2DX4: Microprocessor Systems Project
Final Project

Instructors: Dr. Bruce, Dr. Haddara, Dr. Hranilovic, Dr. Shirani

Hamdan Basharat - L05 - basham1 - 400124515

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 **Hamdan Basharat, basham1, 400124515**

Project Interview Video (Google Drive Link):

https://drive.google.com/file/d/1nP5nKpotvW-f4CcNoyhC3kT7fIjlMwwR/view?usp=sharing

2DX4 - Time-Of-Flight Environment Mapper

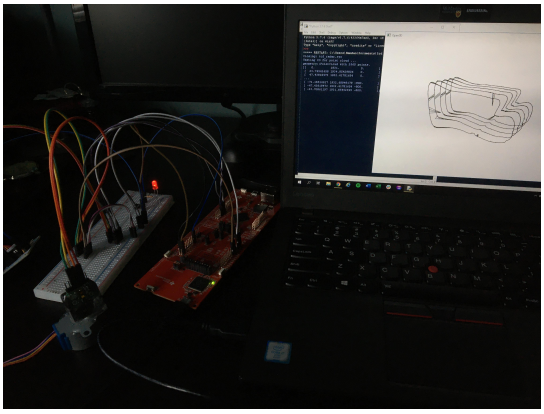
1 Device Overview

1.1 Features

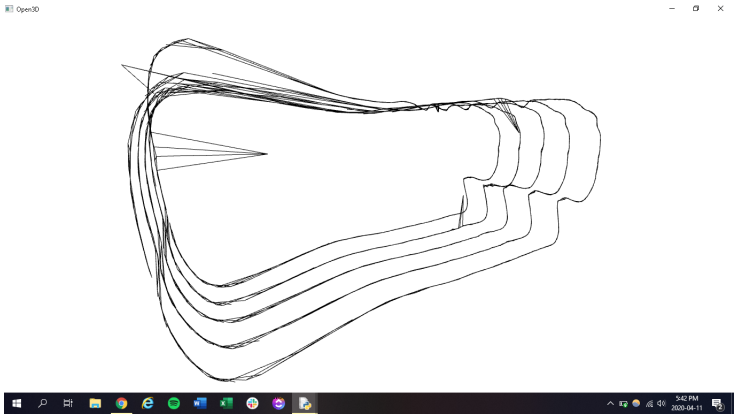
- Microcontroller
  - Texas Instruments MSP432E401Y microcontroller with 120-MHz Arm Cortex-M4F Processor Core With Floating-Point Unit (FPU)
- Memories
  - 1024KB of Flash Memory
  - 256KB of SRAM
  - 6KB of EEPROM
- Bus Speed
  - 24 Mhz
- Operating Voltage
  - ToF sensor operates from 2.6 to 5.5V
  - Stepper motor and other peripherals supplied 5V
- ADC
  - Channels: 16-bit distance reading output
  - Mode Sampling Rate: Short is 50 Hz max, Medium & Long is 30 Hz max
- Cost
  - $145.00 CAD
- Language
  - Kiel Microcontroller Code in Assembly
  - Data Analysing and Visualization Code in Python
- Serial Communication
  - Baud Rate: 115200 bps
  - Terminator: '\n'

1.2 General Description

The Time-Of-Flight Environment Mapper made for the 2DX4 final project is an embedded spatial measurement system that uses time-of-flight to acquire information about the area around it, and visualizes this data through an associated 3D rendering software.
On the market, Light Detection and Ranging (LIDAR) systems are available, however they are very costly and take up a lot of space. This project is a smaller and less expensive alternative that will allow for exploration and navigation for use indoors. Not only are commercial systems more expensive, but they are much more complex than student engineers require. With this alternative system, students can measure physical phenomena accurately at a low cost while gaining experience on how industry data acquisition works.
The system is composed of two main parts: the physical hardware set-up used for data collection, and the computer software that captures this data and visualises it. For the hardware, an MSP432E401Y micro-controller controls a stepper motor with a time-of-flight sensor mounted on it to achieve a 360 degree measurement of distance within the y-z geometric plane. The time-of-flight (ToF) sensor measures distance by emitting a laser, which leaves the sensor and reflects back to the SPAD (single photon avalanche diode) receiving array. By measuring the round trip time of the emitted pulse, the distance from the sensor to



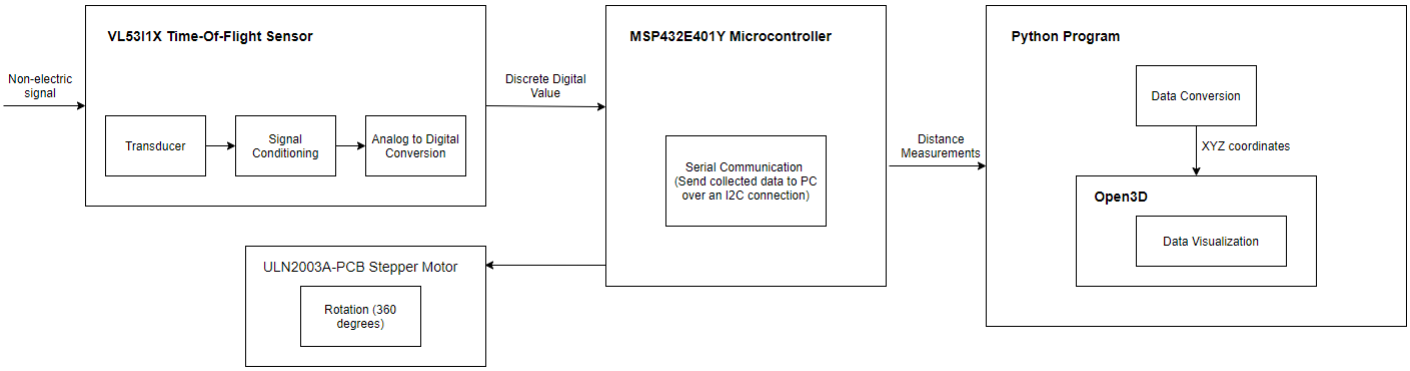
(a) Entire LIDAR device



(b) Open3D Visualization

the reflected surface is found. The distance measurements are sent to the computer over serial communication by establishing an I2C connection, and are accessed through one of the computer's COM ports. Then on the software end of the system, the data is collected from the COM port with a Python program and analysed to produce point coordinates representative of the distances measured while rotating. Using the visualization software Open3D within Python, the points are then used to depict the environment that was mapped using 3D point clouds.

1.3 Functional Block Diagram



2 Device Characteristics Table

Pins		
_____	Stepper Motor	PM0-3
_____	Button	PF2
_____	External LED	PL2
_____	Onboard LED	PN1
_____	SCL	PB2

_____	SDA	PB3
Bus Speed		24 MHz
Serial Port		COM7
Communication Speed		115200 bps

3 Detailed Description

3.1 Distance Measurement

In order to produce a distance measurement from the time-of-flight sensor and put it in a format that can be used by a 3D rendering software there are several steps involved. The process involves the acquisition, transmission, and conversion of data into the xyz plane.
First off, the microcontroller has to determine when exactly to take a measurement. Code from Kiel that is written in Assembly and C is loaded onto the microcontroller and stored. The main function from the code tells the microcontroller what sequence of actions to take when it is reset. From the circuit schematic in section 6, it can seen that the microcontroller is connected to both the time-of-flight sensor the control unit for the stepper motor. The time-of-flight sensor is a small unit that can emit a 940nm class 1 laser. When the laser leaves the sensor and hits a surface, it reflects back to the SPAD (single photon avalanche diode) receiving array. By measuring the round trip time of the emitted pulse, the distance from the sensor to the reflected surface is found. The measurements that are then received are in millimetres. There are three distance modes for measurement, short, medium, and long with which the sensor can measure up to 360cm away. The sensor is mounted on top of the stepper motor which is controlled by the motor driver PCB. The driver takes a four bit command from the microcontroller and with it applies the necessary power pulse to step the motor. By running the correct sequence of position commands in the Kiel code and looping them for a given number of steps, the motor can be turned to a very precise position.
For this project, once main is run the controller waits to see if the button is pressed. Once it is, the sampling sequence is initiated to collect eight distance measurements around the sensor. In a loop, the motor is told to take 64 steps in the clockwise direction, which turns it 45 degrees. At this point the sensor takes a distance measurement and communicates it to the computer. This loop iterates eight times to collect all eight measurements then the motor takes 512 steps in the counter-clockwise direction to reset its position. To communicate the raw sensor data from the microcontroller to my computer, the Kiel code on the board allows for serial communication of data. The schematic shows the connection of the SCL (clock signal pin) and SDA (digital signal pin) on the sensor to Port B to allow for data to be sent over I2C communication. In the code, I2C is initialized at the start of main, which enables these Port B pins. Then in the continuous while true loop inside main, the time of flight sensor is booted up, initialized, and told to start ranging. After the motor rotates to it's given position, the sensor can then checks if data is ready and a distance measurement is taken. From here, the measurement is printed over I2C to the computers COM port. Using

device manager and seeing which port was added when I connected the board, I found out this port is COM7 on my computer. The information is packaged in 8-bit packets, which is always followed by an acknowledge bit. Therefore, eight bits of information are transmitted at once. However, if we include the acknowledge bit, this technically becomes nine. If data is larger than what is allowed for a single serial packet, the data will be divided into components of an allowed size. These packets coupled with an acknowledge bit will be detected, and grouped together accordingly at the receiving end. The motor continues to rotates 45 degrees before another measurement is taken until a full rotation is made and all the data is sent.
Now that the raw sensor data from the microcontroller is coming to the computer, it must be processed to be used. I used Python to capture the data over I2C like before in order to synchronize the program with the data readings. To do so I imported the pySerial library and opened a serial connection to COM7 at a baud rate of 115200. In Kiel, it was specified that only the distance measurement is sent over serial as opposed to range status and other information. In a loop that iterates eight times since there are eight distance measurements being sent over I2C, the software read the line of serial data and stored it to a variable. It then type set it to a string variable and used the 'rstrip' command to remove any extra space or new lines. Since I used Open3D as my 3D rendering software, I had to convert and store the data into an XYZ file. At the start of the code a file was opened under the name 'tof_radar' as an XYZ file so that it could be written into. For each distance measurement, y was made equal to the distance times sin 45 degrees and z equal to distance times cos 45 degrees since eight measurements were taken in a 360 degree rotation. The 45 degrees is multiplied by 'i' from the loop iteration so that it indicated the correct angle as the motor moves. Finally, a string was created combining the x, y, and z values, where x is the displacement of the motor, and the point was written to the file. After the loop finishes both the serial port and the xyz file are closed.

3.2 Displacement

To measure the displacement of the device in x-plane between the rotations, an available option was to use an IMU sensor. To implement this into the project, the IMU would also have to be connected to the I2C channel and it would send positional data of the device along with the time-of-flight measurements. The IMU sensor would be attached to the side of the motor on a part that does not spin, and would be programmed to measure the position in the x-axis after every rotation. Then in the Python program, after the data is received over serial the previous x-position would be subtracted from the current x-position to determine the displacement.
Instead of this, I chose to manually indicate the displacement of the device in between rotations. In the Python code, after each each full rotation of the device the x-value for all subsequent points being placed in the xyz file is set to be 200mm lower than the current x-value. This indicates a 0.2m displacement in the negative x-direction. The displacements

can be seen in figure 1(b) above.

3.3 Visualization

During this point the process, the distance measurements have been received by my computer and the Python program takes them from the serial COM port and converts them to xyz coordinates. To start, my laptop is running Microsoft Windows 10 Pro on a build housing an Intel i7-6600U CPU clocked at 2.6 GHz and 8 GB of RAM memory. The Python program that I am using is of version 3.7.6 which supports the extra libraries used in the project. Now, in order to visualize the data that was received and stored as coordinates my program uses the 3D rendering software Open3D version 0.7.0. This software was downloaded and installed with a wheel from the Python library website. The idea is that Open3D will take the xyz coordinates that were generated and produce a interactive graphic that will represent a map of the room that the sensor ranged.
At the start of the Python script, along with the previous used libraries such as serial, the Open3D library is imported. This gives the program access to functions available in that library. The code initially opens the serial port to COM7 in order the periodically receive distance measurements from the microcontroller and convert them to xyz coordinates like it was mentioned above. Now that an xyz file is available to the program, the function 'o3d.io.read_point_cloud() is called using the file as parameter to generate a point cloud of the data. A point cloud is a set of data points in space, and in our case represent the three-dimensional coordinates taken from the room that the device is placed in. From here, the program places this point cloud as an array in the function 'o3d.visualization.draw_geometries()', which automatically plots the points in a 3D coordinate system. In the visual, each rotation can be seen mapping the perimeter of the room in the y-z plane. Offsets from the y-z plane in the x-axis then represent when the device was displaced in between rotations.
While this achieves the purpose to visualize the mapped measurements, it is hard to distinguish rotations unless there are a very large number of points. To overcome this, the next step in the program is to generate a line set that connects the points together to produce a more appealing visual. To start, a few variables are initialized as 'pt1-4' for the vertices of a square to be placed around each coordinate, and a variable for the offset within the plane. As well an empty array is initialized to hold the lines. Next, a loop that iterates for the number of points in point cloud is run to connect all the points within the plane. During each iteration of the loop the function 'loop.append' is used with the combinations of the vertices variables to connect each one together with a line. At the end of each iteration the offset variable is incremented by the number of displacements used in the trial. The next step is to connect the vertices between each of the planes. The variables initialized before are reset and an additional variable is added for the displacement offset. Like before, a loop now iterating for one less than the number of points is run where 'lines.append' is used to the connect the points together. From here, a constructor call is made to create the line set us-

ing point cloud and lines array. Finally, the function 'o3d.visualization.draw_geometries()' creates the graphic like before but with the lines connected (see figure 1(b) above).

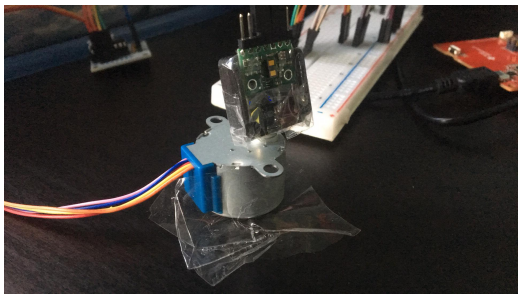
4 Application Example with Expected Output

The Time-Of-Flight Environment Mapper can be used to map a room using distance measurements with the data then being displayed in a 3D rendered visualization. The following steps will walk through setting up the device and an application of the results.

1. Choose a room that you would like to map using the Time-Of-Flight Environment Mapper. The time-of-flight sensor works better in darker rooms since other forms of light will not disrupt the measurements, however it can still be used in ambient light to a reasonable effect.

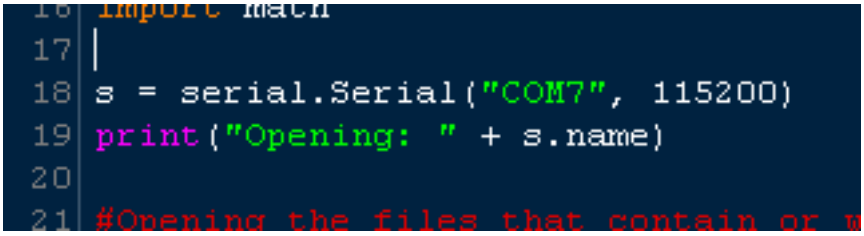
2. Once you have chosen a room, bring the device components and a laptop to be set up.

   - Referring to the circuit schematic from section 6 of this document, ensure that all the device hardware components are wired together correctly (microcontroller, ToF sensor, Stepper motor and driver). Choose a flat surface in the center of the room and secure the stepper motor's base to it. This can be easily done using two short strips of tape connecting the surfaces. Next, take the wired ToF sensor and secure it to the shaft of the motor with the sensor facing parallel to the surface the motor is resting on. This ensures that the plane being measured is the y-z plane. Again this can be done using tape but it helps to include a spacer between the shaft and sensor such as the 3D printed PLA one seen below.



   - The next step is to set up the laptop to receive and visualize the data. The Kiel code should already be loaded onto the microcontroller, however if not, the Kiel environment and project code must be downloaded. Ensure that the build target for the project is the microcontroller, and after connecting the USB to Type-C cable between the laptop and device, load the program. Next, install Python 3.7.6 onto the laptop while ensuring that path access is enabled. The libraries for pySerial and Open3D 0.7.0 can be found online and installed using Python commands in the PC command prompt. Download and open the Python script for this project. Using the PC Device Manager, you can see which COM port was added when the microcontroller was plugged into the laptop. In the Python program, edit the

number for serial port opened at a baud rate of 115200 to match this COM port and save the file.



3. After pressing the reset button on the microcontroller, the device is ready to start mapping the room. Run the Python program on the laptop when you have ensured that you are ready to begin. A new Python window will open indicating that the program has started and that the serial communication has opened to the specified COM port.

4. Press the button on the device to start rotation and taking measurements. The device will automatically rotate the motor and take measurements with the sensor. The external LED will flash and indicate when a distance measurement is taken.

5. After a full rotation, unfasten the device and using a stack of books to displace it 0.2m vertically from the flat surface. After securing it down again, press the button so that the device can take another set of measurements. Repeat this process until five sets of measurements have been taken.

6. At this point, the Python program will detect that it has received all the data and will convert the measurements into xyz coordinates. It will then produce the 3D visualizations of the point coordinates. A new window will open up showing the point cloud representation of the mapping. The left mouse button can be used to rotate the viewing angle, and the scroll wheel to control the magnification. As well, holding the control key along with the left mouse button can be used to reposition the visual. Closing the window opens the next visual which is the preferred point cloud with lines connecting the points.

7. Close any remaining windows and restart the Python program if you wish to repeat the process.

5 Limitations

1. There are limitations to the microcontroller's floating point capability and use of trigonometric functions that result in an error in the reported distance values and coordinates. When receiving a distance measurement from the ToF the microcontroller's ADC has to convert the value to a usable digital one. The larger the ADC's resolution is, the greater the quantization that develops. As well, since trigonometric functions were used and not all angles produce exact values, an error is also produced here.

2. The maximum quantization error is equal to the ADC resolution. The equation for ADC resolution is,

$$resolution = V_{FS}/2^m$$

Therefore, the maximum quantization error for the IMU is $5.49e^{-5}$, and for the ToF is $8.39e^{-5}$.

3. The maximum standard serial communication rate that can be implemented with my PC is a baud rate of 115200 bps. This was verified because RealTerm does not allow for baud rate's higher than this to be received by the computer.

4. The communication method used between the microcontroller and the ToF sensor was I2C serial communication. The speed of this communication method was 115200 bits per second.

5. After reviewing the entire system, the primary limitation on speed comes from reading the xyz file to create the visualization with Open3D. This becomes apparent with large datasets and sample sizes because the point cloud and 3D rendering take longer to make. To test this, I experimented with less rotations and less distance measurements taken, and the speed noticeably increased.

6. Based on the Nyquist Rate, the necessary sampling rate required for the displacement module is twice the frequency of signal frequency from the module. For example, if the displacement module sent an input signal at 1kHz, that signal must be sampled at 2kHz. If the input signal exceeds this frequency, the data will be under sampled and may be missing information.

6 Circuit Schematic

See page 10.

7 Programming Logic Flowchart

See page 11 and 12.

References

[1] I2C. [Online]. Available: https://learn.sparkfun.com/tutorials/i2c/all. [Accessed: 13-Apr-2020].

[2] "Visualization," Visualization - Open3D 0.9.0 documentation. [Online]. Available: http://www.open3d.org/docs/release/tutorial/Basic/visualization.html. [Accessed: 13-Apr-2020].

Figure 2: Environment Mapper Circuit Schematic

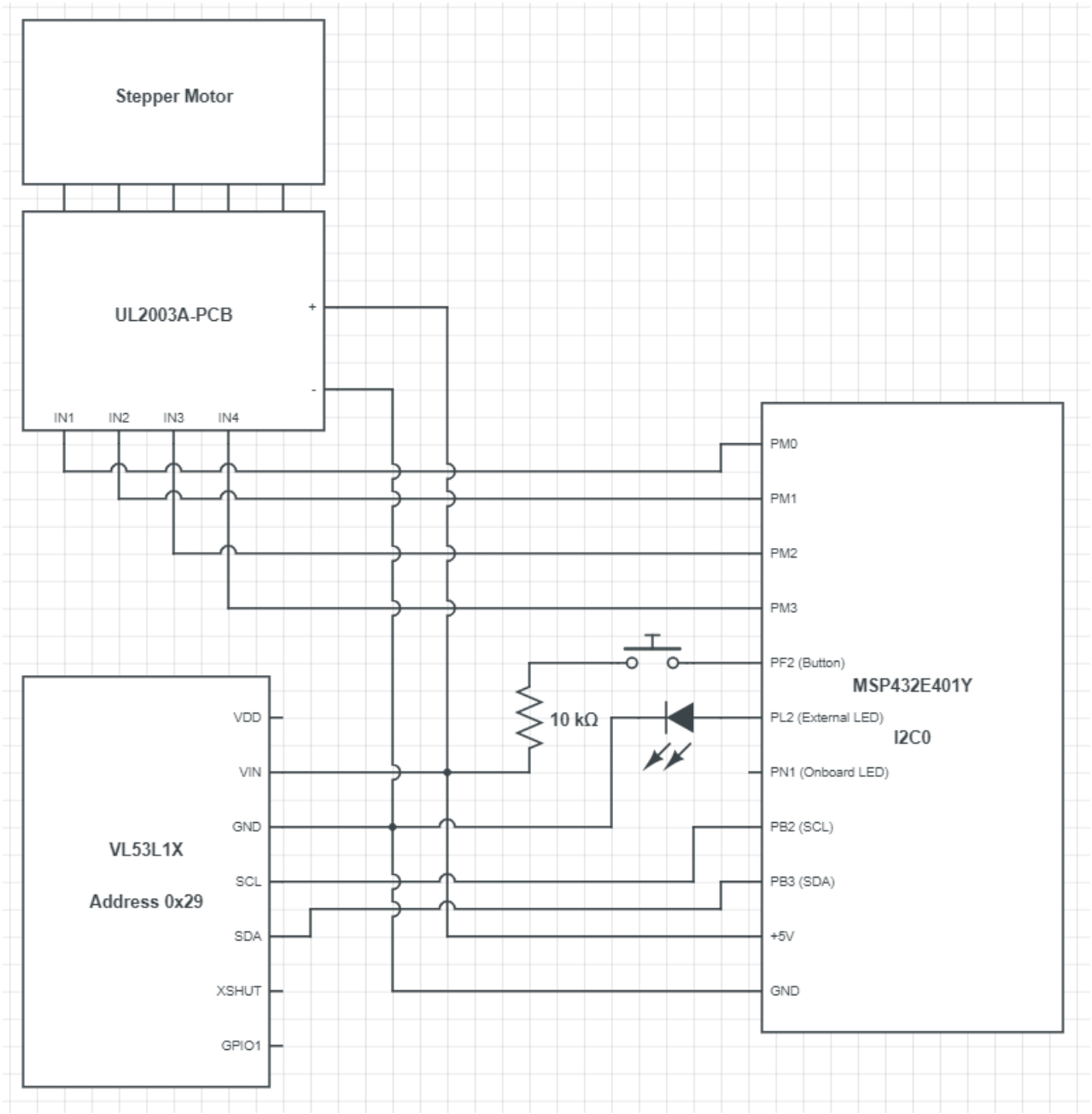


Figure 3: Microcontroller Kiel Code Flowchart

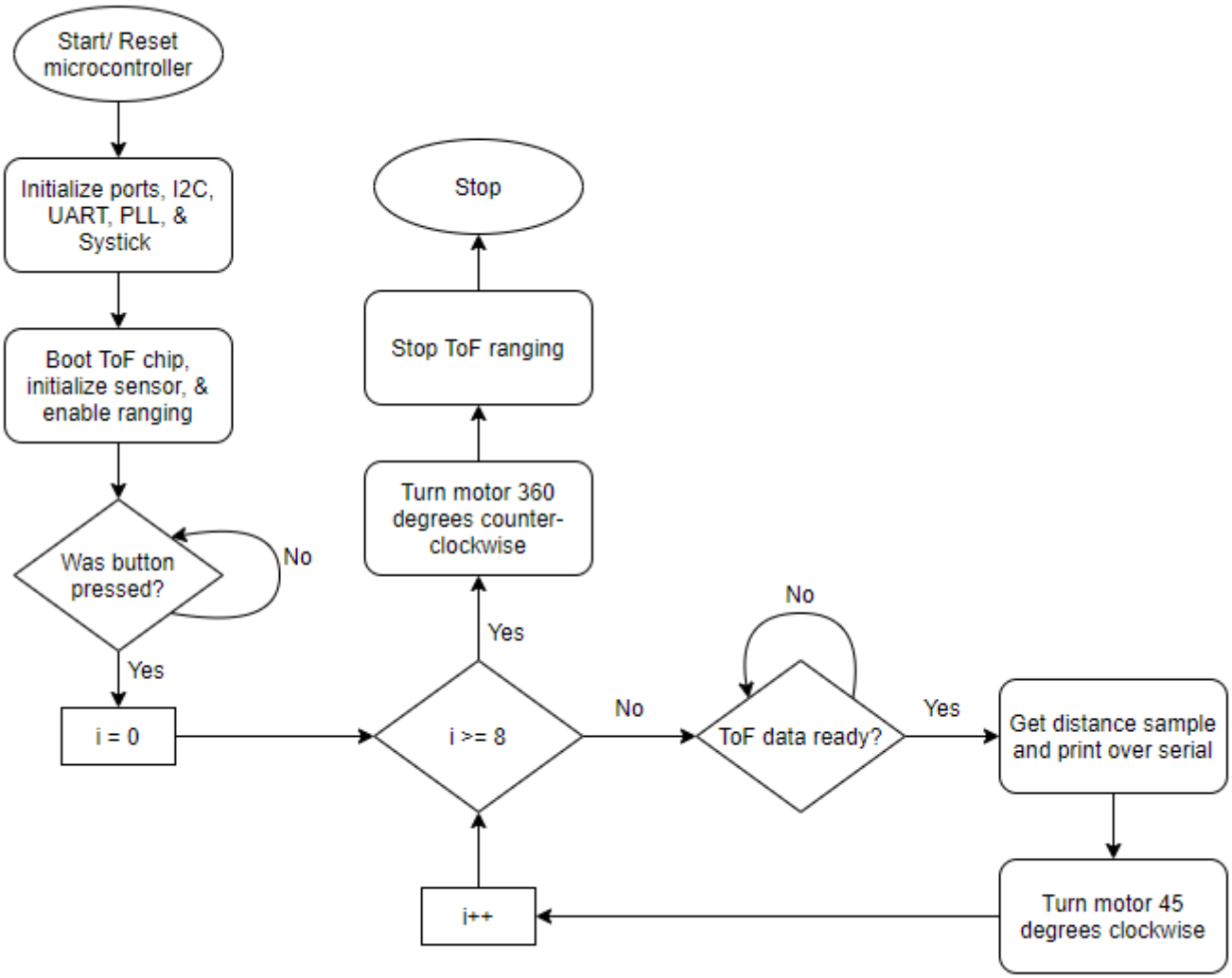


Figure 4: Python Visualization Code Flowchart