2DX4: Microprocessor Systems Project

Final Project

Instructors: Drs. Doyle, Haddara, and Shirani

Date: 15th April, 2021

Harsahib Matharoo – L01 – matharoh - 400185871

As a future member of the engineering profession, the student is responsible for performing the required work in an honest manner, without plagiarism and cheating. Submitting this work with my name and student number is a statement and understanding that this work is my own and adheres to the Academic Integrity Policy of McMaster University and the Code of Conduct of the Professional Engineers of Ontario. Submitted by **[Harsahib Matharoo, matharoh, 400185871]**

Video Links for the demo and the design questions are below:

Full Demo Video (google drive link) is given below:

https://drive.google.com/file/d/179DB-6s0NNZUi09Hhb2SFIURe5O5TWwN/view?usp=sharing

Question 1 google drive video link is given below:

https://drive.google.com/file/d/14q8ho0a4Z67w2JC8b3p1Al9H4Hzu6KDO/view?usp=sharing

Question 2 google drive video link is given below:

https://drive.google.com/file/d/1Q7_gdzf-I2tizgDEn5FDWfXi6oGlwr3B/view?usp=sharing

Question 3 google drive video link is given below:

https://drive.google.com/file/d/1Cwy1_mvATtyiI0AdgKcMxBsd21lsWd_h/view?usp=sharing

Device Overview

Features

- The device which costed an estimated $125 - $130 consists of a: MSP432E401Y SimpleLink™ Ethernet Microcontroller (which operates at 5V DC), VL53L1X Time-of-Flight (ToF) distance sensor (which has already been discussed about), breadboard, single red LED, 28BYJ-48 stepper motor, push button, 10 kilohm and 330-ohm resistors, and many wires.
- Texas Instruments MSP432E401Y SimpleLink™ Ethernet Microcontroller : Has an ARM-Cortex M4F Processor Core, 12 MHz of clock speed, exactly 256 KB of Static RAM memory and 1024 KB of flash memory. Also consists of 4 on-board LEDs for status signaling and other indicators. Consists of Ports A-Q, with each consisting of several pins for building relevant circuits and connections for I/O of data. The microcontroller located on the device consists of: 1024KB of Flash memory, 256KB of SRAM (Static random-access memory), 6KB EEPROM.
- VL53L1X Time-of-Flight (ToF): accurate ranging up to a total of 4 meters of distance in any given area or open space. It has up to 50 Hz of ranging frequency. It features a marvelous 940 nm Class 1 laser, which is almost invisible, safe, and due to its small size and enhanced construction. It has a 2.6 till 5.5 V of operating voltage.
- The rotation speed and the motor control are acquired by the 28BYJ-48 Stepper Motor, which is supported by the ULN2003 Driver, to drive the motor, using a supply of 5 volts.
- uC Bus speed of 120 MHz,12-bit ADC. ARM – Cortex M4F 32-bit C language is used.

General Description

The ToF and the uC communicate using the I2C protocol. On the other hand, the communication between the personal computer and the microcontroller is aided by the UART system. The 2DX4 Final Project, which consists of an integrated 3D scanning system, that can as far as this device is concerned it is a cost-efficient device, by which it is commonly known. It uses the VL53L1X Time-of-Flight (ToF) distance sensor to detect and gather the information about the area surrounding it. LIDAR uses light that is used in the form of laser pulses, and many distinct sensors located on the printed circuit board to measure the distances to the relevant object in the surrounding space. Along with the sensor is a stepper motor, therefore, the sensor provides mapped spatial data and series of points allocating the data in a 3D surrounding space. After the data gathering is done, the light pulses and other forms of data recorded by the system aim to form a 3D representation of the acquired data, from xyz. The transducer converts nonelectrical signal to electrical. Signal conditioning ensures the move to the digital format. ADC conversion uses the discrete values and converts to the digital format.
There is a status onboard LED flash, which acts as an indicator. The push button provides control over the motor rotation and measurements. The ToF sensor located on the motor's shaft works by sending out a photon to an object, which is then reflected, by using the amount of time that it took to reflect back, a distance is calculated in the units of mm. This data is then sampled at a fixed interval, over which each cycle the signal is converted into a digital value which is represented by 12 bits. The actual data representation takes up 8 bits.

Figure 1, 3D Visualization in MeshLab 2020:

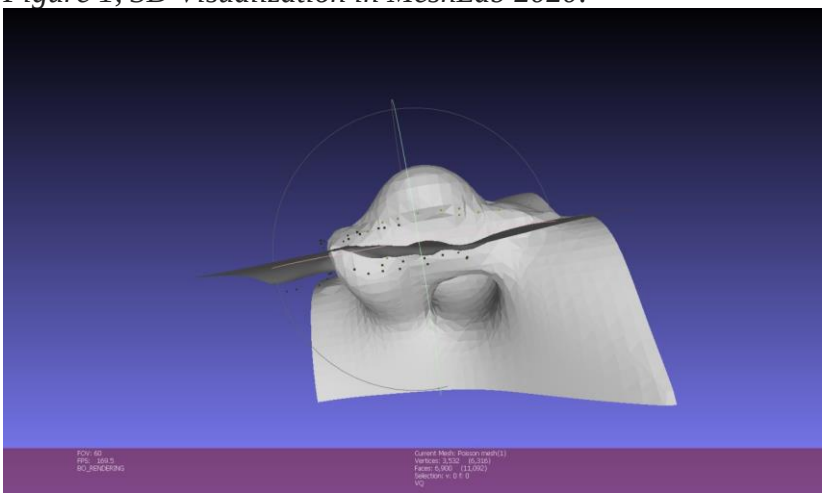
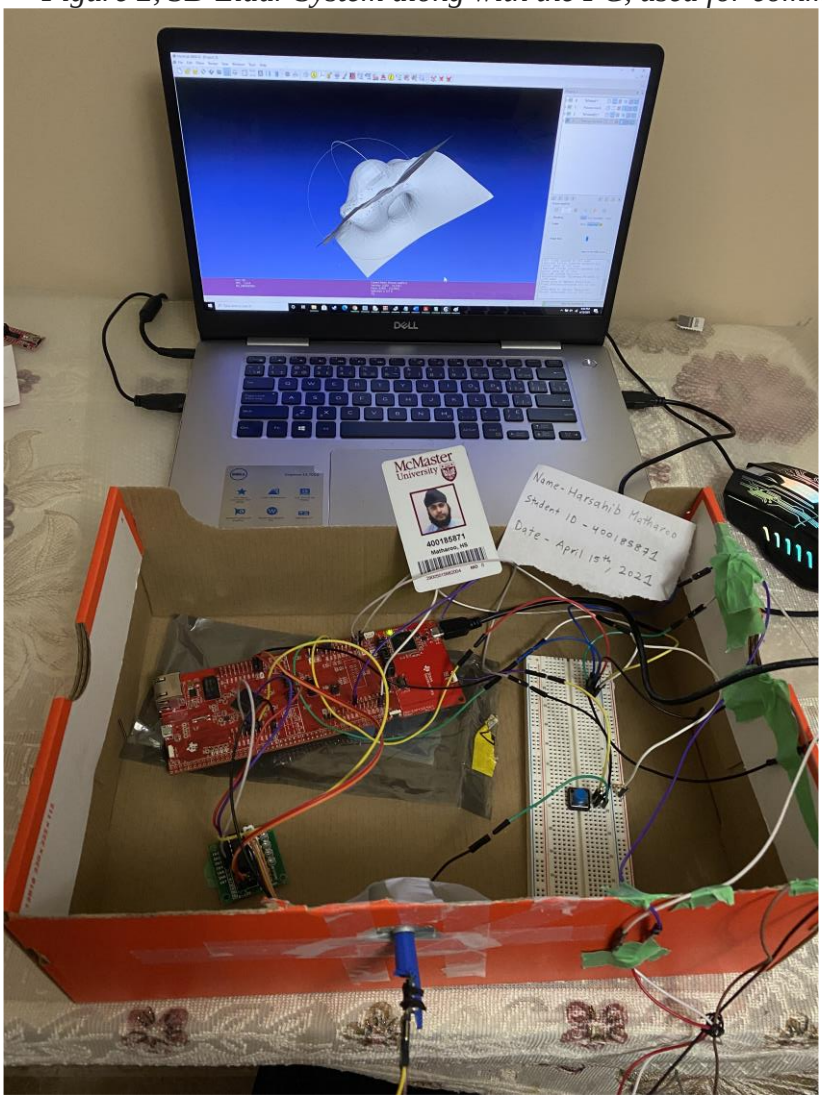


Figure 2, 3D Lidar System along with the PC, used for communication and Visualization:



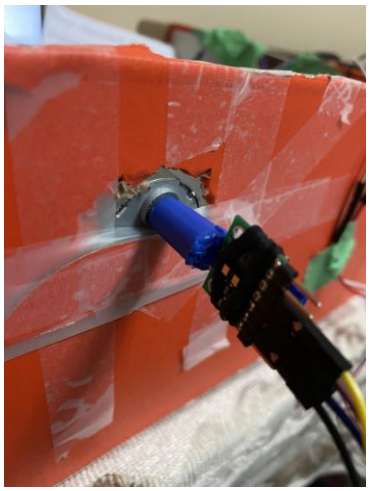


Figure 3, The Block Diagram is shown below, for the 2DX4 Final Project:

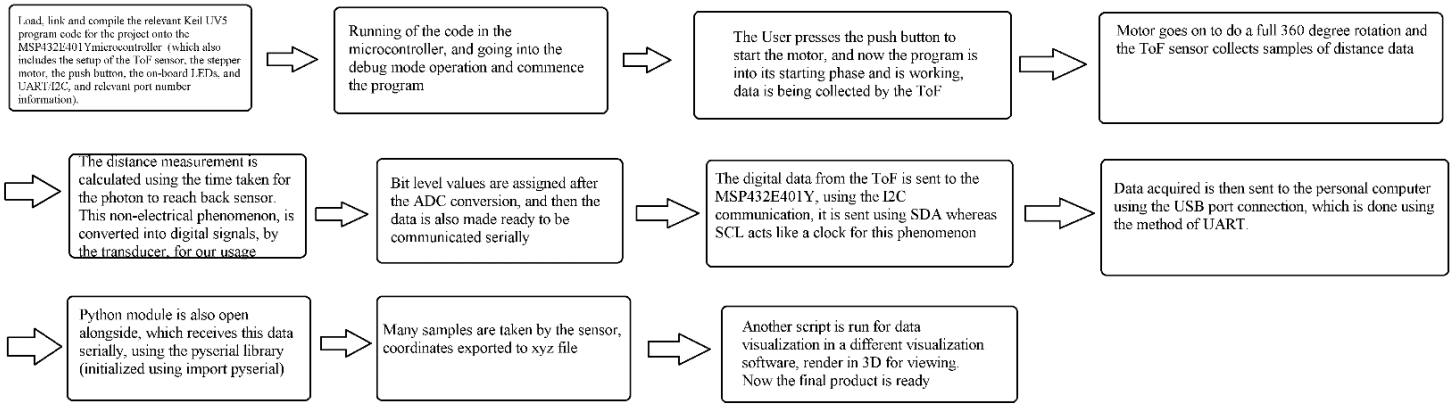


Table 1, Device Characteristics Table:

Device Specification	Summary
Bus Speed	96 MHz
Serial Port	COM3 (Picked for usage) • Serial port was port 3 which is the microcontrollers USB Port. • Speed of 115200 bps (baud rate)
Python Version (used)	3.8.3
VL53L1X Pin Mapping	- VIN is connected to 3.3 V on the microcontroller. - GND is connected to GND on the microcontroller. - SDA is connected to the PB3 pin. - SCL is connected to the PB2 pin.
ULN2003 Stepper Motor Driver Mapping	IN1 – PM0 IN2 – PM1 IN3 – PM2 IN4 – PM3 (-) to GND (+) to 5V
Port and Pin, I/O	• PF4 - onboard LED as an output (distance status as per project requirements) • PL0 - external LED (displacement status as per project requirements), connected to the push button. • PE0 – input from the push button • PM0 - PM3 - stepper motor rotation is done from these pins.

Detailed Description

Distance Measurement

The Time-Of-Flight sensor works exclusively by using its laser sensor, which in turn emits a photon or can also be deemed as an infrared light signal. This signal isn't visible to the naked human eye, and is only 940 nm in its wavelength. The time taken for this photon to be emitted back at the sensor is then used to calculate the distance travelled in millimeters. It uses the formula from kinematics, which relates the distance travelled being equal to the speed multiplied by the time divided by 2. The speed in this case is the speed of light which is about 3*10^8 m/s. This works collaboratively with the stepper motor, which helps the sensor rotate a full 360 degrees of angle. This allows for the sensor to capture the data for a single vertical geometric plane and gather the distance measurements. The captured data is then sent to the microcontroller, MSP432E401Y, which acquires this data using the I2C form of serial communication. The Microcontroller sends the captured data to the Personal Computer via the UART. The computer receives the data at a baud rate of 115.2 Kbps, 8N1. The data received by the python program is then outputted to an xyz file, which contains all the points in the 3D space. The angle measurement is done using the ratio comparison between the measured angle with respect to 360 degrees and its corresponding number of steps with respect to 512 steps for one full rotation. Refer to figures 6 and 7 for the relevant flowcharts for further details about the process using push buttons and the corresponding outputs.

$$Equivalent\ Steps = \ x = \frac{\text{Desired angle}}{360°} * 512\ steps$$

Solving for the desired angle instead, we get:

$$Desired\ Angle = \ x = 360° \text{ x } \frac{Equivalent\ Steps}{512 \text{ steps}}$$

So therefore, desired angle is equal to 11.25 degrees, number of measurements per full 360 degree rotation is 32 measurements and total 320 for the entire project. The coordinate calculation is done using the basic trigonometry principles of cosine and sine. Each quadrant of the trigonometric plane is different so the negative sign should also be accounted for differently.

The x value would be increments depending on if the full 360-degree rotation is completed. Once a 360-degree rotation is completed and all the y and z samples have been taken for that value of x, we increment x. The y value was calculated using the formula, y = d*cos(theta) and the z value was calculated using the formula z = d*sin(theta)**.** These formulas were used using the built-in math library in the c language, and the respective cos and sin functions were used. But another manipulation in these formulae came from the fact that size the plane is divided into four quadrants, the sin and/or the cosine of the value tends to be positive or negative in some of the quadrants. For example, in the first quadrant, all the trigonometric functions such as sine, cosine, and tan are all positive, whereas in the second quadrant, only sin and csc are positive (since csc is 1/sin) and therefore the sin angle in that quadrant would be positive but the cosine angle is negative. The negative of every angle repeats every 180 degrees, which has allowed me to use 180 in many

portions of the program. The angle starts at 0, goes clockwise so negative angle that is decremented. Since negative angles provide the same result as positive of that angle, as 360-degree turn takes you back to the same angle but in a positive number.

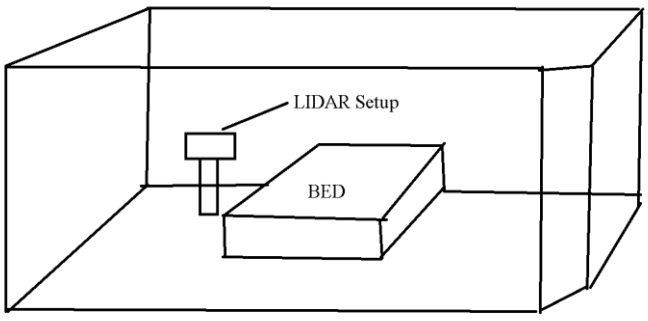
Visualization

To visualize the data into a 3D representation, Open3D was used and it was done using a series of points from the xyz file, the points were outputted to the Open3D console. In other terms, a point cloud was created from the xyz file. In the starting of the python program, pcd is used to term the definition of a point cloud. The outputted data in the corresponding includes on a single line the x value, followed by the y and the z value respectively for that specific point in 3D space. Open3D, numpy libraries were used. Point cloud was printed and also using as.array method in python. Visualization.draw_geometries method allowed for the complete output of the visualization in a new window.

I also managed to visualize the data into a 3D representation using MeshLab. This was done by first creating an xyz file from the python program, and so MeshLab was able to be used to further create a fully meshed .stl file to be outputted by the user. This was done after computing the normal for the point sets and using the Screen Poisson built in option. Since the xyz file contains all the points acquired during the demonstration, and all the specific distances, therefore merely importing the file as a mesh and computing the normal ensures a full 3D figure.

I used the Dell Inspiron 7000 laptop, which was manufactured in 2018, which is a relatively new device and therefore it does contain the latest speeds and the standards set by other laptops available today. The programs were run on the latest version of Windows 10.

Figure 4, Rough isometric sketch of my bedroom, with the LIDAR setup shown below:



Application Example:

1. The device must be carefully place at the end or the edge of any room, having not too bright light, and the relevant connections must be made before the startup of the program.
2. The Personal Computer must be connected to the microcontroller MSP432E401Y using a micro-USB connection on-board.
3. Refer to *figure 5* for the further details about the schematic, for building the system entirely.
4. Open the python file, using any IDE of your personal choice, that can run any python program. Click on the Run button, or any equivalent button the execute the program. Now the serial connection has been opened for communication. Set up the COM3 port, its specifications can be pulled up in the device manager where it shows up as the XDS110 Class Application UART COM3 in the ports section. Make sure it is working. In the CMD program, type in the console, pip install pyserial, this installs pyserial library. By using the python -m serial.tools.list_ports –v command in CMD, confirm that the port specified in the program I have created shows up. If it doesn't, change the line in my program to the name of your personal port number. By running my program, you can confirm all these details listed above.
5. Now, the push button must be pressed to collect the data of the relevant room or hallway that the device is situated in.
6. After the motor finishes its full 360-degree rotation, move the device up in the x direction, or slide the box, whichever way that works. This ensures the data collection in a different geometric plane. The device should be moved by 10 cm approximately.
7. Repeat steps 4 and 5 about 10 times, to ensure that all the data has been collected, and the room's shape has been formed appropriately.
8. Finally, open the next python program and run it the same way as the first one, ensuring the corresponding xyz file has been created in the download folder directory.
9. Run the second python code to view the Open3D output of the 3D Visualization.
10. The second component corresponds to the stl file which is created using the Meshlab program.
11. Open MeshLab, and click import mesh and load the xyz file from its relevant folder.
12. Click on filters, and then click on the "Normals, Curvatures and Orientation bar", and then "compute normal from the point sets". After that, a pop up window would show up and it requires the option "neighbor num" to be set accordingly. This should be set to 120, due to the nature of the data collection in this device, and since there are many neighboring points.
13. Finally, navigate to filters, and then click on "Remeshing, Simplification and Reconstruction". Then click on "Surface Reconstruction: Screened Poisson". Make sure to click the "Apply" button for the settings to be applied and the final product would only be made available once all these options have been checked appropriately.
14. Now, you should be able to see a fully mesh surface, with all the points being connected accordingly. Since the final product is ready, the final can be saved as a .stl file by clicking the "File" menu, and scrolling down to the "Save As" option, and make sure the file type is set to .stl and it shall be named anything of your own choice. This .stl file could now be loaded to the Autodesk Inventor program for further inspection and analysis.

Setup and Implementation

- ✓ It is very important to make sure the wires are properly attached and that they don't get tangled up, although, a counterclockwise ensures that the wires return to their initial state and get untangled even if they did.
- ✓ Make the appropriate connections according to the circuit schematic shown below.
- ✓ Plug the MSP432E401Y prior to the device implementation and running as the computer must recognize it and usually this takes some time, so the computer should be familiar with this new USB device.
- ✓ Keep an eye out for the heating up of the stepper motor, as it does heat up quite a lot at times so this must be one of the precautions.

Coordinate System

- The coordinate system is based on the x, y and z coordinates in 3D space.
- X coordinates are in parallel with the motor's shaft, which is pointing towards the user of the device. Z points straight up, while Y is in line with the body of the motor, and in parallel with the front facing side of the device's box construction.
- Therefore, the device outer box must be kept on a level surface and this would help generate the best outcome.
- The device should also not be held in a human hand and be walked around with, this could also cause disruption to the coordinate and the coordinate system. The data would appear to be inaccurate in this case and may show the actual directions in a different coordinate system that hasn't been defined here.

Limitations

1. It is very well known that the MSP432E401Y microcontroller has a Floating-Point Unit (FPU) that has the ability to support any 32 bit single precision operations as confirmed by the IEEE. It can support addition, subtraction, multiplication, and other square root operations as well. The trigonometric functions are of course used as well, supported by the math.h library which provided useful functions such as cosine, sine, etc. which I found useful for the project. However, 64-bit operations would still not be done in all their entirety and would need further manipulation of being split up, as current capability stands at 32-bit single precision.
2. The maximum quantization error for the ToF sensor for this device is equal to the full scale voltage divided by the 16-bit ADC configuration employed by the ToF sensor itself.
   Calculation: $Max\ Quantization\ Error = \frac{VFS}{2\text{^}16}$, in this case Vfs = 5.5 V, max voltage input to the time-of-flight sensor.
   Therefore, Max Quantization Error = $\frac{5.5\ V}{2^{16}}$ = 8.3923 x 10^{-5} V
3. The maximum standard serial communication rate that can be implemented with the Personal Computer is 128000 bits per second. This was easily verified using my personal computer by searching device manager in the start menu, and then further clicking on Ports (COM & LPT) and then clicking on the standard XDS110 Class Application/User UART(COM3). In the port settings, although this was not the current set up rate, but it is the maximum value that could be achieved from the list of options.
4. I^2C communication was the method by which the ToF sensor and the microcontroller were able to communicate with each other. The clock speed that was used for this serial communication was firmly 100 kbps clock. This was determined from the program itself, where I2C0_MTPR_R was set to 0b00000000000001010000000000111011, which represents the clock of 100 kbps, and a frequency 100 KHz. This was set after the main function and is located at the bottom portion of the program.
5. Reviewing the entire system, the Time-of-flight sensor takes up a handful of time for the setup. There are also some limiting issues with the UART communication platform. In terms of the ToF sensor, the synchronization of the data collection and the speed of the stepper motor might not go hand to hand. The User's PC also have a decent amount of contribution to make to this. If someone's running the program on a 10-year-old computer, then it may not have the specifications according to the standard set by the modern chips and speeds. The Time-of-Flight sensor also is not as old as 10 years so therefore if the user has a newer PC, its serial communication will show a significant improvement. I2C clock speed also is set lower than its usual maximum, which is 400 Kbps. The ToF sensor's incapability to catch up with the speed of the motor could also cause some distortion between the collected points.

Figure 5: Circuit Schematic of the device is shown below:

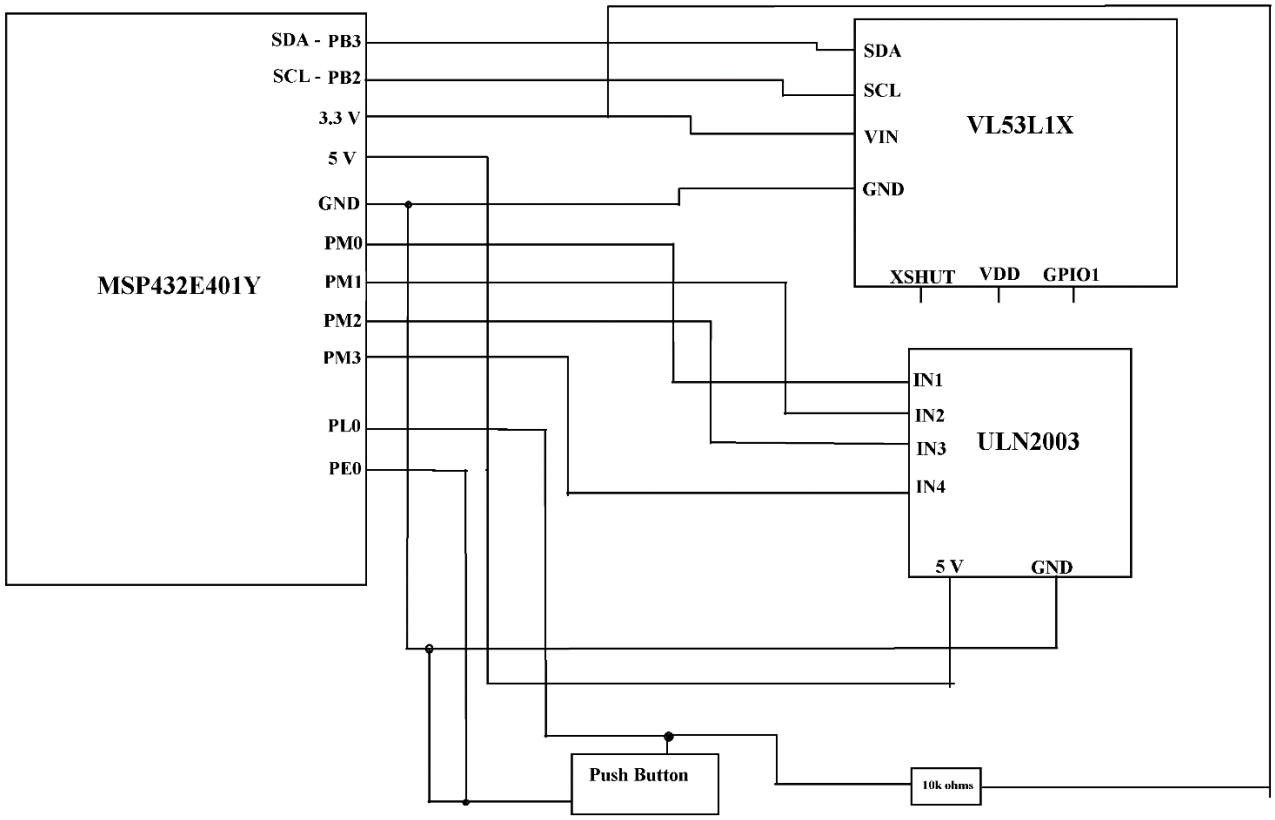


Figure 6, Programmable Logic Flowchart is shown below:

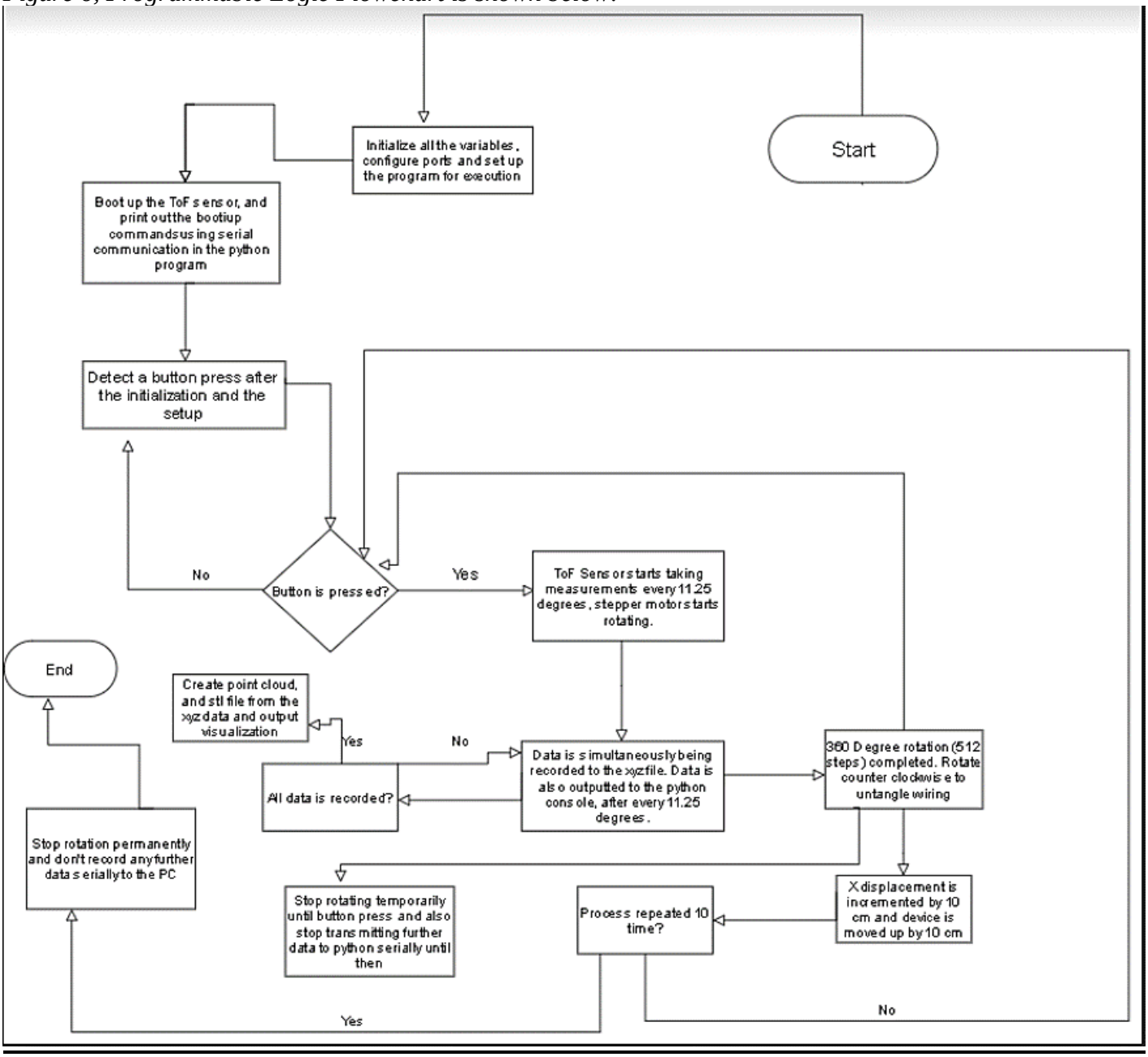


Figure 7, Programming Logic Flowchart for the Python Programs: