COMPENG 2DX3: Microprocessor Systems Project

Project Report

Gurjas Bassi – L01- bassig1 - 400062217

(1) Device Overview

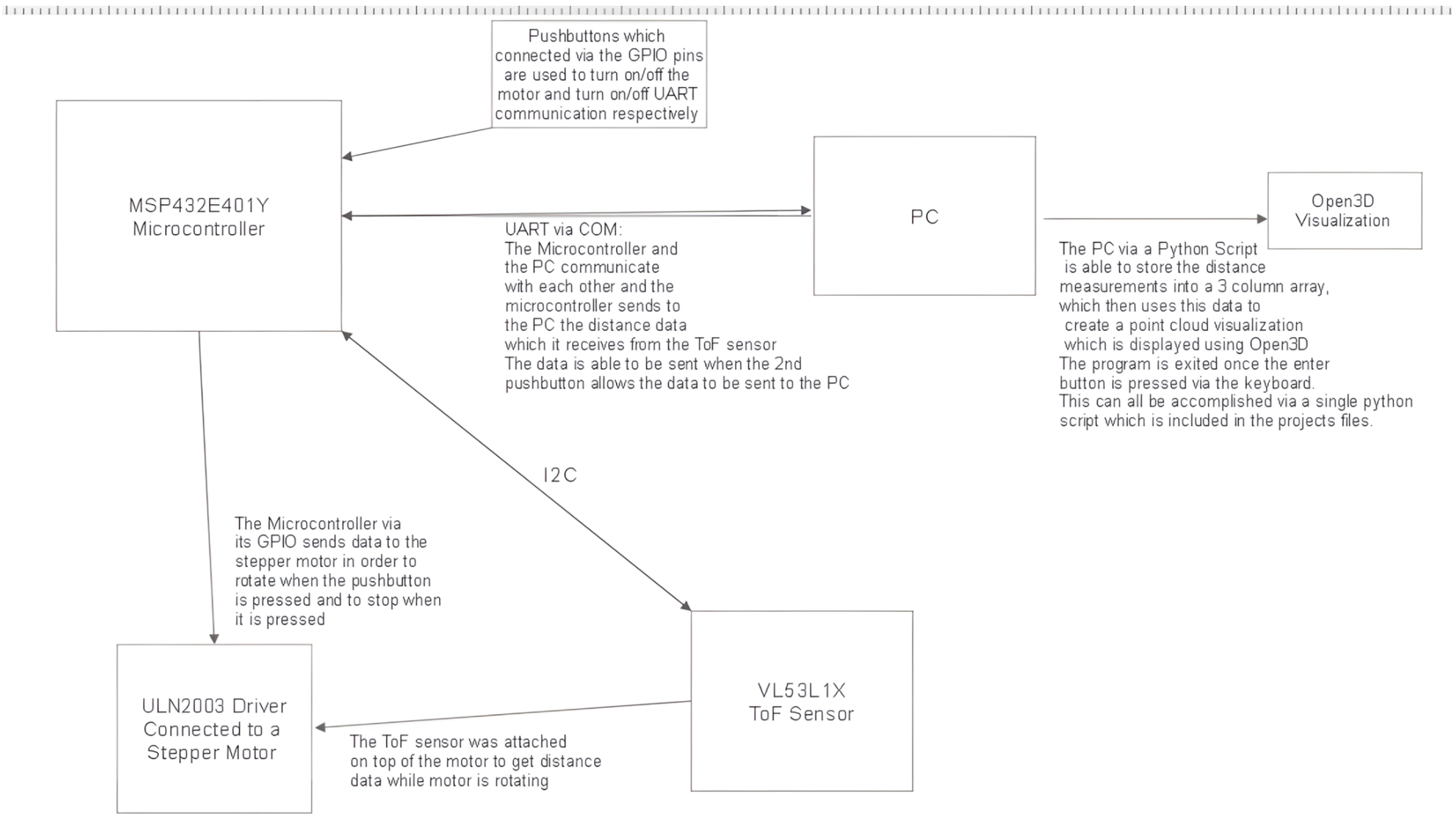
a) Features

- TI MSP432E401Y Microcontroller featuring a 120-MHZ Arm Cortex-M4F Processor Core with FPU that was configured with a bus speed of 80 MHZ. It features 1024 KB of flash memory and 256 KB of SRAM with two 12-bit SAR-Based ADC Modules. Programming for the microcontroller was done in C.
- Pushbuttons: These push buttons were used to start/stop UART communication ensuring data could be sent to the PC along with starting/stopping the stepper motor.
- Stepper Motor: The stepper motor connected to the ULN driver which for a full rotation has 512 steps and is connected to 5V and GND. For the inner rotor the associated step angle is 11.25 degrees.
- ToF: The VL53L1X ToF sensor which is connected to an input voltage of 3.3 V and has a maximum distance in dark conditions of about 360 cm or with TB = 140ms a typical max distance of 4m. The running error for mm in dark or ambient conditions ranges from +- 20mm to +-25mm [3]
- Data Communication: I2C and UART : The microcontroller and ToF module were able to interact via I2C synchronous communication, this distance data was then sent to the PC via UART where it was then stored via a python script and later used for visualization.
- Visualization : Visualization was done using the language Python in which once the data was received from the ToF via UART and stored in a .xyz file visualization would occur using the modules Numpy and Open3D. The visualization was Point Cloud Visualization to graphically recreate the area using distance measurements and a set displacement value. This was done using Python 3.10 and the IDE Visual Studio Code.

b) General Description

The device is an embedded spatial measurement system which uses a ToF sensor to acquire information about the area around it. It features a stepper motor to which this ToF sensor is attached to that allows the ToF to rotate and acquire distance data using a fixed displacement value. It gets this distance measurement using the ToF principle in which the measured distance is the photon travel time/2 * speed of light. Using trigonometric identities it converts this distance data into y and z values in order to create a .xyz file which features an array in order to visualize the mapped spatial information. Using a python script and the data that was acquired with the Numpy and Open3D modules a reconstruction of the Area is able to be shown graphically.The data is transmitted from the ToF to the microcontroller with I2C serial communication and then to the PC via UART from the microcontroller to the PC. Both the stepper motor and UART communication can be started and stopped via push buttons which are connected via the microcontrollers GPIO.This can be compared to commercial LIDAR systems which also use light to detect ranges. Images of the device are featured in the appendix.

c) A data flow graph is a representation on how data flows through a system, below is the block diagram


Pushbuttons which connected via the GPIO pins are used to turn on/off the motor and turn on/off UART communication respectively
MSP432E401Y Microcontroller
PC
Open3D Visualization
UART via COM: The Microcontroller and the PC communicate with each other and the microcontroller sends to the PC the distance data which it receives from the ToF sensor The data is able to be sent when the 2nd pushbutton allows the data to be sent to the PC
The PC via a Python Script is able to store the distance measurements into a 3 column array, which then uses this data to create a point cloud visualization which is displayed using Open3D The program is exited once the enter button is pressed via the keyboard. This can all be accomplished via a single python script which is included in the projects files.
I2C
The Microcontroller via its GPIO sends data to the stepper motor in order to rotate when the pushbutton is pressed and to stop when it is pressed
ULN2003 Driver Connected to a Stepper Motor
VL53L1X ToF Sensor
The ToF sensor was attached on top of the motor to get distance data while motor is rotating

(2) Device Characteristics Table

Characteristic	Description
MSP432E401Y Microcontroller	Microcontroller used to communicate with PC, Stepper Motor, ToF sensor and pushbuttons. The bus speed is 80 MHz this was done by changing PLL.h, with the serial port being COM7 at baud rate of 115200 bps with the COM port being PC specific.
Pushbuttons	These will be used to turn on/off the stepper motor and UART communication, the pins used are shown below with 3.3V and GND also being used

ULN2003 Driver with Stepper Motor	RB-Vel-133 which will be used to rotate the ToF sensor, the pins used are shown below with 5V and GND also being used
VL53L1X ToF sensor	Polo 3415 (VL53L1X Time-of-Flight Distance Sensor) which using the time that is needed for light to bounce from the sensor and return can determine distance, the pins used are shown below with 3.3V and GND also being used
Software	Keil uVision 5 which will be used with the microcontroller, and Python with the following modules serial, math, keyboard, numpy and Open3D being imported

Port	Pins	Application
Port B	3:2 PB3(SDA), PB2(SCL)	I2C
Port E	0	Port used to derive PE0 low to detect if a pushbutton is pressed
Port M	1:0	Ports used for the pushbuttons
Port G	0	The VL53L1X needs to be reset using XSHUT. We will use PG0
Port N	1:0	The PN0 is used as the measurement status and flashes when measurement is taken, while PN1 LED is used to flash when the motor has completed a 45 degree turn in order to keep track of how many times motor has done a rotation
Port H	3:0	This is used for the stepper motor to complete a full rotation
Port F	4,0	These LEDS are used in conjunction with Port N LEDS to flash all of them at once to show ToF is ready

(3) Detailed Description

a) Distance Measurement

The microcontroller communicates with the ToF sensor with the microcontroller having a bus speed of 80 MHz. The VL53L1X ToF sensor is used to take distance measurements, in conjunction with the stepper motor this ToF sensor sends the distance value to the microcontroller via I2C serial communication and in the microcontrollers C code via line status = VL53L1X_GetDistance(dev, &Distance); Once this occurs the user LED flashes to indicate that the distance measurement has been sent. After this the interrupt is cleared to be called to enable the next interrupt and then the resulting distance reading is printed to the UART to allow for processing via the PC which has UART asynchronous communication with the microcontroller. This ToF sends the distance measurements every 1/512 steps for the stepper motor which allows for a full rotation once 512 steps and 512 distance measurements have been sent. As discussed earlier the way the ToF sensor is able to determine the distance is by having two sensors, in which one is used to emit the laser and one to receive it, the distance is then determined via the photon travel time/2 * speed of light. To allow for sending of data via UART to be enabled along with the rotation of the stepper motor push buttons were used for each of these cases. These pushbuttons were configured with start/stop in mind where pressing the button once starts the process and pressing the button another time stops the process. This all occurs in the while loop and incorporates the polling method that constantly checks if an input is pressed. To allow for the accurate distance measurements to be sent the pushbuttons should both be pressed in order of UART pushbutton, stepper motor push button once the python script is already running. Even though the pushbuttons use a polling method the ToF uses an interrupt based method where once the distance data is sent to the microcontroller the interrupt is cleared and this cycle repeats. Everytime data is sent from the microcontroller to the PC via UART the user LED which was set in the project requirements flashes on the microcontroller to indicate this is happening.

An ADC is not required as the ToF module provides a digital signal and has internal modules for this purpose and gives a digital signal for which the microcontroller is able to send to the PC via UART. The ToF is configured using its default configurations and with API functions such as BootState,SensorInit,StartRaning,GetDistance,ClearInterrupt. Thes default configurations were provided in studio code. These API function calls are necessary for proper function of the ToF sensor to retrieve distance data which will then be sent to the PC via UART. The stepper motor and ToF's purpose is to get the distance data while the ToF is rotating and to send this to the microcontroller which then via UART sends this data to the PC where it will be processed and stored into an array. A more clear example can be shown in the block diagram or data flow graph where communication via the stepper motor, microcontroller and ToF sensor can be shown and a visual representation of data flow is able to be seen.

The distance measurement section the distance measurement is separated into y and z axis using trigonometric identities while the x value is the displacement which for the case of the hallway which was to be scanned after each full rotation increases by a meter so the whole system should be moved a meter once a full rotation has occurred. These values of displacement were chosen due to the length of the hallway that was being examined. Although this gives the x value in our array where the data is stored this does not give the y and z values, the y and z values are found from the distance measurement that is sent to the PC via UART. The y and z values were then calculated using distance*cos((2*pi)*((i)/512)) for y and distance*sin((2*pi)*((i)/512)) for z respectively. Using trigonometric identities this is how the y and z components were found to be calculated in order to get accurate distance measurements and the angle is also found using i which is the step that the motor is currently on so this can accurately depict any angle i/512 for 360 degrees with i=512 being 360 degrees. 512 measurements are taken per rotation to allow for very accurate visualization later on. The raw distance value which will then using the methods above be converted to the y and z values needed for the .xyz file is sent via the sprintf(printf_buffer,"%u\r\n", Distance); UART_printf(printf_buffer);.An example of this data being stored is found in the data.xyz file which is included with the projects code in the python folder. To show the data along with storing it in a .xyz file the python script Hallway.py must be running and when enter in the terminal is pressed the script is terminated the data is stored in the .xyz file and visualization begins this is all done in a singular script with another .py file which can be used to just show visualization.

b) Visualization

The visualization process occurs on the PC via a python script which will be running while the microcontroller is able to send distance measurements to it via UART. The program for visualization and storing data were combined to a single program which was preferred for the project. Once the python script is running, it receives the distance measurements and displacement measurements and stores them in an array. As discussed in the distance measurement section the distance measurement is separated into y and z axis using trigonometric identities while the x value is the displacement which for the case of the hallway which was to be scanned after each full rotation increases by a meter so the whole system should be moved a meter once a full rotation has occurred.

The python script handles converting the distance to y and z with y = distance*cos((2*pi)*((i)/512)) with i being the number of steps that has occurred to determine which part of the motor it is on, if it is the 512th step this is a full rotation so the value would be cos(2*pi) for example. This is very similar for z where the z value is z = distance*sin((2*pi)*((i)/512)) . After a full rotation of the stepper motor this i value resets to 0 for 0 degrees. The python script uses modules such as math, keyboard and serial. The math modules allow for the y and z values to be calculated properly, while the serial module allows for communication via the PCs port and the microcontroller in this case it was COM7. The keyboard module is used to stop the script from acquiring distance data via UART and start visualization using Open3D. Once the enter button is detected on the PC via the terminal of the python script the distance data is stored in a .xyz file where visualization takes place and a point cloud representation of the data is made.

The data.xyz file which was created in the same script is now opened and is shown graphically using the Open3D library, using o3d.visualization.draw_geometries([pcd]). This function built into the Open3D library takes a list of Point Cloud objects and visualizes it. Then connecting the vertices and connecting lines in each y-z slice using line maps the point cloud data is shown graphically with lines as well. The visualization of the hallway is shown in the Application Example via figure #2. This was done using Python 3.10 and the IDEVisual Studio Code with the data obtained being stored in data.xyz and the entire program to collect data and visualize the script called Hallway.py, with just the visualization using the data that was already being acquired being script O3DHallwayOnly.py.

(4) Application Example, Instructions, and Expected Output

To operate the system and the application which is an embedded spatial measurement system which uses a ToF sensor to acquire information about the area around it the following instructions guide was developed and features several parts including initializing software, setting up the device and finally running the device along with the software in order to spatial map an area in 3D via visualization. An example of the required hallway is shown at the end of the application section.

Setup Guide:

Software:

- The software that is to be installed prior to usage is shown below
- A MDK for the microcontroller MSP-EXP432E401Y, Keil uVision is to be installed from the following resource https://www2.keil.com/mdk5
- From this the device family pack for the microcontroller must be installed along with the XDS110 Debugging Unit via https://software-dl.ti.com/ccs/esd/documents/xdsdebugprobes/emu_xds_software_package_download.html
- For the visualization and data processing which occurs via Python, Python must be installed. Testing was done with Python 3.10 and output correct results.
- An IDE was used as the file open and close operations for python did not work using IDLE as suggested so running all python scripts in Visual Studio Code seemed to have the best results using a ThinkPad Laptop, reasons for this could have been due to having dual drives resulting in some errors. Python must be added to the path in the windows settings and setup for python in Visual Studio Code can be found https://code.visualstudio.com/docs/python/python-tutorial.
- Once this is complete several modules/libraries in python must be installed these include serial, and keyboard which can be installed in cmd as long as path is set by : pip install keyboard, and by pip install serial. For visualization the libraries of numpy and Open3D

must also be installed using the same methods as pip install numpy and pip install Open3D respectively. The pyserial API must also be installed using pip install pyserial.

- Once complete the following command in command prompt can show which port is available for user UART python3 -m serial.tools.list_ports –v. In my case my user UART is 'COM7' and this is what was used in the python script, for your specific user this must be edited via the IDE to your specific user UART.
- This is all the software that is needed for the system to work as intended.

Physical Hardware:

- The device must then be setup according to the circuit schematic featured in section 6 of this technical datasheet, once these connections are secure and can be shown via figures #5,6 and 7 in the appendix.

Using Device and Visualisation:

- Once the device is set up physically according to the circuit schematic with the ToF sensor mounted to the stepper motor operating the device can begin.
- Connect the microcontroller to the PC via the USB port then using the following command in cmd, ensure the correct port is being used for UART, python3 -m serial.tools.list_ports –v..
- Edit the python script to ensure the users correct COM# port is used by editing this line s = serial.Serial('COM7',115200, timeout = 10) in the Hallway.py file.
- Launching the program Keil uVision and in the folder 2DX_2023_Studio 8 open the project file 2dx_studio_8c.uvprojx. Once this is complete ensure once again that all necessary connections are made in the system and the microcontroller is connected to the PC's usb port.
- Once this is done translate the code, build the code and download it to flash memory, this can be done by pressing in order Ctrl+F7,F7,F8 once the uVision MDK recognizes the microcontroller.
- After this is done press the Button/Switch Reset, this is crucial to ensure only distance will be sent and not print messages for example like botting the ToF chip which will cause the python script to malfunction.
- The program uses an initial displacement of 0 m and after a full rotation of the motor 1m displacement is used, this can be edited also by editing the python script in an IDE.
- Now preferably using the Visual Studio Code IDE run the python script labeled Hallway.py pressing F5.
- Once this is done press the pushbutton to which Port M1 is connected to which should allow for serial communication to begin and immediately after press the push button which is connected to Port M0 to allow the stepper motor to rotate.
- The device should now be printing the distance data on the python scripts terminal and will be storing this into a 3D array in which it has the x value which is displacement, the

y value which was found using trigonometric identities and the z value which is found using trigonometric identities.

- Once you are satisfied with the data that will be saved to the .xyz file to stop the script press enter on your keyboard on the python terminal as this will exit the while loop and start visualization.
- Once this is done you can now press the push buttons again to stop UART serial communication between the microcontroller and PC and the stepper motor.
- Once this is complete using the Open3D library and numpy these data points will be represented by a point cloud object and visualized. Once this window is closed now a graphical representation of the point cloud with connected vertices and connecting lines in each y-z slice using line maps the point cloud data is shown graphically.

Visualization Only:

- To only see visualization once the data set has already been created, in the IDE open the file O3DHallwayOnly.py which will open the Open3D window and just like previously these data points will be represented by a point cloud object and visualized. Once this window is closed now a graphical representation of the point cloud with connected vertices and connecting lines in each y-z slice using line maps the point cloud data is shown graphically.

The figures shown below are the hallway which was to be scanned corresponding to the student number and the hallway recreated with a point cloud representation. 3 rotations were done with a total displacement of 2m taking place, the reason the visualization does not exactly look like the hallway is due the amount of rotations that were the done the wires were in the way at some points for the ToF, if more time was given a 3D printed apparatus would not of had this issue.

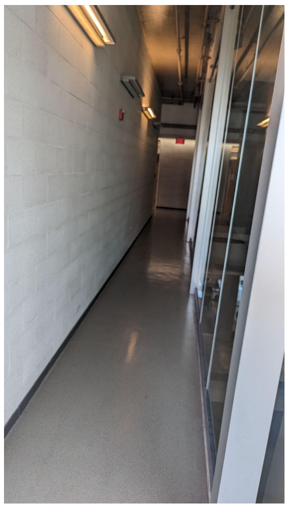


Figure #1 Hallway which was to be scanned

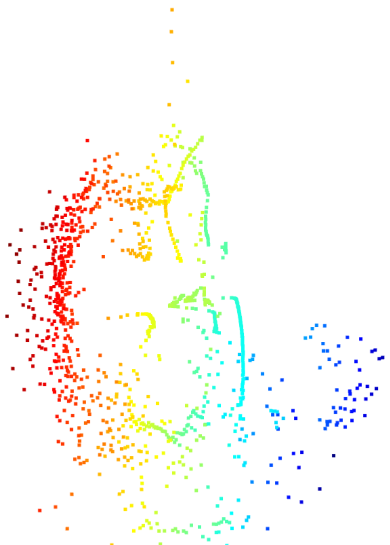
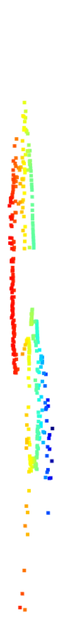


Figure #2 a front view of the hallway



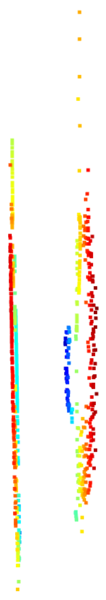


Figure #3 a side view of the hallway showing the displacement without vertices connecting

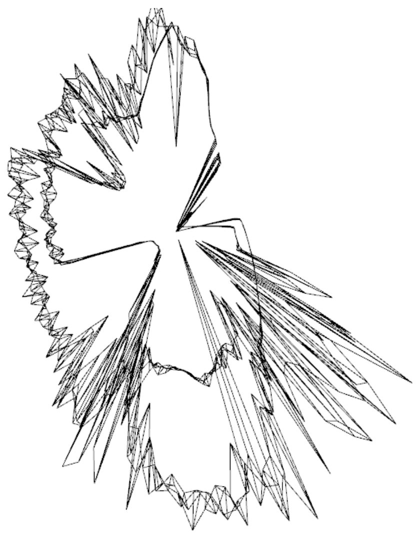


Figure #4 The hallway with the vertices connected.

(5) Limitations

(1)

Using a microcontroller such as ours in regards to floating point capability there may be some errors due to the relative strength of the processor and memory the microcontroller has compared to an average computer. In regards to floating point units, 32 bit instructions are used for single precision data-processing operations [2]. This should not cause any issues for our project as 32 bit instructions for floating point numbers should be more than enough precision. The limitation of trigonometric functions however does not apply to our project since the distance value is transmitted to the PC via UART, the python script handles the use of trigonometric functions to determine the y and z values. These operations were used when the math module was imported into the python script.

(2)

Although according to the datasheet the running error for mm in dark or ambient conditions ranges from +- 20mm to +-25mm [3] The maximum quantization error as discussed in lectures can be found with VFS/2^n for each module depending on the number of bits that are required. With the VFS being the full scale voltage in this specific part. For total running a single byte will not fit this so 16-bit distance reading would be needed to store the distance measurement. The maximum quantization error would then be 3.3V/2^16 = 0.050354 mV.

(3)

The maximum serial communication rate which can be implemented with the PC is 115200 bits/s; this is the baud rate of 115200. This was confirmed via the labs and testing using software such as Realterm via the COM port. This speed was also implemented and again verified via Realterm and python with the line being used for UART being serial.Serial('COM7',115200, timeout = 10) to ensure this was the serial communication rate that will be used.

(4)

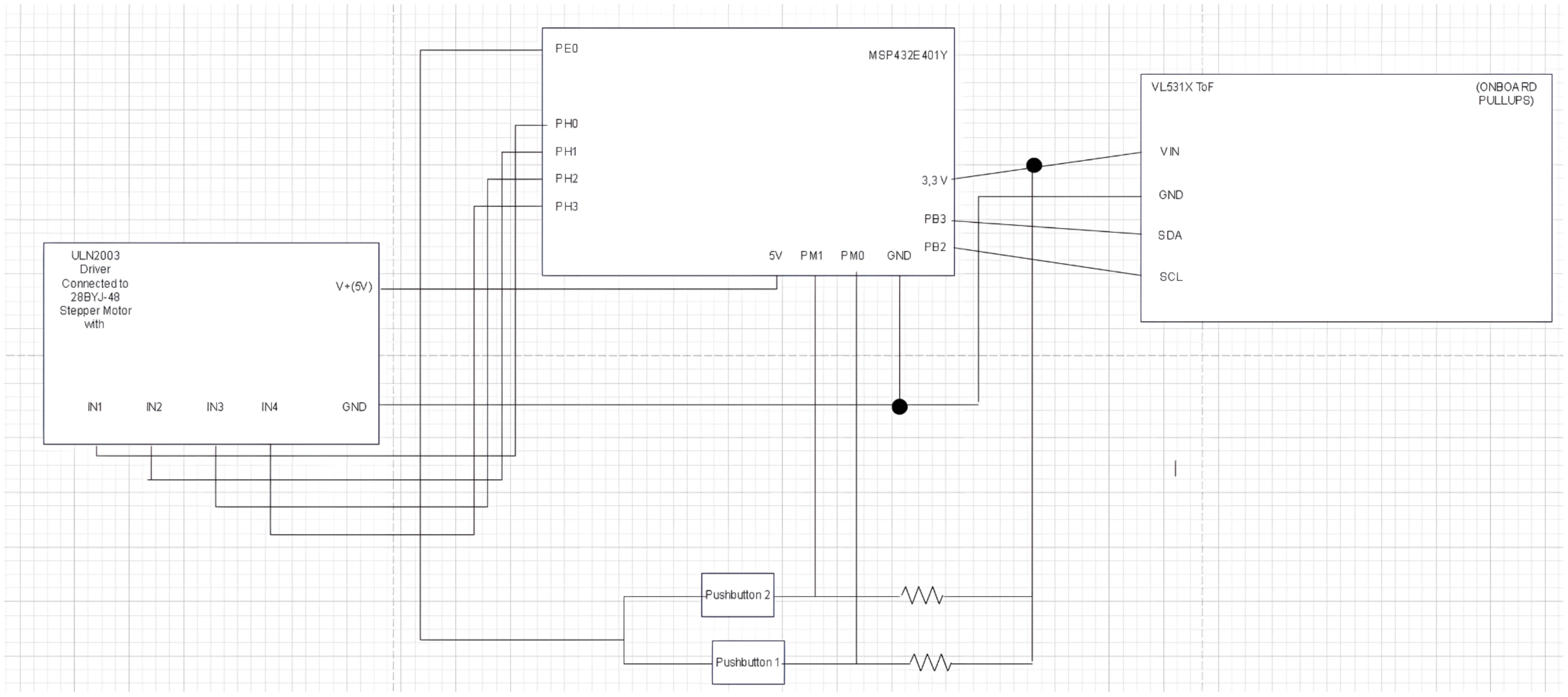
The ToF sensor and the microcontroller communicate to each other via I2C serial communication and was configured with the project for a 100 kbps clock. This can be shown in the C project code with line I2C0_MTPR_R = 0b00000000000000101000000000111011;

(5)

The element that is the primarily limitation on the whole system's speed would have to be the ToF sensor as the bus speed of the microcontroller was set to 80 MHz still the maximum speed at which data was able to be transmitted via the I2C bus is still 400 kbit/s [3]. Again the ToF via I2C was configured to run at 100 kbit/s so this was the primary limitation on the system speed. This was tested by reducing the delay that the stepper motor receives to much lower than the 10ms it is currently set at and the ToF sensor would not be able to accurately get measurements fast enough . This was also tested by moving an object very fast in front of the

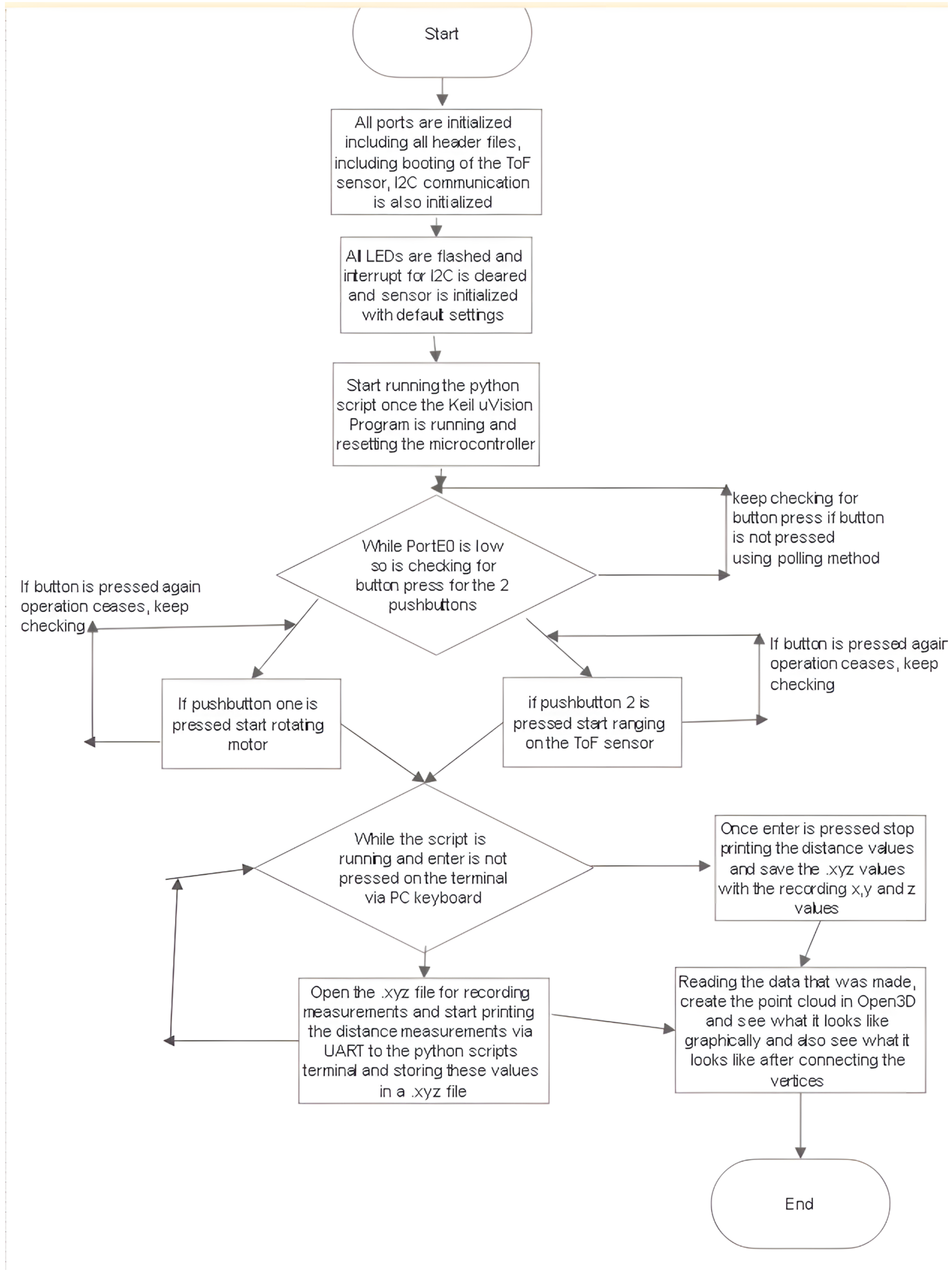
ToF and the ToF would miss some of these measurements; this was mainly tested in the labs. The timing budget is 100 ms for the ToF sensor.

(6) Circuit Schematic



This circuit schematic shows how the device should be connected along with all the pins that are necessary on the microcontrollers GPIO, note the ToF sensor should be mounted on the stepper motor.

(7) Programming Logic Flowchart(s)


Start
All ports are initialized including all header files, including booting of the ToF sensor, I2C communication is also initialized
All LEDs are flashed and interrupt for I2C is cleared and sensor is initialized with default settings
Start running the python script once the Keil uVision Program is running and resetting the microcontroller
While PortE0 is low so is checking for button press for the 2 pushbuttons
keep checking for button press if button is not pressed using polling method
If button is pressed again operation ceases, keep checking
If pushbutton one is pressed start rotating motor
if pushbutton 2 is pressed start ranging on the ToF sensor
If button is pressed agair operation ceases, keep checking
While the script is running and enter is not pressed on the terminal via PC keyboard
Once enter is pressed stop printing the distance values and save the .xyz values with the recording x,y and z values
Open the .xyz file for recording measurements and start printing the distance measurements via UART to the python scripts terminal and storing these values in a .xyz file
Reading the data that was made, create the point cloud in Open3D and see what it looks like graphically and also see what it looks like after connecting the vertices
End

References and Appendix

[1] “2022_2023_2DX3_Project_Specification”, COMPENG 2DX3: Microprocessor Systems Project, Department of Electrical and Computer Engineering, McMaster University, Winter, 2023

[2] “MSP432E401Y - Microcontroller Data Sheet”, COMPENG 2DX3: Microprocessor Systems Project, Department of Electrical and Computer Engineering, McMaster University, Winter, 2023

[3] “vl53l1x”, COMPENG 2DX3: Microprocessor Systems Project, Department of Electrical and Computer Engineering, McMaster University, Winter, 2023

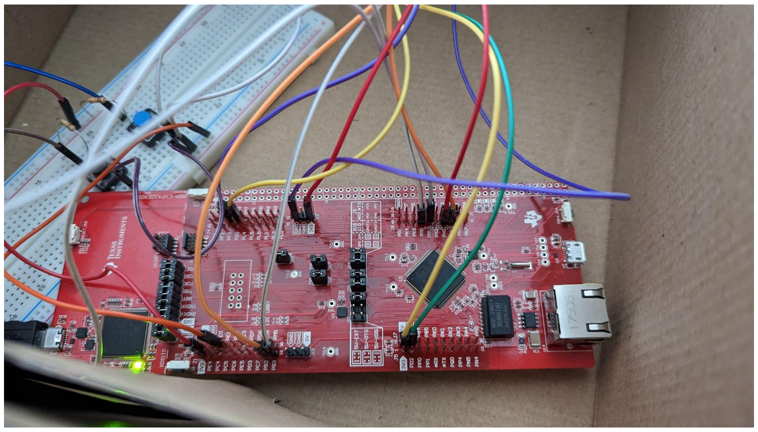
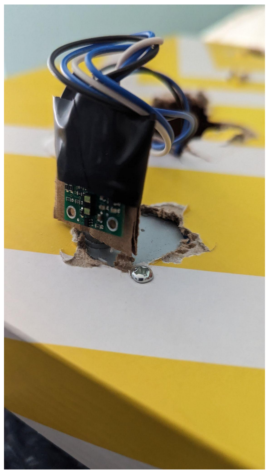
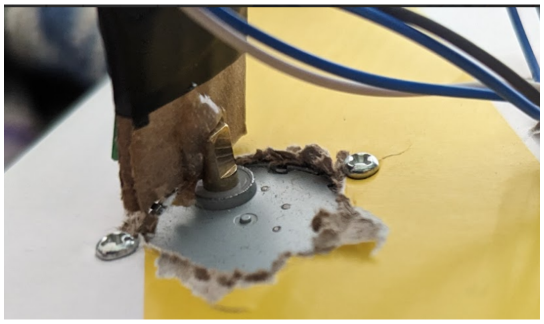


Figure #5 which shows the device connected along with pushbuttons.



Figures #6 and #7 which shows the ToF sensor mounted on top of a stepper motor. Using electrical tape, the ToF sensor was attached onto a piece of very hard cardboard which then via a glue gun was attached to the stepper motor allowing for stable mounting.



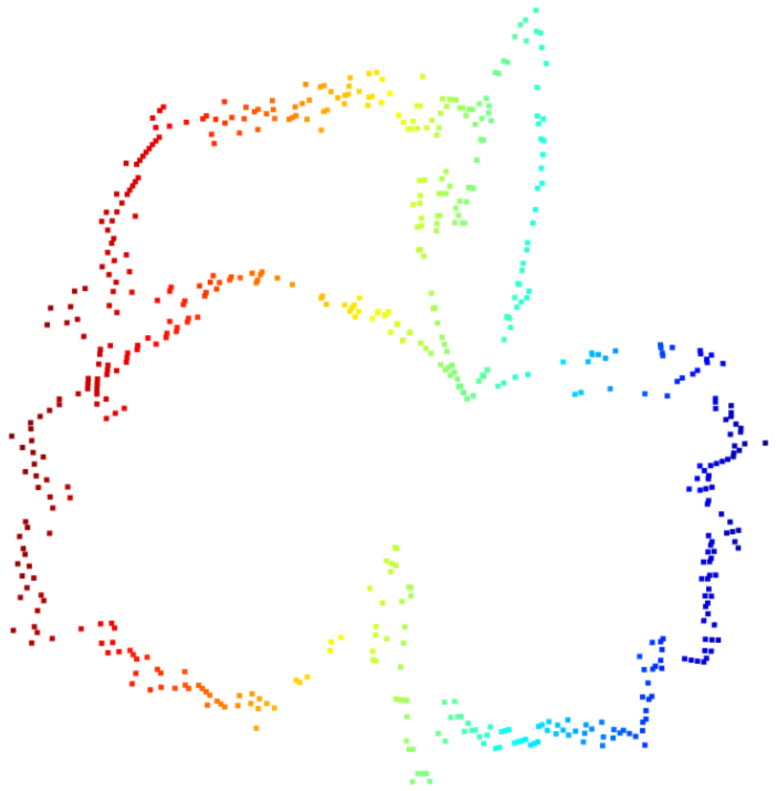


Figure #8 Visualization for the Demo which placed the ToF and Stepper Motor under a cup, the reason while it is not perfectly symmetrical is due to the cup having no openings on the bottom so due to this the wires connected to the ToF ended up tanging causing not a perfect circle otherwise this should have occurred.