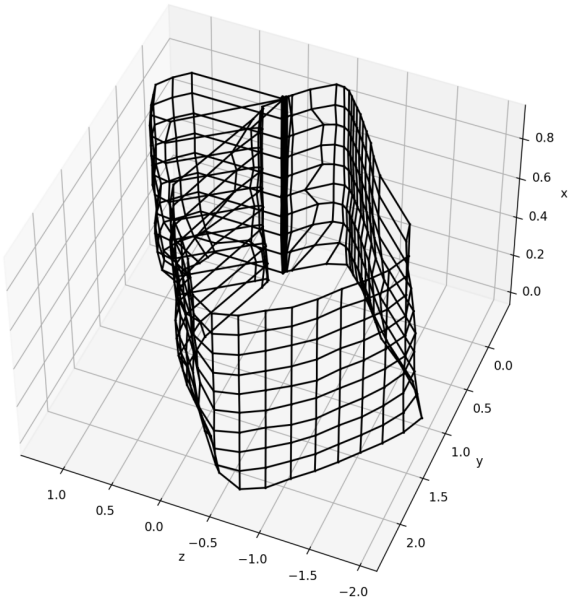
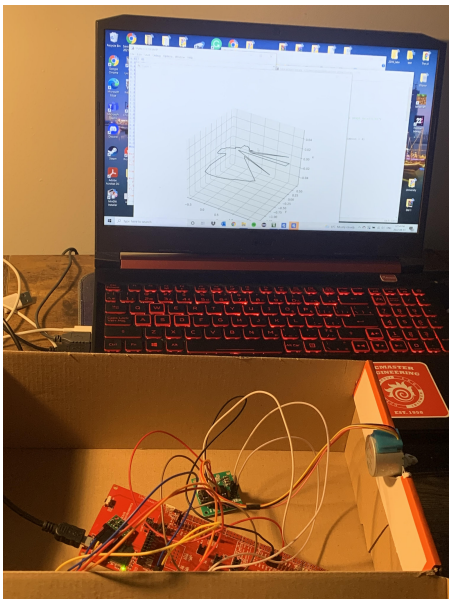
2DX3: Microprocessor Systems
Final Project

Instructors: Drs. Boursalie, Doyle, and Haddara

Sava Jankovic-jankovs-400292525-2DX3-Monday Afternoon-L06

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 [Sava Jankovic,jankovs,400292525]





Device Overview

This section outlines the features of the project, the general description of the project, and the data flow graph that resulted from it.

Features:

- Texas Instruments (TI) MSP432E401Y microcontroller device
  - Processor: Cortex M4
  - Bus speed at 48MHz
- TOF sensor
  - Model VL53L1X
  - Range of up to four meters distance from the center point
  - Voltage operating range: 2.6-3.5 V
- Data communication types
  - I2C is used between the microcontroller and the TOF sensor
  - UART is used between the microcontroller and the PC (the code that is processed in the PC)
- Stepper Motor
  - UNL2003
  - Since it's a half stepper, it would require 512 steps to do a full 360 rotation.
  - Attached to LED indicators, that indicates when the motor is moving.
  - Operates within 5 to 12 volts.

General Description:

For the 2DX3 final project, we were asked to create a Lidar system that would be capable of scanning and recording the surrounding area using the VL53L1X ToF sensor. This sensor would ultimately be used to gather information about the area around it and process it through the microcontroller. To get a 360-degree view of our project (in our case, a hallway) we would use a UNL2003 stepper motor. This stepper motor would also enable us to get the measurements within a single vertical geo plane. To make the project easier to conduct, I placed the motor and the sensor on top of a shoebox, which wouldn't require me to manually hold the motor and the ToF sensor as it is rotating.

For the project, the ToF sensor that I used (VL53L1X) would consistently emit IR pulses to its surroundings, which would in turn measure the distance of the space around it and feedback to the system. This analog data would then be converted to digital data and be fed back to the microcontroller via an I2C communication protocol. In turn, the microcontroller is connected to the PC using a USB port, and the data from the sensors such as the angle, distance, and displacement is transferred to the computer through the UART communication protocol. Using the python code, the computer would receive this data, and visualize it on an XYZ plane, showing the graphic of the room through the sensor.

Block Diagram:

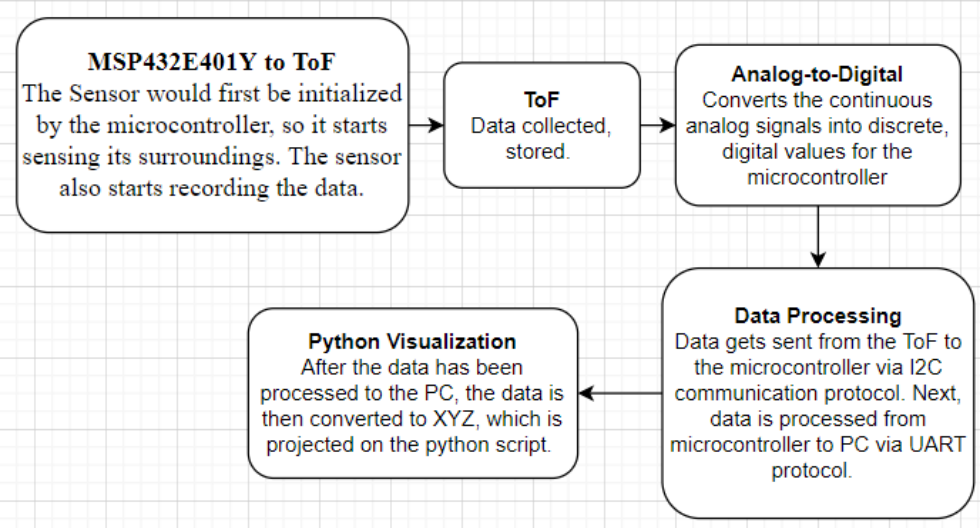


Figure 1: Block Diagram

Device Characteristics:

Device	Feature	Detail
MSP432E401Y and PC	Bus Speed	48MHz
	Serial Port	COM5
	Baud Rate	115200
	Python Version	3.9.12

Device	Pin on sensor	Pin on Microcontroller
VL53L1X	VDD	/
	VIN	3V3
	GND	GND
	SDA	PB3
	SCL	PB2
	XSHUT	/
	GPIO1	/

Device	Pin	Microcontroller pin
ULN2003	IN1	PM0
	IN2	PM1
	IN3	PM2
	IN4	PM3
	+	5V
	-	GND

Detailed Description

Distance Measurement:

The distance measurement for my final project was recorded using the VL53L1X device (the ToF), that is attached to my microcontroller through ports PB2 and PB3. The ToF was attached to the stepper motor by glue tac and rotated along with the motor throughout the demo. In turn, since the ToF was rotated 360 degrees, it made a measurement for every step of the way, in its 360-degree plane.

The VL53L1X sensor uses infrared technology to calculate the distance surrounding the sensor and the overall setup. To calculate the total distance, the sensor would emit the waves to nearby objects and would measure the time that it would take for the IR signals to bounce off the surroundings. Based on the configurations of the ToF, the distance calculated is based on the time it takes for the signals to bounce back. Finally, once the data is gathered, it is communicated to the MSP432E401Y by the I2C communication protocol.

The settings of the ToF that were used for the project were all default settings that were provided by the ToF API, which could be found in the user manual provided to us.

As the ToF was activated and started taking in information about its surroundings, the process of visualization is clearly outlined in the flowchart in Figure 2. As seen on the flowchart, throughout the microcontroller initialization, the I2C capabilities are activated on the microcontroller, which would be used to communicate to the ToF sensor. Additionally, the other variables that are required for the sensor are also initialized before use and help the sensor activate and collect data. Once booted, the sensor is on and ready to capture real-life data, using its IR waves, but it is waiting on the interrupt button to be pressed by the user.

The interrupt button on the microcontroller is used to initialize and halt the data transmission by the ToF sensor and is pressed at the very beginning of the project, which would also initialize the motor movement as well. The state of the system is stored on my Keil code, and the variable responsible for it is sensorState, which is a boolean that signals whether the button is pressed or not. When the variable is set as 0, the button hasn't been pressed, and the sensor is in standby mode. However, when the interrupt-based button is pressed and the program is initialized, an interrupt is triggered, which changes the value of the boolean to 1. Since I have two loops in my code, each one will be executed based on the value of the sensorState. For example, when sensorState is 0, there will be one while loop used for the microcontroller. However, when the sensorState changes to 1 (this is when the button is pressed), the code will ignore the first loop and enter the second loop near the end of the Keil code.

When the system is inclined to capture real-life data, the ToF sensor would be initiated, collecting data for the entirety of the step motor rotation, 512 steps to be exact. When the data is gathered from the sensor after the button is pressed, it then gets transmitted from the ToF to the microcontroller using the I2C communication protocol. Once the data is received at the microcontroller, it ten gets forwarded to the PC through the USB port, using the UART communication protocol. It would then be processed through my python code, and the data would be graphed on the .xyz file. Once the motor moves all 512 steps, the sensor would halt its data extraction.

Once my python script receives the data from the microcontroller through the UART communication protocol, these data measurements would be written to a graphical xyz file, which shows the graphic of the sensor input data. The y and the z components of the graph are calculated using distance measurements, which could be found through trigonometry. With the assumption that initially the ToF is facing upward, the y and the z components can be easily calculated using two different equations based on the displacement. These are: $z = dsin(\theta)\ and\ y = dcos(\theta)$. In my python code, these equations were entered as **zpoint[o*scans_spin+i] = data[i]*math.cos(math.radians((360/scans_spin)*(i)))** and **ypoint[o*scans_spin+i] = data[i]*math.sin(math.radians((360/scans_spin)*(i)))** respectively**.** The displacement values will be discussed in the displacement measurement section. As for the angle of the motor turning, this can be calculated by steps performed/rotation steps * 2π.

Displacement Measurement:

Throughout the project, we understood that the sensor would only be able to calculate the values of the y and the z displacement at first. This meant that we would manually have to create an x displacement. This was done by manually moving the sensor for every rotation, along the x axis. Since the x displacement is manually created and measured, there was no need to modify any x component within the code before it was ran and sketched on the xyz file.

When the new x displacement data is gathered by the sensor as the box is moved, it would get transmitted to the microcontrolelr after the second trial onwards. This is why the first trial is presented as a 2D sketch, because the x displacement does not exist yet.

Visualization:

The 3D visualization of the data was performed through the .xyz file associated with my python program. The software that was used to sketch this 3D data was the Python library, which then used the libraries functions to visualize the data that was taken in from the sensor.

The 3D visualization was tested on my Acer Nitro 5 Windows 10 laptop that uses an OS build of 19044.1586. The RAM of my laptop is 8 GB, although 7.84 is usable. Additionally, the laptop uses an Intel(R) Core(™) i5-9300H CPU processor, at 2.40GHz. The data collection for this project was run on Python 3.7.12 version.

The visualization was done through one of the python libraries, that graphically produces plots as a result of the input from the ToF. It presents this data through a xyz format and is called mplot3d, which was imported as Axes3D onto the code.

At first, the visualization of the hallway was projected in a 2D sketch, because of errors associated with numpy and the casting of complex values to real ones. However, the visualization was further improved as I moved the sensor forward, creating a larger x displacement. In my code, there was an option to further pile multiple 2D sketches on top of each other every time that the sensor did a 360 rotation with the stepper motor. As a result, a larger 3D graphic was created. Furthermore, I did this 9 more times, with a displacement of 0.1 each time. Henceforth, the final graphic showed an x displacement of 1, and looked like a 3D visualization of the hallway that was needed.

Application Example and User Guide:

To use the lidar system, the user would have to go through a thorough process of setting up the computer programs and getting the system ready. This process is necessary if the user wants to use their PC to graph the ToF data and extract it from the microcontroller. For Windows 10 machines just like mine, this is the process:

1. Download and install Python 3.7 from the Microsoft store. Although you can install from google as well, it's preferred that you install from the Microsoft store as it makes the command prompt use much simpler. Also, you should only download versions 3.6 to 3.9, as the graphics part of this project isn't supported by 3.10.

2. As soon as python is installed, open the command prompt by searching up cmd in your search bad. When the black screen appears, type in "python" to run the program through the command prompt. Once you do this, make sure to install all the desired packages such as serial, NumPy and matplotlib. To do this, you would need to enter "pip install [package]" for all three of the packages.

3. Once the packages have been successfully installed, you will need to use the python code for the first time. Since you can't just utilize and save the code through the python app itself, you will need to use the python IDE for opening the code up. For Python 3.7, you will use the Python 3.7 IDLE app, which you can simply search up in the search bar.

4. The IDLE app is very important in executing your python code. To run any .py code that you have, simply go to file->open and choose the python file that you wish to write on/ edit.

Once you have set up your packages and have python working, you're ready to collect and graph your data from the ToF sensor. To do so, you will need to follow the following steps:

1. Open the specific python file that you will use for the project in the Python IDLE 3.7.

2. Once you have inputted your created python code, you should connect the code to its rightful port. The port that should be connected to your python code is one through UART, as it's the protocol used for communication between the microcontroller and the PC. To figure out the port for your specific computer:
   - Open up cmd (Command Prompt)
   - Type in python -m serial.tools.list_ports -v
   - This will tell you which port on your computer is receiving UART communication protocol.

   Once you know your port, make sure to add it to your code, along with a baud rate of 115200 and a timeout of your choice. The code for this process should look like this:
   s = serial.Serial('COM5', baudrate = 115200, timeout = 4)

3. When you're sure that the information for the ports is correct, run your code through the IDLE.

4. Next, connect your microcontroller to your PC, using the USB port that's right beside the reset button on the MC. ***Make sure that your Keil code is already loaded to the microcontroller, otherwise the python code will not work!***

5. Make sure that your setup is correctly adjusted. For our project, it is recommended that you place the stepper motor into a hole (preferably on a shoebox!), and attach the sensor to the tip of the motor, so they rotate in phase. Make sure that the sensor is facing upwards, which would make sure that the wires are not entangled.

6. Once the device is connected, make sure that you press the button right beside the USB input, as it would reset the microcontroller and get it ready for the process.

7. As soon as everything is ready, run the code. The first question asks you if you'd like to delete previous graph data, answer yes. Then, press the GPIO J1 button, which would begin the rotation of the motor and would activate the ToF recording.

8. Wait for the whole rotation to finish, once it is finished and all the data is presented on the python script, the code will ask you if you'd like to graph the data out. Say yes, and you should get a first glance at the 2D presentation of the room you are in (if the sensor was detecting its surroundings correctly).

9. Once you're done with that step, run the module again, this time make sure to say N to deletion of previous graphing data. Now, move forward as much as you'd like, to acquire a third (x) displacement. Once again, press the GPIO J1 button, and wait for another rotation to finish. Once finished, graph the second plot, and you will see that there is a third dimension this time, the x displacement.

10. Repeat these steps as many times as you'd like. For my code, I conducted these steps a total of ten times, to make sure that the hallway is large enough for presentation.

Limitations:

1. The MSP for our project has a floating-point unit (FPU) that can support 32-bit addition, subtraction, and other operations that are inputted by the user. Additionally, this built-in characteristic can also handle conversions between different floating-point data formats, which makes it useful for operations that we will use for our ToF sensor. Therefore, the trigonometric functions are imported from the math.h library can be performed on the float variables within our microcontroller. This is no issue for the project.

2. Maximum quantization error for the ToF sensor would be the distance that it can sense the surroundings up to, divided by 2 to the power of the bits represented. Henceforth, 4m=4000mm, meaning that 4000mm/2^16 = 6.10*10^-2 mm.

3. For my PC, the standard communication rate is about 128000 bits per second. However, for our code, we use 115200, which means that our project would still be possible to finish at this baud rate limitation. This is verified by checking the computer port settings for the UART Port (in my case, COM5).

4. The integrated circuit I2C communication protocol is implemented between the ToF sensor and the Microcontroller. The PB2 is connected to SCL, whereas the PB3 pin is connected to the SDA pin on the sensor. The data between the ToF would go through the microcontroller, and then make its way to the PC through the USB port COM5 (which varies by the user's PC).

5. The primary limitation of the speed may be the ToF sensor, which would take more time in working compared to the stepper motor. The ToF sensor can cause more delays in the program, whereas the stepper motor would not cause as many, because it takes a small portion of the whole loop, whereas the ToF sensor takes up more space in the code. The way that this can be tested is by adjusting the time delays in the code, to see how the program would respond to the absence of delays for the motor compared to the sensor. Through tests, I concluded that when the delays within the program were removed, the sensor did not show any data. The program had to pause, which created bigger time delays in the process. However, when I did this with the motor, the motor did not rotate, but the program went on without any further time delays.

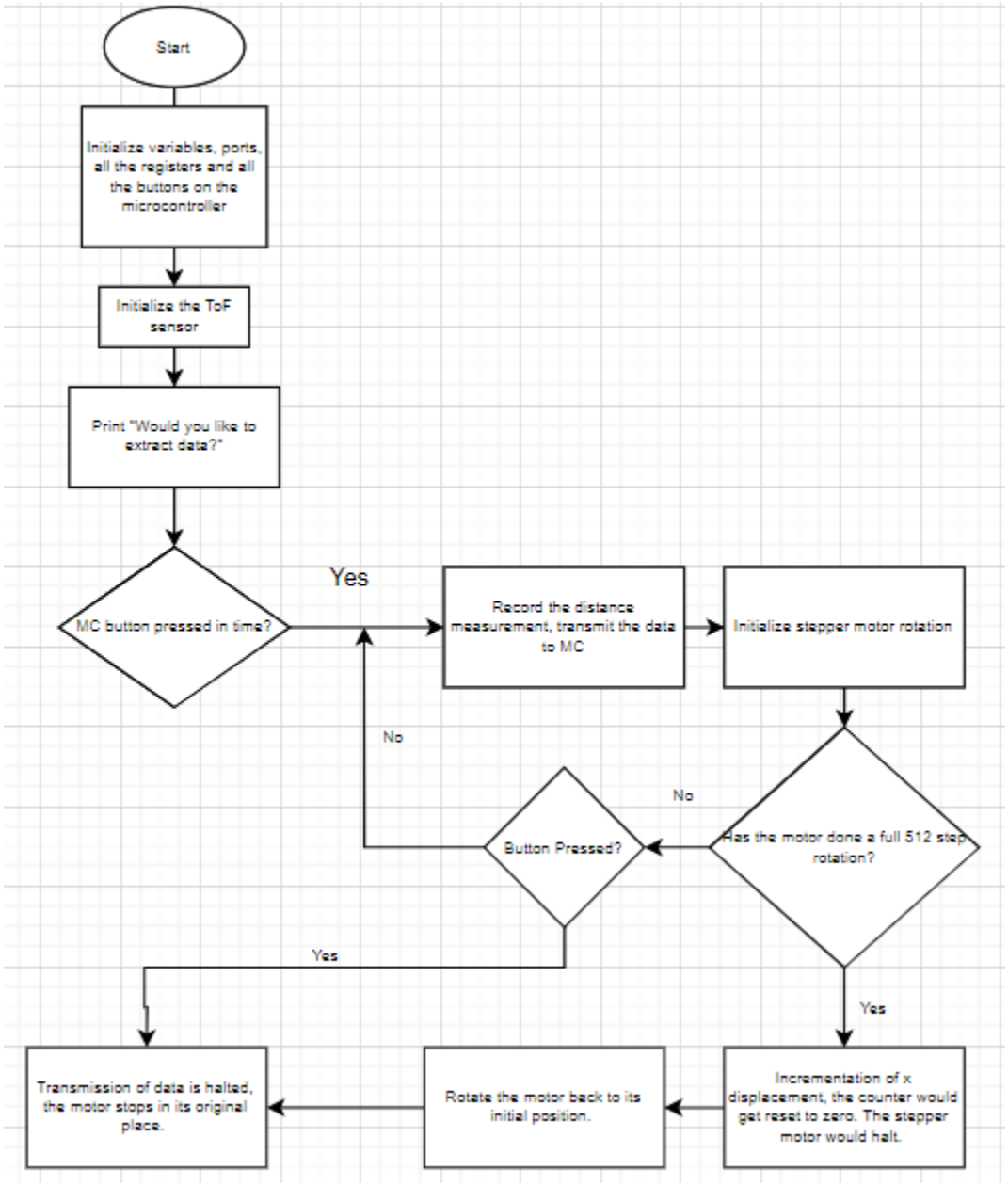


Figure 2: Microcontroller Keil Flow Chart

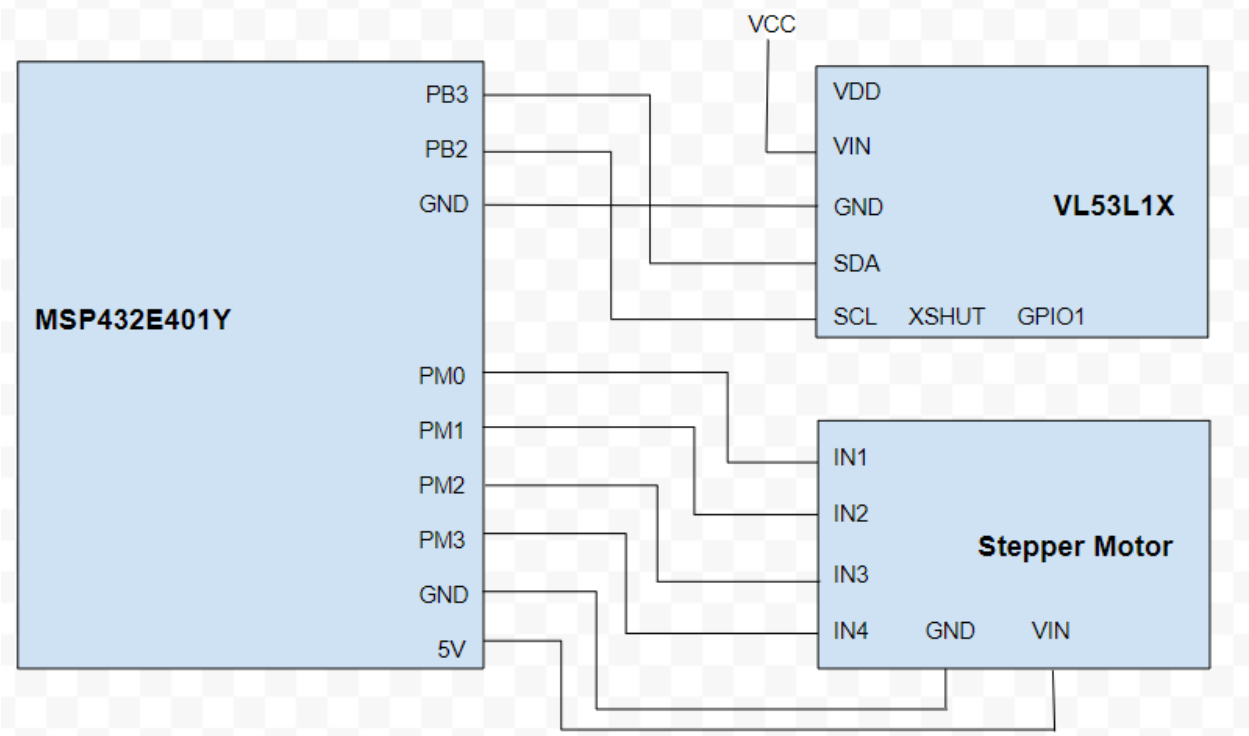


Figure 3: Circuit Schematic