2DX4: Microprocessor Systems Project
Final Project

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

Ahmed Allam – L07 – MACID 1 - allama2

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 **[Ahmed Allam, allama2, 400263198]**

Google Drive Link to Project Video

FinalProjectVideo_Part 1_Demo_allama2:
https://drive.google.com/file/d/1Cksb5qxkejtUeqcgkLdkldPofI6v3bkw/view?usp=sharing

FinalProjectVideo_Part 2_DesignQuestions_allama2:
https://drive.google.com/file/d/1jgoejH5MykNSbtEnLrrVyx7McEKL1sj4/view?usp=sharing

1. Device Overview

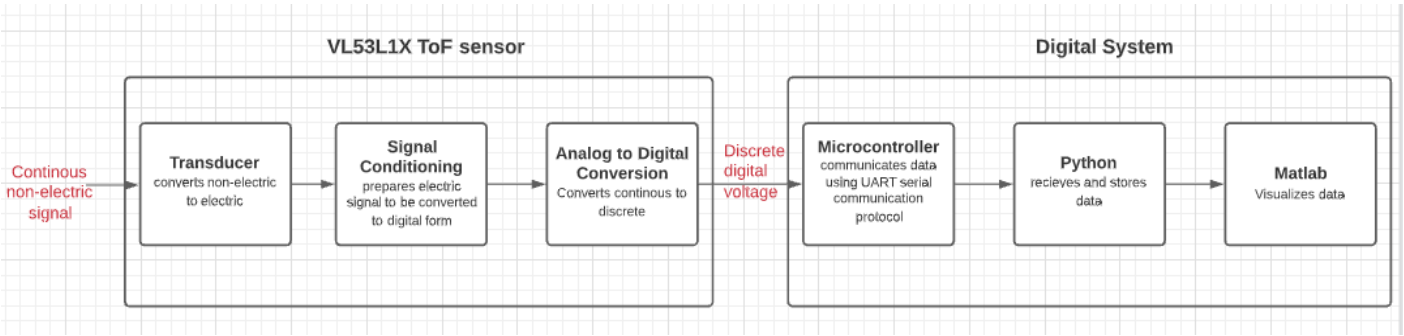
Features

- Obtains distances up to 360 cm using a ToF (Time of Flight) sensor
- Has short, medium, and long as distance ranging modes
- Capable of a full rotation in both clockwise and counterclockwise directions
- Produces a 3D configuration of the surrounding area
- Stores data
- Memory varies based on user's preferences but is in the order of KB
- Uses a bus speed of 30 MHz
- Requires an operating voltage of 5V for the stepper motor and 3.3V for the ToF sensor
- Setup cost of approximately $400
- Uses C, Python, and MATLAB programming languages
- Requires a baud rate of 115200 bps

General Description

The design consists of three main components, MSP432E401Y microcontroller, 28BYJ-48 stepper motor, and the VL53L1X ToF sensor. Other parts that are part of the final setup include two sided female jumper wires and a USB cable for a microcontroller connection to the computer.
The system runs on a C programmed keil code to perform a full rotation scan of the yz plane. Individual scans happen at a synchronized rate based on the user's choice of degrees per scan. Upon completion of a full scan, the device will rotate in the opposite direction to return back to its original position.
Using a python programmed code to do so, the distance measurements of the ToF sensor are saved into a xyz file. Multiple scans of the yz plane can be performed as the device ensures that no data is being overwritten once stored. Once all the information has been acquired, the user can implement a MATLAB code that plots the coordinates obtained from the measurements in a 3D configuration.

Block Diagram



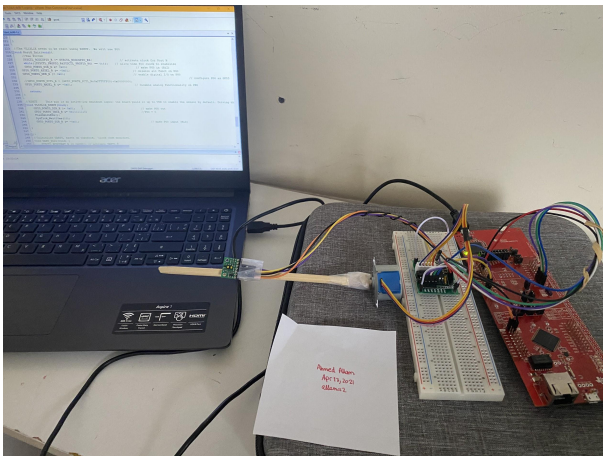


Figure 1: Device setup

```
Opening: COM3
output:Program Begins

output:2DX4 Program Studio Code 1

output:Model_ID=0xea , Module_Type=0xcc

output:ToF Chip Booted!

output:One moment...

output:writing configuration Successful.

output:StartRanging Successful.

output:check for data ready done.

output:SensorInit Successful.

output:StartRanging Successful.

output:1704

X, Y, Z:350 0.0 1704.0

output:1699

X, Y, Z:350 331.4584571054019 1666.3541914050886

output:1669

X, Y, Z:350 638.6986486173348 1541.9549397613375

output:1114

X, Y, Z:350 618.9052395838369 926.2571481050354

output:785

X, Y, Z:350 555.0788232314399 555.0788232314399

output:586
```

Figure 2: PC display of information in Python

2. Device Characteristic Table

Characteristic	Specification
Key Components	● MSP432E401Y microcontroller ● 28BYJ-48 stepper motor ● ULN2003 driver board ● VL53L1X ToF sensor
Pins	- Port L: Pins 0,1,2, and 3 - Port B: Pins 2,3 - 5V supply - 3.3V supply - Ground
Bus Speed	30 MHz
Serial Part	UART and I2C serial communication protocol
Communication Speed	Baud rate: 115200 bps

3. Detailed Description

Distance Measurement

The VL53L1X Time of Flight sensor integrates the use of a Single Photon Avalanche Diode (SPAD) array along with a 940 nm class 1 laser and infrared filters in order to produce the best ranging performance with a high accuracy. The device's features include the implementation of the latest ST technology which allows it to cover larger distance ranges regardless of the presence of

any ambient light or light reflections in the way. The user requires a MSP432E401Y microcontroller to communicate with the sensor's driver using the Application Programming Interface (API) allowing the transmit of data once acquired. Communication takes place following the controller and peripheral relationship. The peripheral (the sensor in this case) can be controlled to start/stop ranging, and configure the system accuracy. Distance can be obtained using the time of flight principle (Eqn 0.0) provided at the end of the page.

The main purpose of a microcontroller unit in this design is to maintain an ordered execution of the commands and data acquisition by the ToF sensor. The microcontroller manages such responsibilities as it controllers the sensor driver in a serial method of communication using I2C. This method of communication is considered asynchronous and thus requires no clock signals to control the execution process. It is controlled by the user instead through push buttons. Communication must also take place at a fixed baud rate which is set to be 115200 bps. This rate refers to how fast data transmission can take place from the sensor to the user application.

The system requires the use of keil software run in a C programmable language to set up the environment of the Cortex-M microcontrollers. The input received from the reset button by the microcontroller decides whether or not the code is run based on the state of the button. This brings up the finite state machine concept to analyze the two possible and the two possible states outlined based on this input. Once the reset button is pressed, initialization of some ports and the communication method must be done first to establish the grounds of the running code afterwards. It is the user's choice to configure which port that will connect the stepper motor to the microcontroller, in this design, port L was used. Initialization of both I2C and UART is also required. Initialization includes the acknowledgement of the address of the peripheral, enabling functions, configuring the sending/receiving data lines (SDA) and the synchronous data transfer line (SCL). I2C will be used to communicate between the ToF and the microcontroller, while UART will be used to communicate between the microcontroller and a data processing application such as Python.

After the environment setup has been completed, it is time to establish a connection between the microcontroller and the ToF sensor. After connecting the wires in the same setup as shown in the circuit schematic in figure 5, the model ID and the module type of the sensor are sent to the microcontroller to verify a successful connection as shown in the flowchart in figure 6. Upon verification, the microcontroller waits for a trigger that indicates the sensor is ready for activation. The interrupt is acknowledged and the ToF chip gets activated. The sensor is finally configured with the default setting and a StartRanging command is executed to enable the ranging. The microcontroller then instructs the stepper motor which is connected to the ULN2003 driver board to begin rotating in the clockwise direction. The stepper motor will rotate for a full 360 deg using the full step mode before coming to a step. As a result, the

*(0.0) distance = (photon travel time/2) * speed of light

number of iterations required is set to 512 with each iteration containing 4 steps to produce the total number of steps required to conduct a full rotation. The microcontroller relies on the polling method to continuously check if the desired angle is met before applying the ranging process. The user is allowed to choose at what angle ranging should take place. Once an angle is reached, the microcontroller uses the interrupt method to wait for the ToF to obtain the data until a trigger is initiated indicating a successful range. The two functions getRangeStatus() and getDistance() are returned once a scan is complete. GetRangeStatus() indicates whether or not a scan was successful while getDistance returns the integer distance value as the output of this part of the system in millimeters after using (Eqn 0.0). After all the angles were reached in a full rotation and ranging was successfully executed at each angle, a StopRanging command is passed to mark the end of the scanning process. Finally, keil executes another loop that ensures the device rotates in the counterclockwise direction to return back to its original starting point. The number of iterations in the loop is the same as the one used in the earlier loop, however, the steps in each iteration are put in reverse when compared to the steps in the earlier loop.

This was a detailed representation of the responsibilities of the microcontroller in this system and how it behaves as the mediator between the ToF sensor and the user application used to process the acquired data such as Python or RealTerm. With the help of Keil software, the user can now control the microcontroller which controls the other components in the design.

Visualization

The computer used for this display runs on a Windows 10 setup. Some of the computer specs include a total memory of 4Gb running at 2400 MHz speed. The computer's capacity is 60 Gb and the shared GPU memory is 2 Gb.

After the Time of Flight sensor acquires the distance measurements, data is transmitted to the microcontroller using the UART serial communication protocol which is indirectly connected to the application used to display the information processing whether it be RealTerm, python, or any other data processor. In this specific design, a python program was used to decode the data transmitted and output it in a human readable form as distances in millimeters. Pyserial library takes care of this conversion process. As the data is being received by the sensor driver and thereby python as well, a decode command is continuously used to display the messages and distances being processed. The data received is then filtered into integer inputs only as shown in figure 7 and ignores any other symbols or characters that come along in order to prevent an irregular input to be processed as part of the data. More importantly, once a distance has been transmitted from the ToF sensor to the python code, two equations are applied to determine the y and z coordinates of the given distance. The angle can be calculated between the device's starting position and the line where the scan took place (depends on the scan number in the full rotation) using (Eqn 1.0) provided at the end of the page. Sine and cosine rules require importing the math library and can then be used assuming a right triangle

to calculate the y and z coordinates using (Eqn 2.0) and (Eqn 3.0) respectively. The x coordinate is implemented manually based on the user's displacement after every scan. In this specific design, fixed displacement intervals of 35 cm was selected which divides the area scanned into equal sections using (Eqn 4.0)

A specific file of xyz format is then created to store the coordinates obtained from each measurement once it has been processed and broken down as points on the yz plane with a fixed x coordinate for each scan. Finally, the file is closed to store the coordinates of the full scan.

Further data processing takes place in matlab once the coordinates obtained from all the scans have been successfully stored. As pointed out in figure 8, the xyz file is loaded onto matlab using the "load" command. Three variables are then declared to represent the x, y, and z coordinates and every variable is assigned to its respective column from the xyz file. The final steps comes with using the "plot3" command which plots a 3D configuration of the three declared variables.

*(1.0) angle = 11.25*i for i=0,1,...,32
*(2.0) y-coordinate = distance * sin(angle*π/180)
*(3.0) z-coordinate = distance * sin(angle*π/180)
*(4.0) x-coordinate = 350*i for i=0,1,...,10

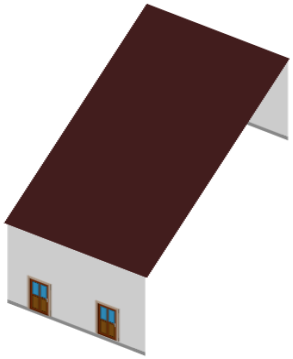


Figure 3: Rough sketch of the surrounding space. (Note: The ceiling is inclined)

4. Application Example

The following set of instructions will help guide the user to successfully set up and run the device. The instructions assume keil, python, and matlab algorithms are already provided with the design.

1) Assuming all the device components are already present (if not, please obtain the list of key components provided the device characteristic table), get 10 double sided female jumper wires.
2) Refer to the circuit schematic of the design provided in figure 5
3) Connect the stepper motor to the ULN2003 driver board by attaching its wires in the designated area (the white box).
4) Using a jumper wire, connect the negative port of the ULN2003 board to the ground on the microcontroller.
5) Using a jumper wire, connect the positive port of the ULN2003 board to a 5V supply on the microcontroller.
6) Using 4 jumper wires, connect In1-In4 to PL0-PL3 in that order.
7) The connection between the stepper motor and the microcontroller is now completed.
8) The next section deals with connecting the ToF sensor to the microcontroller.
9) Using a jumper wire, connect V_{in} from the ToF chip to 3.3V supply on the microcontroller.
10) Using a jumper wire, connect the GND port on the ToF chip to the ground on the microcontroller.

11) Using a jumper wire, connect the SDA port on the ToF chip to the PB3 on the microcontroller.
12) Finally, using a jumper wire, connect the SCL post on the ToF sensor to the PB2 on the microcontroller.
13) Once completed, the final setup of connections should look like this:

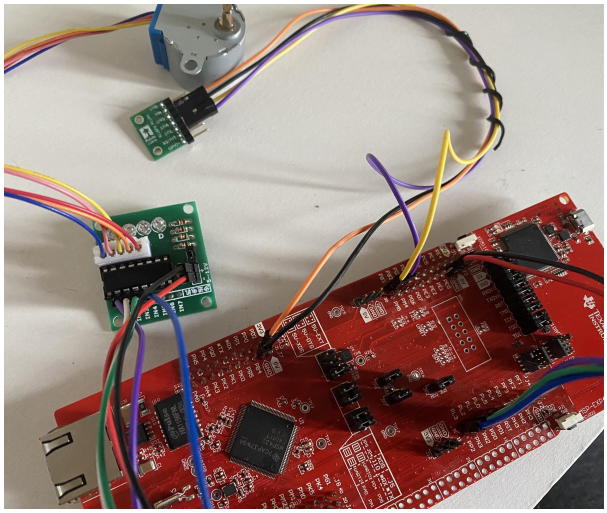


Figure 4: Circuit schematic of device

14) Obtain a USB cable and connect the microcontroller to the computer.
15) Remember the port number on the computer where the USB is connected as it will be used in later steps.
16) Open keil, modify the bus frequency if needed by changing the PSYDIV number in the PLL.h file and changing the multiplier number in SysTick.c
17) Open the main algorithm, quickly go over the code and compare it to the detailed description provided above to ensure no errors exist.
18) Modify the angle at which scans take place only for testing processes (keep 11.25 deg as the default).
19) Translate, build, and load the code onto the microcontroller.
20) Do not press run yet!
21) Open the Python algorithm.
22) Check the port number used is the right number. If not, modify the code at the line 14 (first parameter) to open the port corresponding to the USB cable.
23) Ensure the baud rate is set to 115200 bps at line 14 (second parameter).
24) The user can change the name of the xyz file if they feel the name is not very descriptive of the purpose of the scans.
25) Modify the range of the for loop only if the angle was changed.
26) Save any changes if any.
27) Run the module. The Python shell will automatically open.
28) Place the stepper motor on its side with the blue side pointing towards the user.
29) Hold the stepper motor in place.
30) Ensure that the ToF is placed pointing towards the ceiling as its initial position.
31) Press on the reset button on the microcontroller.
32) Wait until the ToF makes a one complete rotation and rotates back to its starting position.
33) Note: this scan was attempted in the yz plane.
34) The user can now move forward or backward along the x axis based on their choice. (If decided to move, make sure the x coordinate is manually updated based on the displacement taken).
35) Repeat steps 26-31 to acquire more scans.

36) Upon completion of the scans, the user can now switch to matlab for the 3D visualization of the data.
37) Open the matlab algorithm.
38) Load the xyz file by adding the path where the file was stored.
39) The user has the choice to change the x,y,z, or title labels
40) Save the changes.
41) A 3D plot will be displayed representing a 3D configuration of the surrounding area.
42) This concludes the instructions required for acquiring and mapping the signals.

5. Limitations

1) Most computers dealt with these days are binary computers. As a result, floating points can not be represented with the same accuracy since they are usually outputted in a base 10 format and there will usually be a quantization error due to the rounding of the floating number to be printed as its nearest binary representation. This was seen in Matlab and Python part of the design. The y and z coordinates get rounded to the nearest binary representation in Python while they get rounded to only 4 decimal places in Matlab which can be very essential in small measurement designs.
2) The maximum quantization error is the maximum range of measurement/ 2^(the number of bits available to obtain the distance in the ToF sensor). The sensor is set to long mode and therefore a total range of 0-4000mm. The number of bits used to acquire data in the time of flight sensor is 16 bits. The maximum quantization error is thus 4000/(2^16) = 0.061 mm/bit.
3) The maximum standard serial communication rate to implement with this laptop is 128k bps. It was verified by going to the device manager app and confirming this number.
4) The communication method that took place between the microcontroller and the ToF sensor relied on the I2C serial communications protocol. The microcontroller uses a half duplex method of communication, as it sends the synchronized clock signals to the ToF sensor and in return it receives the distance measurements obtained from the ToF. The speed at which this communication happens at 100 kbps according to the VL53L1X datasheet.
5) When analyzing the entire design, multiple components pose an impact on the speed of data transmission and processing. However, the ToF affects the speed the most based on its contribution and responsibilities in the system. The angle at which data acquisition is very important in influencing the overall speed of ToF sensors. Small angle measurements can be accurate but require more time and thus forces the motor to take longer periods of time in each rotation which ends up in heating the motor. Larger angles require less time but are less accurate which can be a problem when looking at it from the Nquist rate point of view as it might fail to capture small changes in the surroundings

6. Circuit Schematic

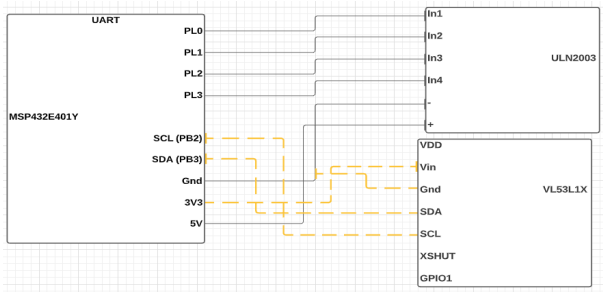


Figure 5: Circuit schematic of the system

7. Flowcharts

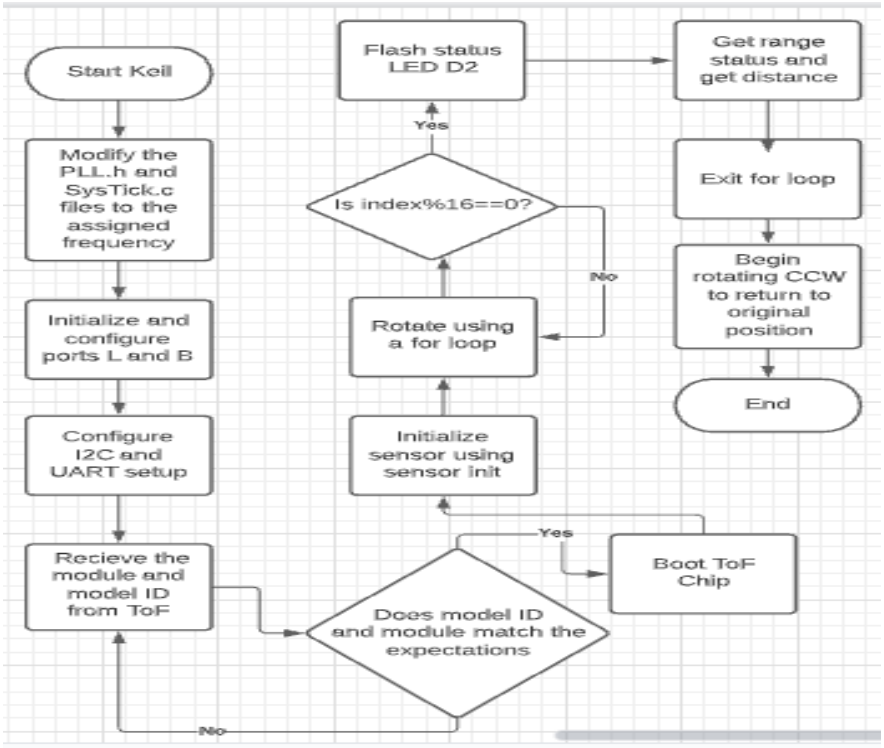


Figure 6: Keil flowchart

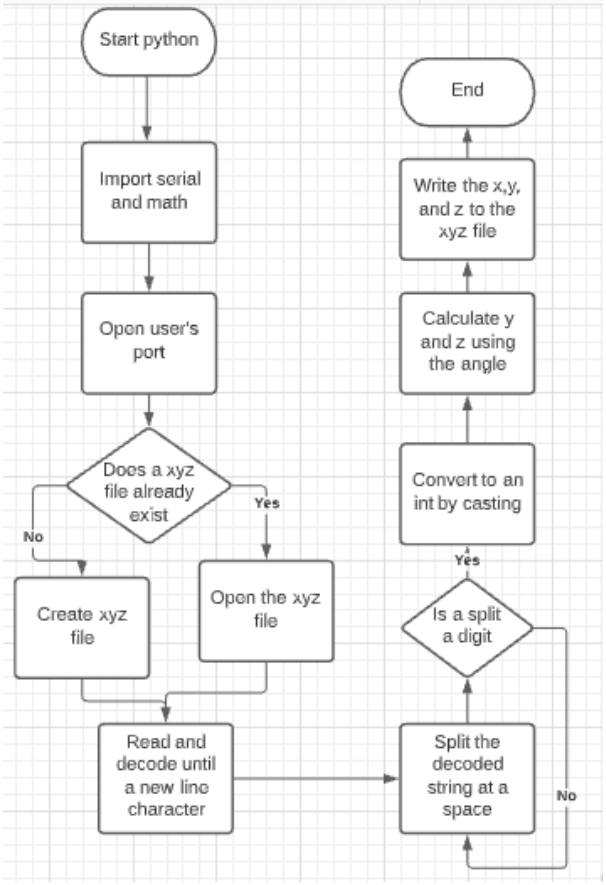


Figure 7: Python flowchart

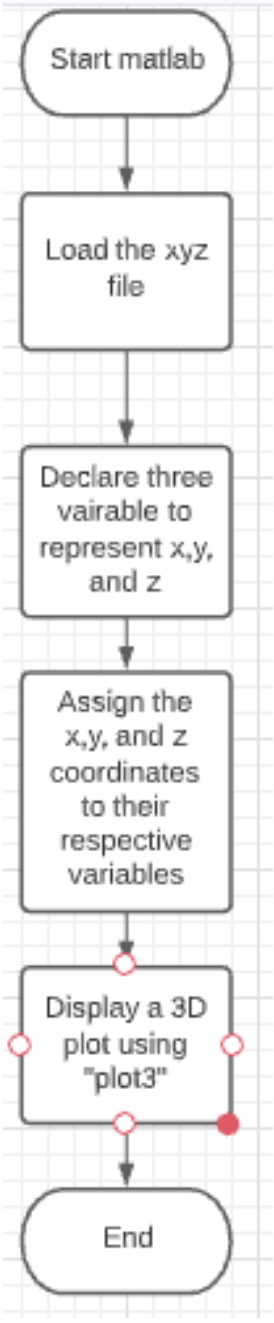


Figure 8: Matlab flowchart

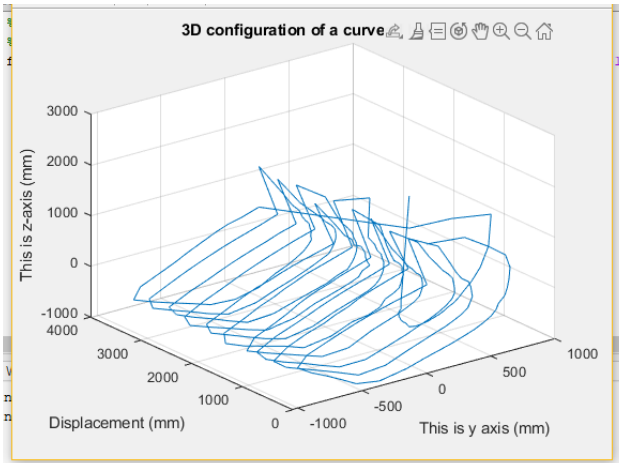


Figure 9: 3D configuration of the area used in the demo