2DX: Microprocessor Systems Final Project

Instructors: Drs. Boursalie, Doyle, and Haddara

Boran Seckin – seckinb – 400305852
2DX3 – Thursday Afternoon – L04

April 11, 2022

Table of Contents

Device Overview

Features

- Scans a 3D area to create a spatial reconstruction of the space.
- Integrated using embedded systems and built around MSP432E401Y microcontroller.
- Works at 30 MHz bus speed at 5 volts.
- Uses relatively cheaper and smaller parts such as 3415-POLOLU VL53L1X time-of-flight and MOT-28BYJ48 stepper motor with total cost of 75 CAD.
- Has a 2-button interface with near plug-and-play convenience.
- Generates 3D models that are easy to navigate.
- Has adjustable resolution settings.
- Uses both I^2C and UART (115200 bps) for serial communication.
- Published with MIT license and therefore, open source.

General Descriptions

The objective of this project is to create an inexpensive and small alternative to Commercial Light Detection and Ranging (LIDAR) equipment using a time-of-flight sensor and a rotary mechanism. The device should be able to collect spatial measurements within a single vertical geometric plane. The collected data should be first stored in onboard device memory and then, transmitted to a personal computer (PC) using a wired serial interface. The PC should be able to represent the data in a way that it is a reconstruction of the 3D space which the measurements are taken of.

This implementation of the project uses a time-of-flight sensor that is connected to a stepper motor. A time-of-fight sensor measures the distance by transmitting a light beam towards a surface and recapturing the reflection of it. The time-of-flight sensor used in this project is 3415-POLOLU VL53L1X which costs 18 CAD. Additionally, MOT-28BYJ48 stepper motor is used to precisely rotate the sensor 11.25 degrees after each data acquisition for 32 times a set, in order to collect 32 points per 360 degrees. The motor costs 7 CAD.

Both the time-of-flight sensor and the stepper motor is controlled by the microcontroller MSP432E401Y found in the SimpleLink MSP-EXP432E401Y LaunchPad Development Kit. The ARM Cortex-M4F processor that is powering the microcontroller is set to work at 30 MHz bus speed [1]. It operates at 5 volts that is supplied by the micro-USB cable which is connected to the PC. It has a 1024 kilobytes of flash memory and 256 kilobytes of static random-access memory [1]. It supports both UART and I^2C as serial communication methods which are used in this project [1]. It is programmed using C. The kit costs 50 CAD.

The time-of-flight sensor is attached to the system sideways using an extension part so that, when it is rotated by the stepper motor, it will have a clear view of its 360-degree environment. The system is also equipped with 2 push buttons with software debounce. The black button is used to start and stop the data acquisition sequence. When pressed, it will command the microcontroller to take one full set of measurements. If pressed for a second time during a set, it will interrupt the set and stop. The yellow button is used to rotate the sensor in the reverse direction so that the wires can be untangled if needed. It follows the same set of rules as the black button in the opposite direction and with no measurements.

The measured data is transmitted to from the time-of-flight sensor to the microcontroller using Inter-Integrated Circuit (I^2C) serial communication protocol. The measurement is stored in local memory before it is transmitted to the PC using Universal Asynchronous Receiver-Transmitter (UART) serial communication protocol. Finally, the data is collected, parsed and visualized by the Python code running on the PC.

Block Diagram

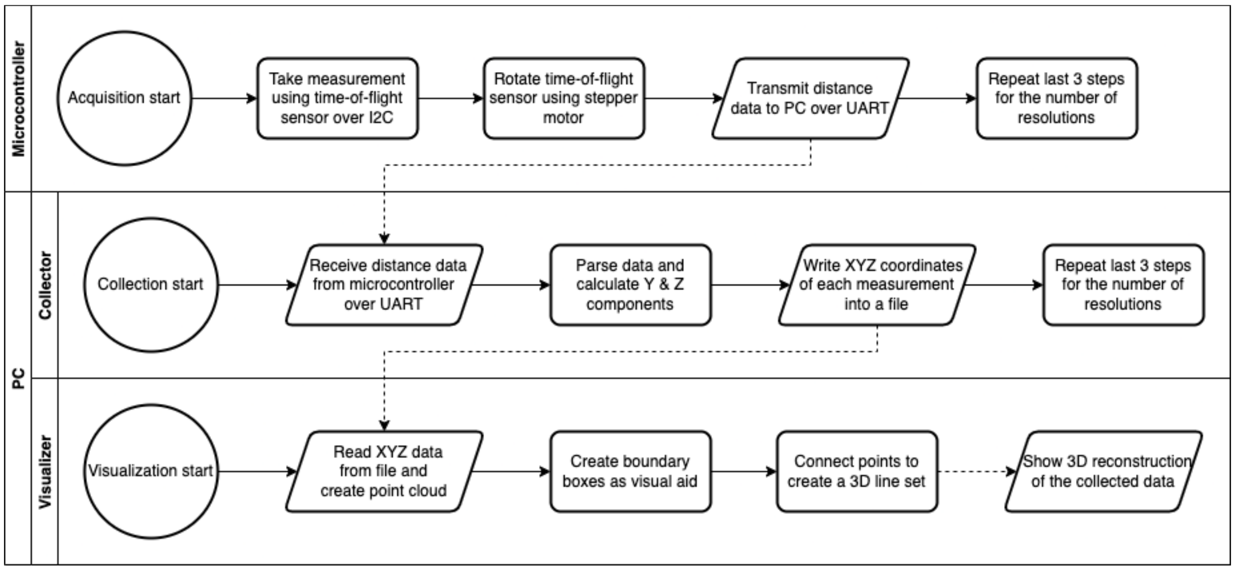
Microcontroller
Acquisition start
Take measurement using time-of-flight sensor over I2C
Rotate time-of-flight sensor using stepper motor
Transmit distance data to PC over UART
Repeat last 3 steps for the number of resolutions
PC
Collector
Collection start
Receive distance data from microcontroller over UART
Parse data and calculate Y & Z components
Write XYZ coordinates of each measurement into a file
Repeat last 3 steps for the number of resolutions
Visualizer
Visualization start
Read XYZ data from file and create point cloud
Create boundary boxes as visual aid
Connect points to create a 3D line set
Show 3D reconstruction of the collected data

Figure 1 - Block Diagram

Device Characteristic

- The microcontroller's bus speed is set to 30 MHz.
- Both push buttons use an active-high configuration.
- The time-of-flight sensor is supplied with 3.3 volts and the stepper motor is supplied with 5 volts.
- The UART serial communication is set to use a baudrate of 115200 with 8-bit word length, no parity bits, one stop bit and FIFOs.
- The I^2C address used for the time-of-flight sensor is 0x29.
- The C code uses the provided TM4C1294NCPDT Register Definitions, PLL, SysTick, UART, onboardLEDs and VL53L1X libraries.
- The Python code uses PySerial and Open3D libraries.

Pin Assignments

Microcontroller	Circuit
PL0	Servo Controller IN1
PL1	Servo Controller IN2
PL2	Servo Controller IN3
PL3	Servo Controller IN4
PB2	I2C SDA
PB3	I2C SCL
PH0	Start/Stop Push Button
PH1	Reverse Rotation Push Button

Detailed Description

Distance Measurement

The system uses the 3415-POLOLU VL53L1X time-of-flight sensor. The time-of-fight sensor has two surfaces. One of them transmits a light beam towards a target and the other one recaptures the reflection of that light. Since the speed of light is a known constant, the time it takes for the light to travel to the target and back can be used to calculate the distance between the sensor and the target.

The time-of-flight sensor has 4 connections. VIN and GND are connected to 3.3 volts and ground respectively on the microcontroller [2]. Additionally, to interface with the sensor, SDA and SCL pins are connected to PB2 and PB3 on the microcontroller respectively to establish an I^{2}C serial connection. The sensor is configured and used using the provided VL53L1X API library. The sensor is setup using the provided SensorInit function and the distance is measured using the StartRanging function. The measured distance is then, accessed using the GetDistance function.

Additionally, MOT-28BYJ48 stepper motor is used to precisely rotate the sensor clockwise between every measurement. Its IN1, IN2, IN3 and IN4 pins are connected to the PL0, PL1, PL2 and PL3 ports respectively on the microcontroller. The plus and minus ports of are also connected to 5 volts and ground respectively on the microcontroller. The stepper motor works by changing the polarities of the 4 electromagnets inside the motor. By changing the polarities in the correct order, the shaft between the magnets can be rotate in both rotations. The method used to switch polarities is called full step where two adjacent magnets are powered at all times. Between each state change (step), the code waits for 5 milliseconds to make sure the shaft has enough time to rotate. Since the resolution is set to 32, the system will take 32 measurements per 360-degree set. Therefore, the stepper motor will rotate the sensor 11.25 degrees after each data acquisition. This value is calculated using the formula 1 where the “number of measurements” value dictates how many measurements there will be in one set.

$$step\ degree = \frac{360}{number\ of\ measurements} \qquad [1]$$

The microcontroller is programmed in C. The main code works in an infinite loop as a finite state machine as shown in Figure 5 - Flowchart of the microcontroller code. The code first initializes all the necessary functionality such as setting Phase-Locked Loop (PLL) for the bus speed, SysTick for waiting, onboardLEDs for visual output, I^2C for sensor communication, UART for PC communication and GPIO ports for servo controls and button inputs. It then initializes and configures the time-of-flight sensor. Once the sensor is ready, it transmits "start" word over UART and enters the infinite loop.

Inside the infinite loop, the code first checks if the button connected to the PH0 is pressed. If it is pressed, it will enter the data acquisition procedure. To indicate the start, all the onboard LEDs are flashed, and a 50-millisecond delay is waited. Then, the time-of-flight sensor is issued a command to start measuring the range. The program waits for the sensor data to be available with 5 millisecond intervals. When the data is ready, it retrieves the distance data from the sensor using I^2C and temporarily saves it in onboard memory. It then, rotates the stepper motor and consequently the sensor clockwise by the amount calculated using formula 1. After the rotation, the distance data is transmitted to the PC over UART as explained in the next section. When the transmission is completed, it flashes the onboard LED connected to the PN1 port and waits for 100 milliseconds. Finally, the button PH0 is checked again to determine if it is pressed to stop the process. If it is, the process is halted, and the program returns back to the infinite loop. All the steps above, starting with waiting for data availability, are repeated according to the number of measurements specified in the code as resolution.

If the button PH1 is pressed instead, the stepper motor is rotated counter-clockwise 360 degrees. During this process, if the PH1 button is pressed again, the operation will be halted, and the program will return to the infinite loop.

Both buttons in the system uses an active-high configuration. Each button presses for stopping a process are debounce using software. When the button is pressed, the program will wait for it to be released to prevent accidentally registering multiple inputs.

Transmission

The distance data that is taken in the XY plane by the time-of-flight sensor is sent to the PC using UART serial communication method. The microcontroller is configured so that it uses UART0 over the micro-USB that is used to power and flash code. UART is configured to use a baudrate of 115200 with 8-bit word length, no parity bits, one stop bit and FIFOs.

The collector code that is written in Python is used for interfacing between the microcontroller and the PC. The logic is presented as a flowchart in Figure 6 - Flowchart of the PC collector code. Since the VL53L1X library also uses UART internally to communicate the state of the time-of-flight sensor during initialization, the collector must be notified when the actual distance data is going to be transmitted. When run, the collector waits until it reads the word “start” which is transmitted by the microcontroller right after all the setup code have run successfully.

When the word “start” is received by the collector code over UART, it executes to the data collection logic. Every line of data is transmitted by the microcontroller is terminated with a carriage return (“\r”) character and a line feed (“\n”) character. This way, the receiver can easily track when all the numbers for the current distance measurement is sent. For each line received, the collector first, parses the string as an integer and then, separates the distance data into its Y and Z components according to the formulae 2 and 3. The “distance” is the value received from the microcontroller. The “number of sets” value dictates how many times the system will be moved along the X axis and take a full set of measurements. The “number of measurements” value dictates how many measurements there will be in one set. And the “measurement counter” value keeps track of the current number of measurements in the set.

$$y = \frac{distance}{number\ of\ sets} * \sin\left(\frac{2\pi}{number\ of\ measurements} * measurement\ counter\right) \qquad [2]$$

$$z = \frac{distance}{number\ of\ sets} * \cos\left(\frac{2\pi}{number\ of\ measurements} * measurement\ counter\right) \qquad [3]$$

The X value is calculated using the formula 4 where the “set counter” value keeps track of the current number of sets. The value 50 is arbitrarily chosen relative to the amount the system is moved in along the X axis.

$$x = set\ counter * 50 \qquad [4]$$

The calculated XYZ value is appended with a line break character (“\n”) to the file “data.txt” to be read by the visualizer code. As a result, each line on the file represents a measurement of the spatial area in the form of “X [space] Y [space] Z \n”. To increase the reliability of the file input/output (IO), the file is opened and closed before and after each set respectively.

Visualization

The XYZ data that is prepared by the collector code is read by the visualizer code as outline in Figure 7 - Flowchart of the PC visualizer code. The program uses Open3D library to convert the XYZ data into a point cloud object. While it is possible to present this data directly as points in 3D space, it is not easy to comprehend the end result. Therefore, a line set is created by connecting every point to its adjacent points. This set includes intra-connecting lines in the set and inter-connecting lines between the sets. To improve the visualization of the 3D area further, a boundary box for each set is also drawn onto the final result and connected to each other. This creates a reference frame for the 3D reconstruction and makes it easier to understand the result. The user interface created by the Open3D library also offers functionalities to rotate the model and zoom.

Application Example

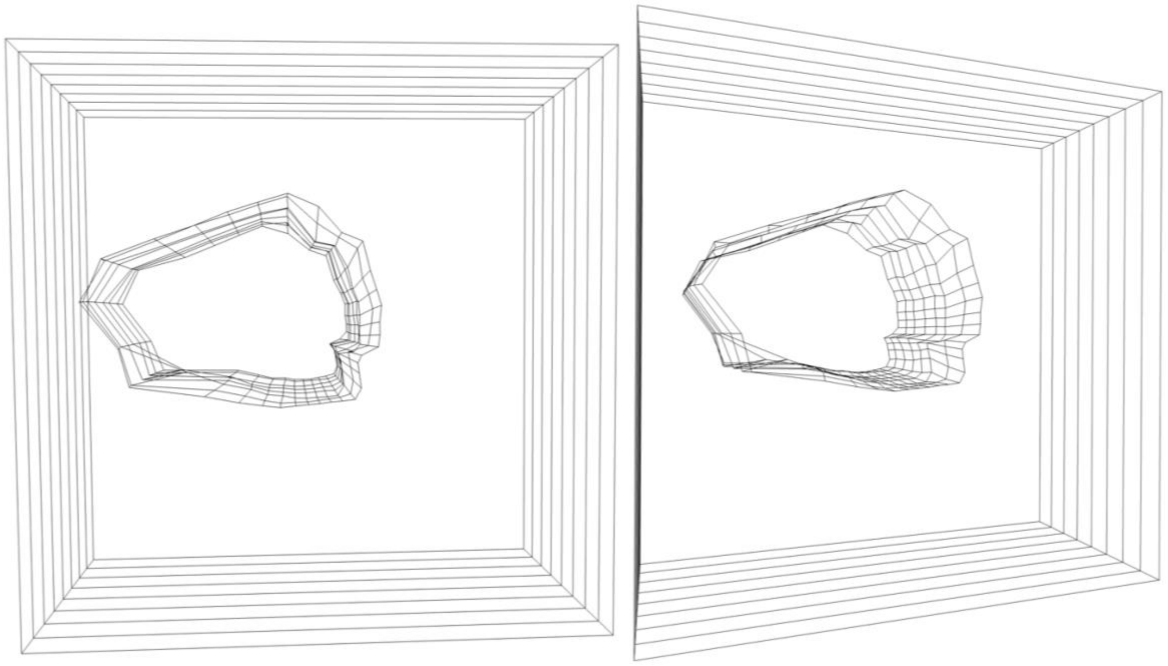
Figure 2 - Reconstruction of ITB hallway

Figure 2 - Reconstruction of ITB hallway shows the reconstruction of the hallway in ITB from different angles. The system is placed closer to the right side of the wall where there is a bench integrated in the wall. For this result, 8 sets of 32 measurements are taken by moving the system forward between each set. In the reconstruction, the bench and its lower ceiling can be seen on the right side.

The device is held so that the top of time-of-flight sensor is facing forward. In that case, the x-axis is located along the extension of the sensor. And the Y and Z axes refer to height and width of the reconstruction.

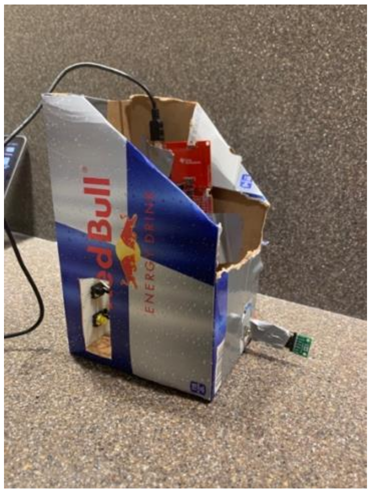
Figure 3 - Photo of the system

User Guide

1- Install Python version 3.9.11 or lower.
2- Install Python packages numpy, open3d and pyserial.
3- Connect the microcontroller to the PC using a USB cable by the micro-USB port beside the reset button.
4- Find the serial port that assigned by the computer (should be in the form of COMX where X is a number) and change the string "/dev/tty.usbmodemME4010231" by that port number on line 18 in the collector.py file.
   NOTE: The code is written to be used by a Linux based operating system and no change is required for these systems.
5- Run the collector.py code.
6- Reset the microcontroller using the reset button.
7- Wait for the "start" print on the terminal or until all the onboard LEDs are turned off.
8- Hold the system stable and press the button connected to the port PH0 to start data acquisition for the current set.
   NOTE: This step can be interrupted anytime by pressing the PH0 button again. This will require a restart of the system, proceed from the step 5.
9- Wait for the current set to be completed by either waiting for the "set-X" print on the terminal or observing the servo to come to a halt and stay stationary for at least 1 second.
10- If the wires are tangled around the sensor, press the PH1 button to untangle the wires.
   NOTE: This step can be interrupted anytime by pressing the PH1 button again.
11- Move the system forwards or backwards for ~20 centimeters.
12- Repeat steps 8 to 11 until all the sets are completed.
   NOTE: The collector code will exit gracefully when all steps are complete.
13- Run the visualizer.py code.
14- Observe the 3D reconstruction created by the system.

Limitations

1- Due to the Open3D library used in the Python collector code, Python versions 9 and lower must be used to run the code.
2- Python float values are stored as 64-bit double-precision values, therefore, the maximum value for a measurement calculation is approximately $1.8 * 10^{308}$ [3].
3- The VL53L1X time-of-flight sensor can only be interfaced using I^2C serial communication protocol which is limited to 400 kHz [2].
4- The VL53L1X time-of-flight sensor can range maximum of 4 meters at a maximum frequency of 50 Hz [2].
5- The maximum quantization error is calculated to be $6.10 * 10^{-2}$ millimeters using the equation 5 where "max value" is 4 meters and "n" is 16.

$$error = \frac{\max value}{2^n} \qquad [5]$$

6- The serial communication with the PC uses UART which has a standard maximum baudrate of 115200 bits per second [4].
7- The stepper motor requires at least 2 milliseconds between each step.
8- The slowest part of the system is the stepper motor, where each rotation takes at least 20 milliseconds.

Circuit Schematic

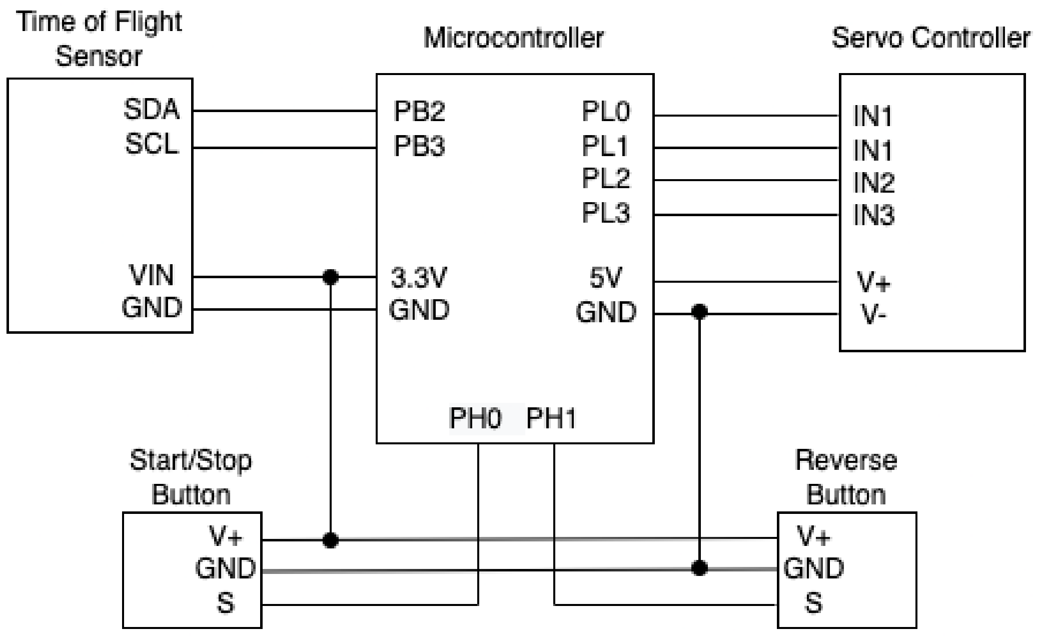

Figure 4 - Circuit schematic of the system

Programming Logic Flowcharts

Microcontroller Code

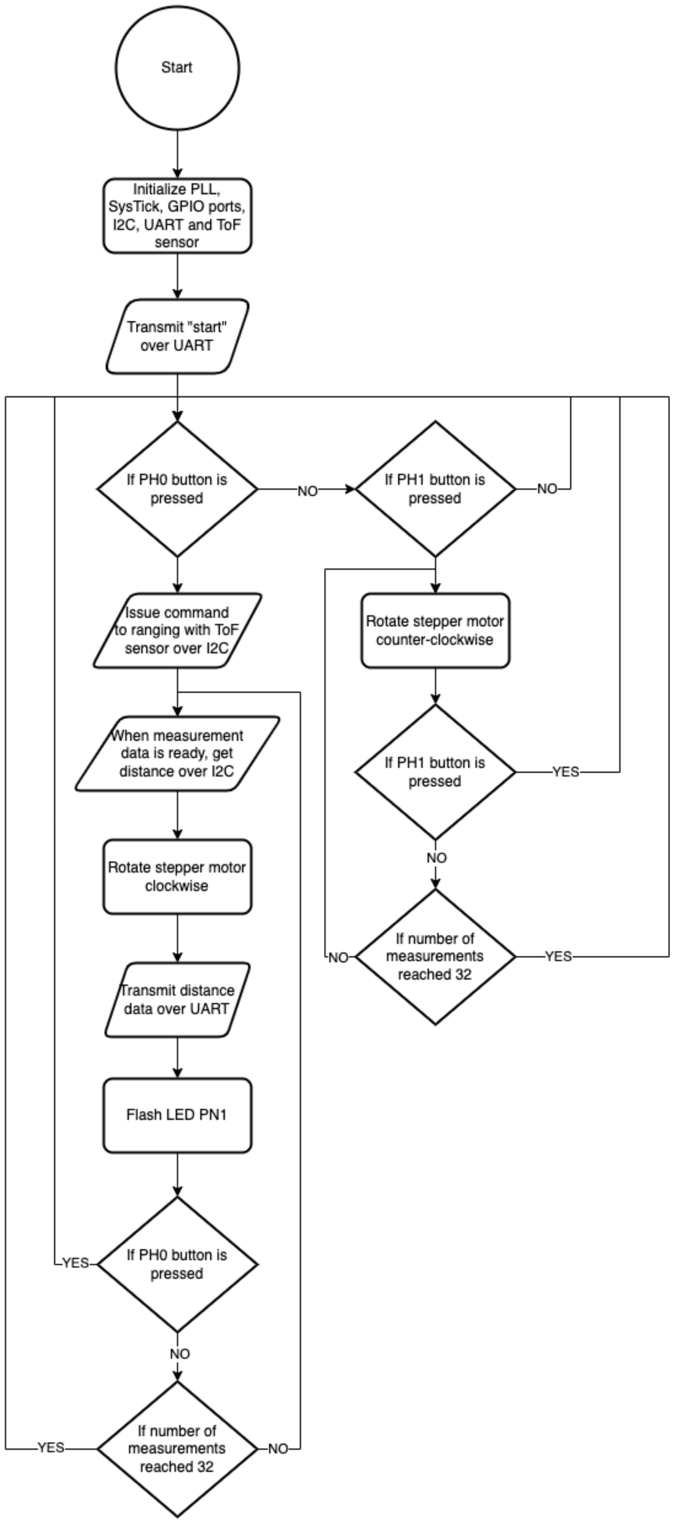
Start
Initialize PLL, SysTick, GPIO ports, I2C, UART and ToF sensor
Transmit "start" over UART
If PH0 button is pressed
NO
If PH1 button is pressed
NO
Issue command to ranging with ToF sensor over I2C
Rotate stepper motor counter-clockwise
When measurement data is ready, get distance over I2C
If PH1 button is pressed
YES
NO
Rotate stepper motor clockwise
NO
If number of measurements reached 32
YES
Transmit distance data over UART
Flash LED PN1
YES
If PH0 button is pressed
NO
YES
If number of measurements reached 32
NO

Figure 5 - Flowchart of the microcontroller code

PC Code – Collector

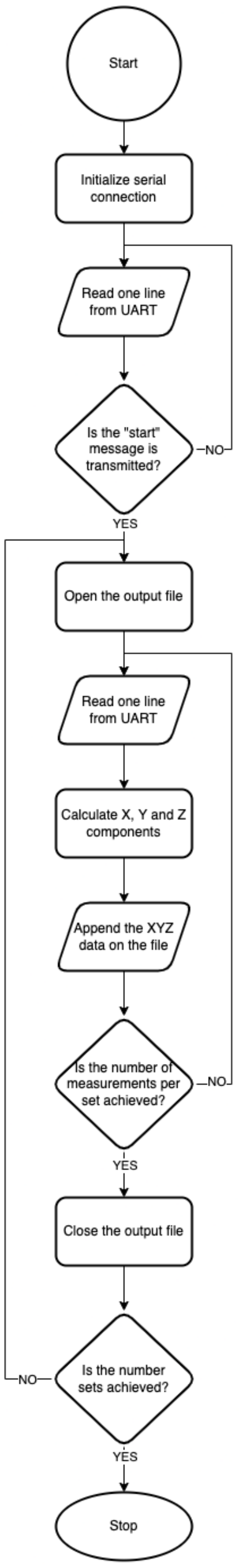
Start
Initialize serial connection
Read one line from UART
Is the "start" message is transmitted?
NO
YES
Open the output file
Read one line from UART
Calculate X, Y and Z components
Append the XYZ data on the file
Is the number of measurements per set achieved?
NO
YES
Close the output file
Is the number sets achieved?
NO
YES
Stop

Figure 6 - Flowchart of the PC collector code

PC Code – Visualizer

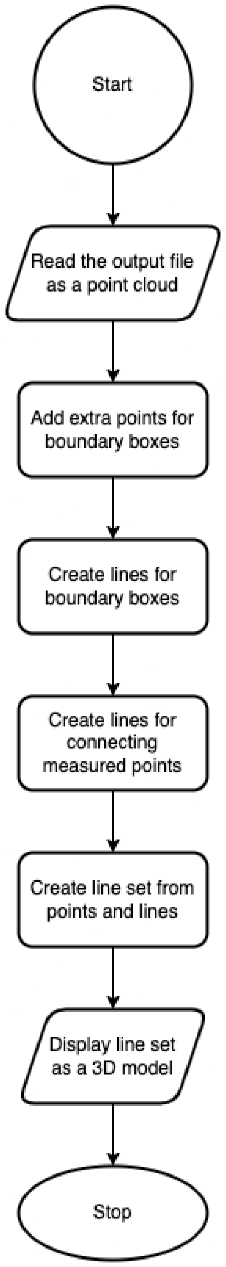
Start
Read the output file as a point cloud
Add extra points for boundary boxes
Create lines for boundary boxes
Create lines for connecting measured points
Create line set from points and lines
Display line set as a 3D model
Stop

Figure 7 - Flowchart of the PC visualizer code

References

[1] “MSP432E401Y Product Details,” *Texas Instruments*. [Online]. Available: https://www.ti.com/product/MSP432E401Y. [Accessed: 11-Apr-2022].

[2] “VL53L1X Datasheet” [Online]. Available: https://avenue.cllmcmaster.ca/d2l/le/content/418585/viewContent/3616160/View. [Accessed: 11-Apr-2022].

[3] “Python: Float type and its methods,” *Geeks For Geeks*, 16-Nov-2018. [Online]. Available: https://www.geeksforgeeks.org/python-float-type-and-its-methods/. [Accessed: 11-Apr-2022].

[4] “Serial Communication,” *Sparkfun*. [Online]. Available: https://learn.sparkfun.com/tutorials/serial-communication. [Accessed: 11-Apr-2022].