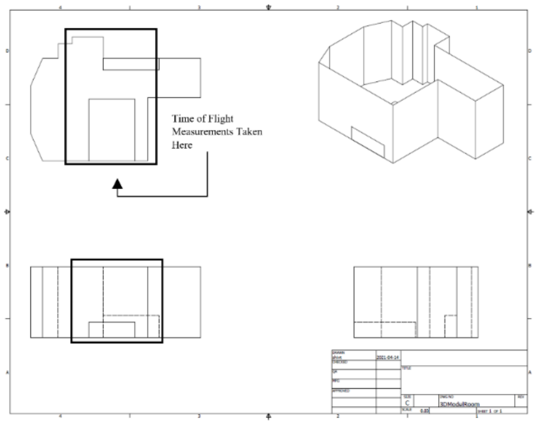


Figure 4: Multiview and Isometric Drawing of the Surrounding Area

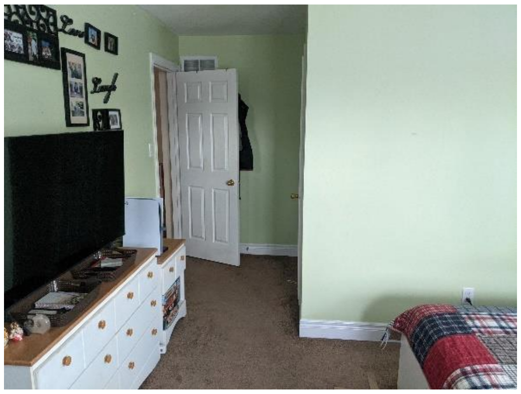


Figure 5: Picture of the Testing Space

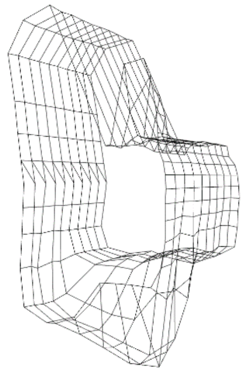


Figure 6: Sample Plot Using Open3D

Application Example

Python 3.8 is used to communicate and visualize the signal that is acquired from the time-of-flight sensor. UART communication protocol is used to show the data that is received from the sensor. The program uses the pySerial library to open the UART serial port, COM6 with 115200 bps baud rate. The signal is received from the slave device (time-of-flight sensor) using I^2C protocol and is sent to the microcontroller and UART communication protocol. In the Python program, each bit is read individually and decoded to determine what data is being transmitted to the PC.

1) A new object of the serial class is declared, which will open the UART serial port COM6.
2) Bits from the microcontroller are read one-by-one via UART protocol.
3) When the motor reaches an angle at which the slave device receives a single. I^2C protocol is used to decode the signal and read by the program.
4) The decoded signal is then stored as an integer to determine the XYZ coordinates. X is determined by incremented the variable by a defined amount. While YZ are determined through trigonometric functions found in Python's math library.
5) These coordinates are stored within an *.xyz* file.
6) Once all signals are acquired and measurements are received, the coordinates are plotted on an XYZ plane using the Open3D Libaray and NumPy Library.

Setup Guide

1) The components of the device must be connected to the MSP-EXP432E401Y microcontroller. This includes:
   - The Push Button
   - The Stepper Motor
   - The VL53L1X Time-of-Flight Sensor
2) The push button can be easily set-up using a 10 kilo-ohm resistor. Referring to the circuit schematic, the button is placed on to the breadboard, with the resistor is connected to the left set of pins on the button. The resistor is connected to a voltage source of 3.3 V from

the microcontroller and Port M Pin 0. The right set of pins on the push button are directly connected to Port E Pin 0.

3) Next, the stepper motor is connected from its' input pins and the predesigned breakout board to Port H Pins. Input Pin 1 to Input Pin 4 is connected to Port H Pin 0 to Pin 3, respectively. The motor is powered by a 5V voltage source and ground are connected directly to the microcontroller.
4) Looking at the circuit schematic, the time-of-flight sensor is directly connected to the microcontroller. The V_{in} on the sensor is connected to the 3.3V voltage source on the microcontroller, and the GND pin is connected to GND on the microcontroller. Next the data line (SDA) for I^2C is connected to Port B Pin 3. The clock line (SLC) is connected to Port B Pin 2.
5) After the components are connected and ready, looking at Keil uVision5, the PLL.h file must be modified to hold the number 15 in PSYSDIV. This will set the Bus Speed/System Clock to 30 MHz.
6) Next, the multiplier in SysTick.c file must be updated for the new Bus Speed. Using the formula:

$$delay\ (in\ seconds) = (Multipler) \times \frac{1}{30\ MHz}$$

$$Multiplier = 0.01\ s\ \times \frac{30\ 000\ 000}{s} = 300\ 000$$

<u>Equation 4: Clock Delay Equation</u>

This new multiplier will assure that the delay in Systick_Wait10ms is 10ms. This assumes that the bus speed is 30 MHz.

7) When communicating using UART, we need to set the communication protocols. This device uses serial communication which will send the bits one-by-one over the communication channel. The communication channel used is COM6.
8) The communication channel also needs a Baud Rate, which tells the channel the number bits that should be transmitted per second. This device uses a baud rate of 115200 bits per second.

Limitations

1) Summarize any limitations of the microcontroller floating point capability and use of trigonometric functions.

Most embedded systems are built with a 32-bit CPU, which can execute mathematical and arithmetic operations. These operations can be implemented using signed and unsigned integers. Using floating point operations comes with trade-offs. The code can favour performance over precision or vice versa, but not both. Performance can help to reduce the amount of time it takes to execute the operation, while precision returns more accurate calculations. The CPU within the MSP-EXP432E401Y has a Floating-Point Unit. The FPU only allows for single-precision floating point for 32-bit

CPU, whereas double-precision can be used within a 64-bit CPU. The FPU allows the microcontroller to handle floating-point numbers and operations. The microcontroller can execute trigonometric functions from the C's math library on floating-point values.

2) Calculate your maximum quantization error for each of the Time-of-Flight modules.

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

$$Max\ Quantization\ Error = 0.061035$$

3) What is the maximum standard serial communication rate you can implement with the PC? How did you verify this?

The maximum standard serial communication rate that can be implemented on the PC is 128000 bits per second. This can be verified by going to the Device Manager on the PC. After this, click the view button on the top panel and make sure that "Show Hidden Devices" is checked. Now you see a category called "Ports (COM & LTP)", when this is expanded you can click on the UART port and the maximum baud rate can be seen under "Port Settings" and you can expand bits per second. There you can see that the maximum baud rate.

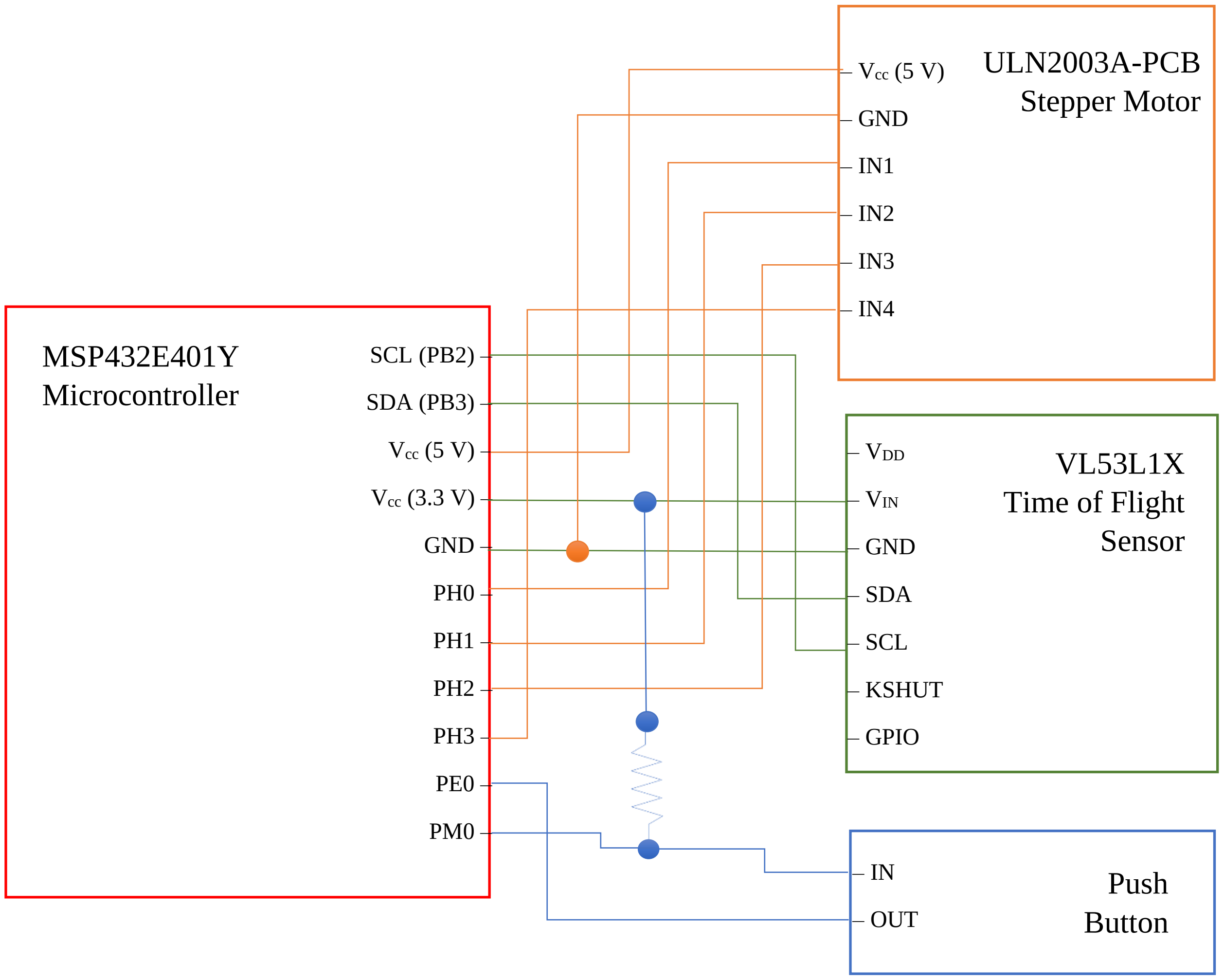
4) What were the communication method(s) and speed used between the microcontroller and the Time-of-Flight modules?

The Time-of-Flight modules use I^{2}C as a serial communication protocol which transfers bits along the data line (SDA). The microcontroller and the Time-of-Flight sensor transmit data between each other with a 100kHz frequency and 100kbps clock speed.

5) Reviewing the entire system, which element is the primary limitation on speed? How did you test this?

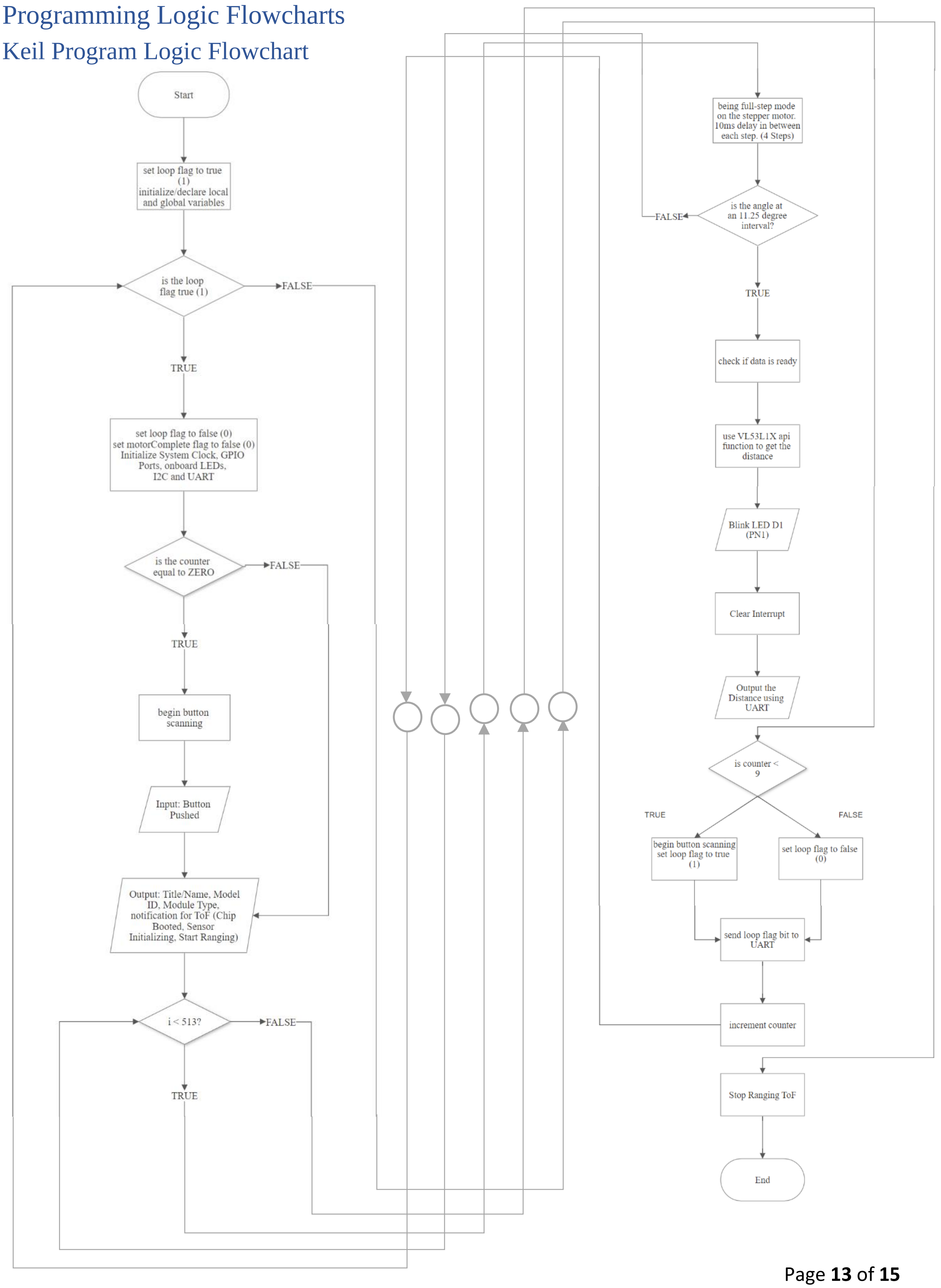
After reviewing the entire system, the time-of-flight sensor is the primary limitation on speed. When doing my demo, the sensor takes a long time to get ready to start ranging and booting up. This was tested by taking multiply measurements and the main reason it takes a long time to take 10 scans is that the sensor must start and stop ranging between every scan.

Circuit Schematic

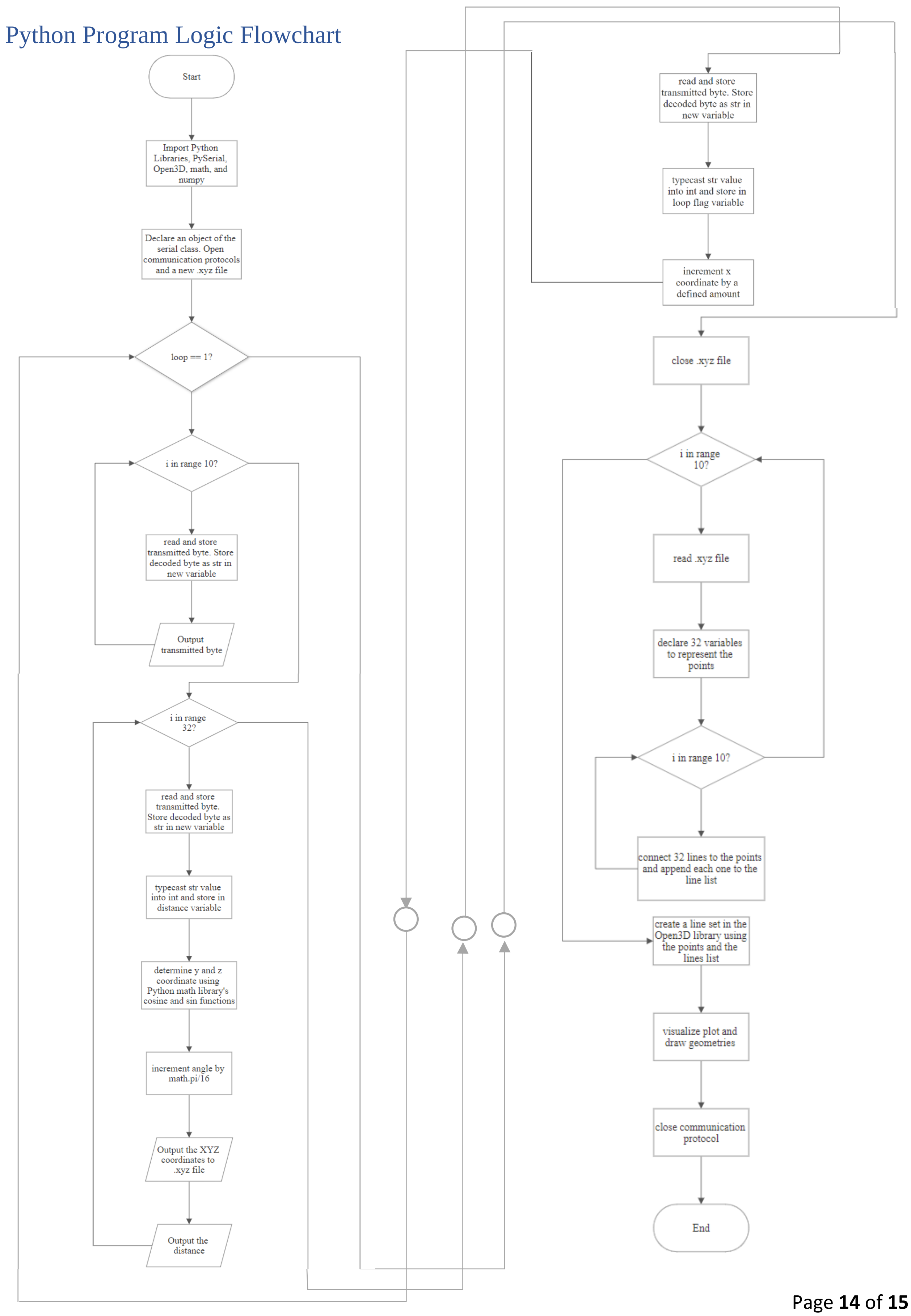


Programming Logic Flowcharts

Keil Program Logic Flowchart


Start
set loop flag to true (1) initialize/declare local and global variables
is the loop flag true (1)
FALSE
TRUE
set loop flag to false (0) set motorComplete flag to false (0) Initialize System Clock, GPIO Ports, onboard LEDs, I2C and UART
is the counter equal to ZERO
FALSE
TRUE
begin button scanning
Input: Button Pushed
Output: Title/Name, Model ID, Module Type, notification for ToF (Chip Booted, Sensor Initializing, Start Ranging)
i < 513?
FALSE
TRUE
being full-step mode on the stepper motor. 10ms delay in between each step. (4 Steps)
is the angle at an 11.25 degree interval?
FALSE
TRUE
check if data is ready
use VL53L1X api function to get the distance
Blink LED D1 (PN1)
Clear Interrupt
Output the Distance using UART
is counter < 9
TRUE
FALSE
begin button scanning set loop flag to true (1)
set loop flag to false (0)
send loop flag bit to UART
increment counter
Stop Ranging ToF
End

Python Program Logic Flowchart


Start
Import Python Libraries, PySerial, Open3D, math, and numpy
Declare an object of the serial class. Open communication protocols and a new .xyz file
loop == 1?
i in range 10?
read and store transmitted byte. Store decoded byte as str in new variable
Output transmitted byte
i in range 32?
read and store transmitted byte. Store decoded byte as str in new variable
typecast str value into int and store in distance variable
determine y and z coordinate using Python math library's cosine and sin functions
increment angle by math.pi/16
Output the XYZ coordinates to .xyz file
Output the distance
read and store transmitted byte. Store decoded byte as str in new variable
typecast str value into int and store in loop flag variable
increment x coordinate by a defined amount
close .xyz file
i in range 10?
read .xyz file
declare 32 variables to represent the points
i in range 10?
connect 32 lines to the points and append each one to the line list
create a line set in the Open3D library using the points and the lines list
visualize plot and draw geometries
close communication protocol
End

Google Drive Folder Link

Folder Link:
https://drive.google.com/drive/folders/1si1VFN_DC12h_BKRZIB1T7qur2pzmz12?usp=sharing

Demo Video:
https://drive.google.com/file/d/1Akqu4sS5wRiRU-SGRSVdOlWiPneT0qkX/view?usp=sharing

Question 1 Response:
https://drive.google.com/file/d/1xoge6PsYrYEy9FqgKTU5Zn9TlxzDT9yC/view?usp=sharing

Question 2 Response:
https://drive.google.com/file/d/1KWO7d9NRPV562ucaxG5Mjqy8CFLU8E_C/view?usp=sharing

Question 3 Response:
https://drive.google.com/file/d/1iTSK5PJxBXwt1VOS3JuU9sBwRrgWsuzS/view?usp=sharing

Keil Program:
https://drive.google.com/file/d/1wz5h_xoPP6cR2Jb1V_uY8ugn1CVF7l6u/view?usp=sharing

Python Program:
https://drive.google.com/file/d/1d0KnJ5U7CRaCOnirA1pwf9a3o2vczVil/view?usp=sharing