ELEC230 – Laboratory 11 – Sensor Interfacing

Introduction

This week you will be interfacing the first of two sensor units to your Raspberry Pi Zero, which is required to complete Assignment 2. This first sensor is the VL53L1X rangefinder that was introduced in last week's lecture. The second sensor will be an inertial measurement unit (IMU): a combined accelerometer, rate gyro and magnetometer. By the way, after today's lab, there's probably never been a better time to start building up your understanding of sensors by reading the recommended textbook [1]. See the instructions on Canvas to help with access. You could start with Chapter 4.

Documenting work for later use in your Report

Throughout, document your progress in preparation for your later Report. You can do this by taking screenshots of your terminal in MobaXterm, and the various commands and their results (but include the full PC desktop with taskbar showing date/time, as usual); and by taking photos.

Part I: Interfacing the VL53L1X

Firstly a note on laser safety. The VL53L1X is considered eye safe as long as no optics are used to focus the cylindrical cone produced by the laser emitter. Therefore do not put any optics in front of the VL53L1X which may focus the laser. Also it is good practice to **NOT LOOK directly** at the laser, it shouldn't cause any harm but it's not worth the risk.

As indicated in the lecture on the I2C bus, there are two active signals, the Serial Data line (SDA) and Serial Clock line (SCL). Figure 1 shows the Schematic Diagram of the VL53L1X-SATEL board that you will be using. The board can operate off a 3.3V power supply that can be provided by the Pi-Zero.

The pin connections on the VL53L1X you need to use are: pin 2 (SCL), pin 4 (SDA), pin 6 (GND), pin 5 (VDD, 3.3V). Take your time and also look at Figure 2 and the surrounding information first to orientate yourself. Making these connections is nowhere near as easy as it looks, **particularly since the schematic (Figure 1) flips the relevant columns around!** Make a note of which colour jumper wire is connected to which pin, for use at the next step: connecting up to the Pi-Zero. **Do NOT proceed to power on your Pi-Zero until you are absolutely sure that you have got all your connections correct!**

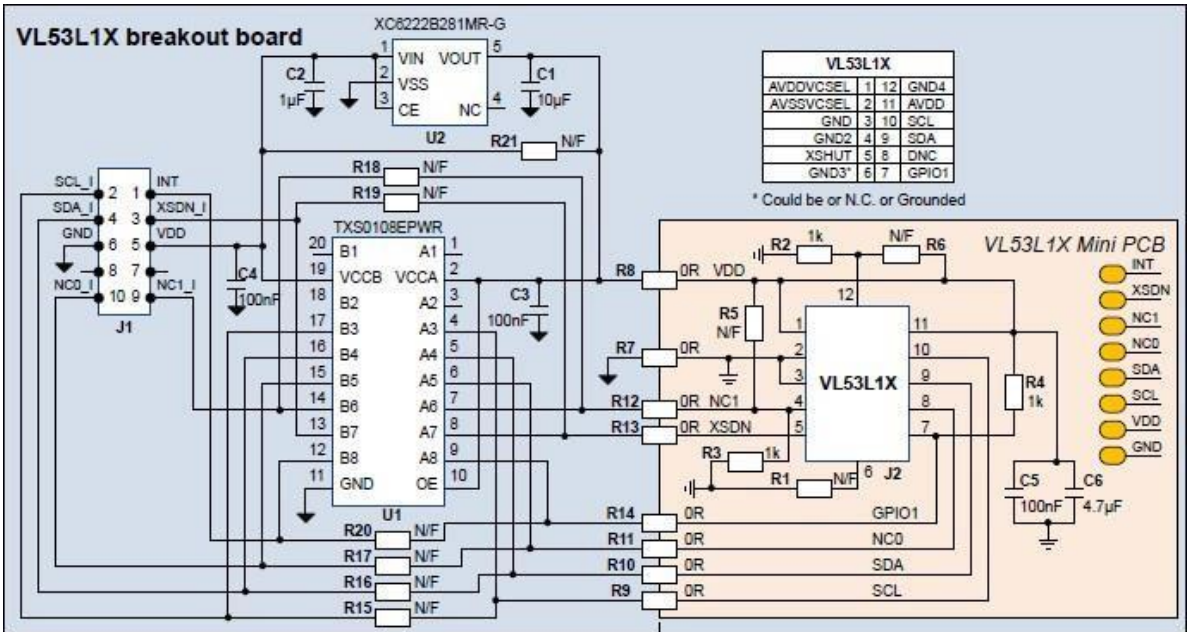

Figure 1 - VL53L1X-SATEL Schematic Diagram

Figure 2 shows a photograph of the VL53L1X-SATEL hardware. Pin 1 can be identified by the “square” solder pad which is the bottom left pad in Figure 2. To allow the connection of wires between the

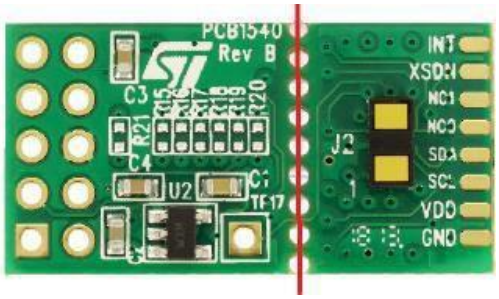

Figure 2 - VL53L1X-SATEL Hardware

VL53L1X-SATEL and the Raspberry Pi Zero W, a 10 pin header has been soldered for you onto the front side of the VL53L1X-SATEL board shown in Figure 2. Thus, the pins protrude from the rear of the board, and female-female cables can be used to form the connections between the two boards. So, let’s now look at how you can complete the connection on the side of the Raspberry Pi Zero W.

You should use the following pins on the Pi-Zero: pin 1 (3.3V), pin 6 (GND), pin 3 (SDA), pin 5 (SCL) to connect to the equivalent connections on the VL53L1X-SATEL using the female-female wires provided. Figure 3 show the Pin connections on the Raspberry Pi Zero W that you are using. Again pin 1 is indicated by the “square” solder pad located as the top left connection in Figure 3. However you may not be able to see this on your Raspberry Pi Zero W as the connector is soldered in place.

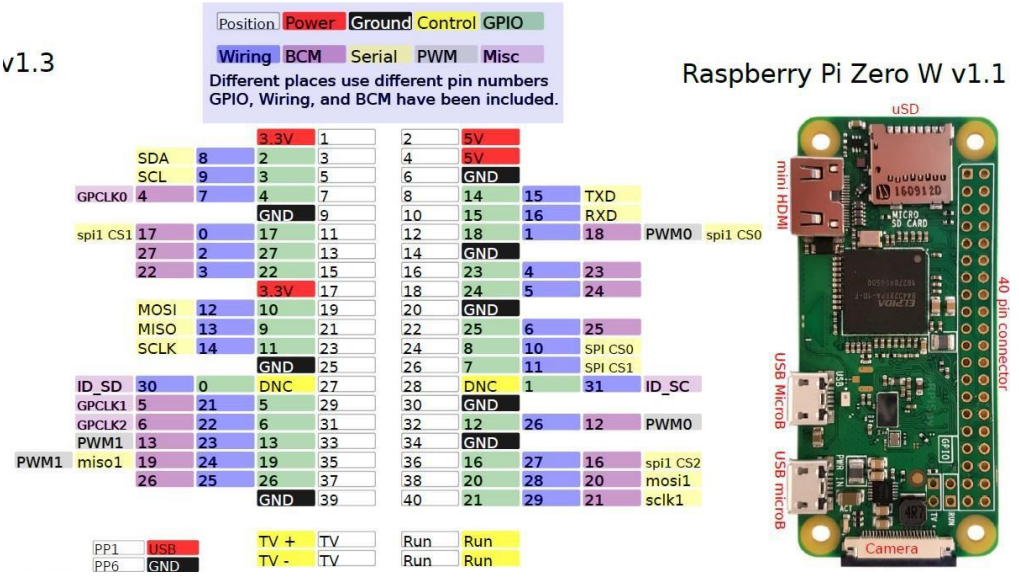

Figure 3 - Raspberry Pi Zero W Pin Connections

Double-check that your connections are correct before you power up your Raspberry Pi Zero W. Hopefully your Raspberry Pi Zero W will boot up correctly. **If the Pi-Zero or more likely, the VL53L1X-SATEL board starts to feel hot, disconnect the Pi-Zero power immediately and re-check that you have the right connections**. It’s easy to make a mistake! The next stage is to configure the I2C bus on the Raspberry Pi and install some utility programmes so that the I2C bus can be analysed.

Enabling the I2C bus

The command:

```
sudo raspi-config
```

is used to enable the I2C bus. Select “5” Interface Options, Select “5” I2C and enable the Device. You should see screens like those shown in Figure 4. After enabling the I2C bus, reboot your Raspberry Pi.

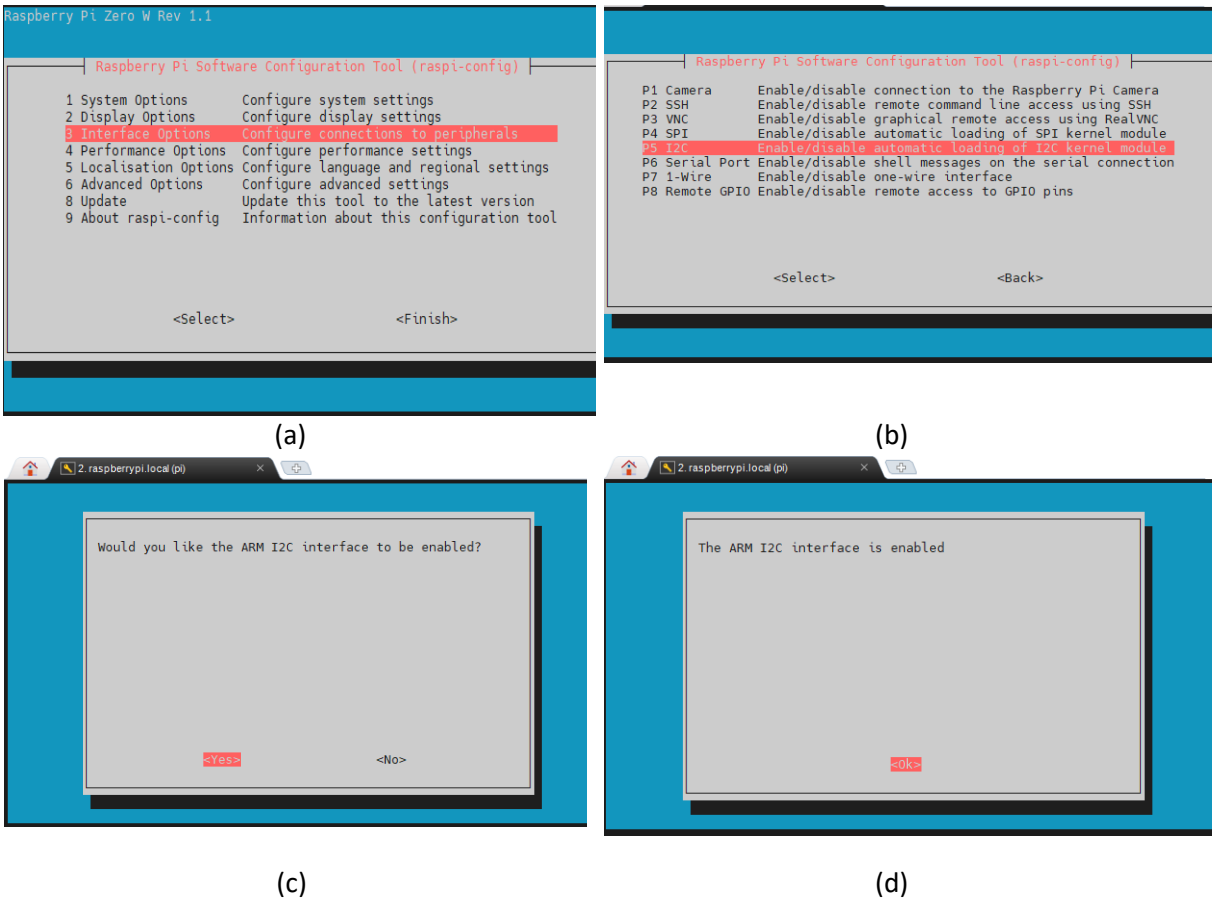

Figure 4 - sudo raspi-config

The I2C bus interface should now be enabled. (Back in the home menu, press the right arrow key twice and press Enter to select <Finish>.) You should be able to see the i2c-1 bus in the "/dev"directory as shown in Figure 5

```
pi@raspberrypi:~ $ ls /dev
autofs           gpiomem   loop-control        ram0   ram9     tty12  tty26  tty4   tty53  ttyAMA0   vcs6      vcsu5
block            hwrng     mapper              ram1   random   tty13  tty27  tty40  tty54  ttyprintk vcsa      vcsu6
btrfs-control    i2c-1     mem                 ram10  raw      tty14  tty28  tty41  tty55  uhid      vcsa1     vhci
bus              initctl   memory_bandwidth    ram11  rfkill   tty15  tty29  tty42  tty56  uinput    vcsa2     video10
cachefiles       input     mmcblk0             ram12  serial1  tty16  tty3   tty43  tty57  urandom   vcsa3     video11
char             kmsg      mmcblk0p1           ram13  shm      tty17  tty30  tty44  tty58  v4l       vcsa4     video12
console          log       mmcblk0p2           ram14  snd      tty18  tty31  tty45  tty59  vchiq     vcsa5     watchdog
cpu_dma_latency  loop0     mqueue              ram15  stderr   tty19  tty32  tty46  tty6   vcio      vcsa6     watchdog0
cuse             loop1     net                 ram2   stdin    tty2   tty33  tty47  tty60  vc-mem    vcsm      zero
disk             loop2     network_latency     ram3   stdout   tty20  tty34  tty48  tty61  vcs       vcsm-cma
fb0              loop3     network_throughput  ram4   tty      tty21  tty35  tty49  tty62  vcs1      vcsu
fd               loop4     null                ram5   tty0     tty22  tty36  tty5   tty63  vcs2      vcsu1
full             loop5     ppp                 ram6   tty1     tty23  tty37  tty50  tty7   vcs3      vcsu2
fuse             loop6     ptmx                ram7   tty10    tty24  tty38  tty51  tty8   vcs4      vcsu3
gpiochip0        loop7     pts                 ram8   tty11    tty25  tty39  tty52  tty9   vcs5      vcsu4
pi@raspberrypi:~ $
```

Figure 5 - Result of "ls /dev" showing the "i2c-1" bus

Installing i2c-tools

To install the i2c-tools you should first update the system and then install the tools using these commands:

```
sudo apt-get update
sudo apt-get install i2c-tools
```

Once the i2c-tools are installed you can "probe" the i2c bus to determine what devices are connected. The command to use is

```
i2cdetect -y 1
```

which should result in device 0x29 being identified if your VL53L1X is correctly connected, as shown in Figure 6. You can see the options for i2cdetect by examining the "man" page for i2cdetect by executing

```
man i2cdetect
```

In this example the i2c-1 bus is probed by passing "1" to the i2cdetect command.

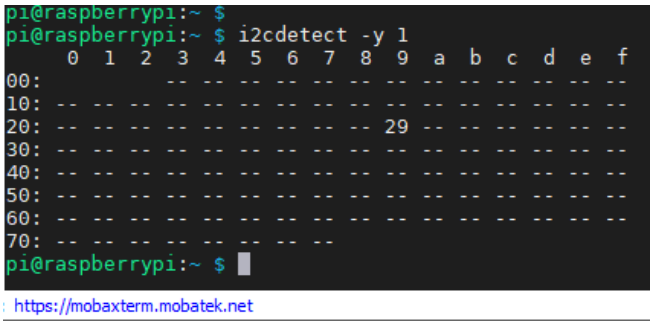

Figure 6 - Results from i2cdetect - y 1

Installing VL53L1X example software

If you search the internet for software for the VL531X sensor you will mostly find code for the Arduino or code written in Python. However you can download some demo C++ code from the Waveshare wiki:

https://www.waveshare.com/wiki/VL53L1X_Distance_Sensor

The compiled version of the Waveshare libraries require the "wiringpi" libraries to be installed which should be done using the following command.

```
sudo apt-get install wiringpi
```

If you install 7zip using Install University Applications, you can download and extract the "VL53L1X-Distance-Sensor-Demo.7z" (.7z is a compressed archive file format i.e. a zip file) from the Waveshare wiki by selecting "Demo Code" from the resources section. If you find this difficult, look on Canvas in Week 11 on a page entitled 'Raspberry' where the files have been placed. Follow instructions there.

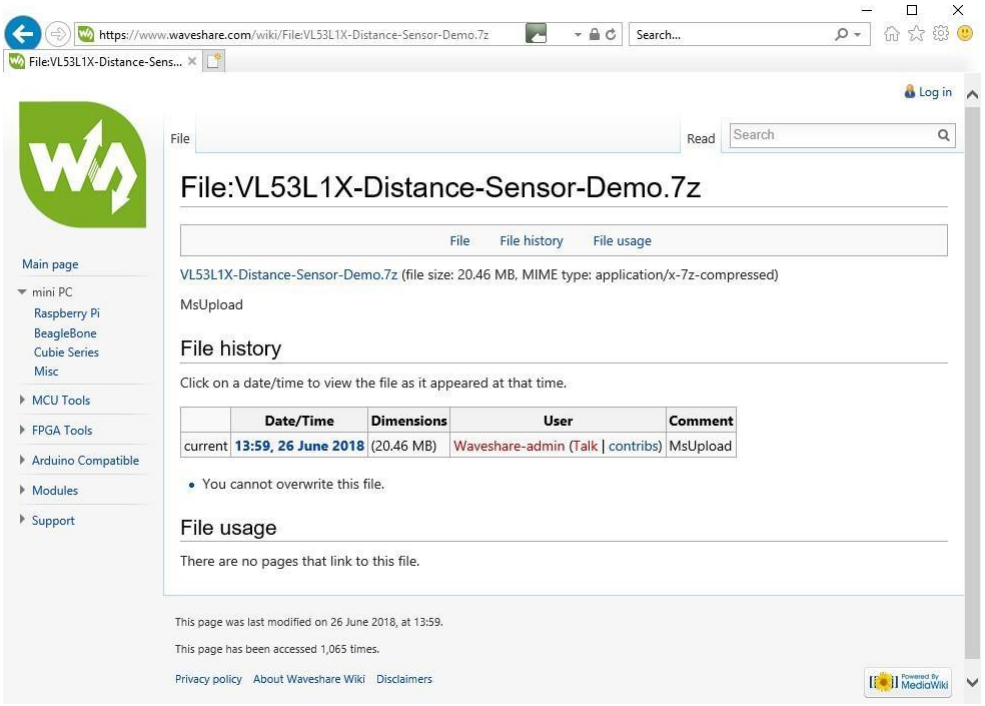

Figure 7 - Waveshare "Demo Code"

If you download all the files from Canvas to your MWS PC, then you need to copy them into a new “Raspberry” directory you have created on your Raspberry Pi. Or if you downloaded and extracted with 7zip, then just find the ‘Raspberry’ directory and copy it straight over. The “Range” file in this directory is a precompiled executable, however it doesn’t have execute permissions. You can give it executable permissions usingthe “chmod +x Range” command as shown in Figure 8.

```
pi@raspberrypi:~/Raspberry $ ls -la
total 316
drwxr-xr-x  2 pi pi   4096 Dec  7 11:33 .
drwxr-xr-x 25 pi pi   4096 Dec  7 11:33 ..
-rw-r--r--  1 pi pi    109 Dec  7 11:33 Makefile
-rw-r--r--  1 pi pi  14676 Dec  7 11:33 Range
-rw-r--r--  1 pi pi    817 Dec  7 11:33 Range.cpp
-rw-r--r--  1 pi pi 267625 Dec  7 11:33 vl53l1_register_map.h
-rw-r--r--  1 pi pi  15212 Dec  7 11:33 VL53L1X.cpp
-rw-r--r--  1 pi pi   3230 Dec  7 11:33 VL53L1X.h
pi@raspberrypi:~/Raspberry $ chmod +x Range
pi@raspberrypi:~/Raspberry $ ls -la
total 316
drwxr-xr-x  2 pi pi   4096 Dec  7 11:33 .
drwxr-xr-x 25 pi pi   4096 Dec  7 11:33 ..
-rw-r--r--  1 pi pi    109 Dec  7 11:33 Makefile
-rwxr-xr-x  1 pi pi  14676 Dec  7 11:33 Range
-rw-r--r--  1 pi pi    817 Dec  7 11:33 Range.cpp
-rw-r--r--  1 pi pi 267625 Dec  7 11:33 vl53l1_register_map.h
-rw-r--r--  1 pi pi  15212 Dec  7 11:33 VL53L1X.cpp
-rw-r--r--  1 pi pi   3230 Dec  7 11:33 VL53L1X.h
pi@raspberrypi:~/Raspberry $
```

Figure 8 - Giving "Range" executable permissions

You can now execute the “Range” program using “./Range”. This should then show the range measurements as shown in Figure 9. Point the sensor at different targets to see the range change.

```
pi@raspberrypi:~/Raspberry $ ./Range
Distance(mm) :1804
Distance(mm) :1816
Distance(mm) :1802
Distance(mm) :1795
Distance(mm) :1795
Distance(mm) :1803
Distance(mm) :1789
Distance(mm) :1785
Distance(mm) :1791
Distance(mm) :1780
Distance(mm) :1791
Distance(mm) :1823
Distance(mm) :1811
```

Figure 9 - Running the "Range" program

Note: if at a later stage you want to rebuild the “Range.cpp” program that incorporates the “VL53L1X.cpp” program, you can use the “Makefile” provided. First delete the existing “Range” program using “make clean” and then rebuild the program using “make” as shown in Figure 10. **You shouldnow take some time to examine the programs to understand how they operate.**

```
pi@raspberrypi:~/Raspberry $ make clean
rm Range
pi@raspberrypi:~/Raspberry $ make
g++ -Wall -o Range Range.cpp VL53L1X.cpp -lwiringPi
Range.cpp: In function 'int main()':
Range.cpp:15:9: warning: variable 'status' set but not used [-Wunused-but-set-variable]
    bool status;
         ^~~~~~
Range.cpp:19:7: warning: variable 'rate' set but not used [-Wunused-but-set-variable]
   int rate = 0;
       ^~~~
VL53L1X.cpp: In member function 'bool VL53L1X::begin(uint8_t)':
VL53L1X.cpp:107:64: warning: suggest parentheses around comparison in operand of '&' [-Wparentheses]
   while (readRegister16(VL53L1_FIRMWARE__SYSTEM_STATUS) & 0x01 == 0)
                                                            ~~~~^~~~
VL53L1X.cpp: In member function 'void VL53L1X::startMeasurement(uint8_t)':
VL53L1X.cpp:129:16: warning: variable 'result' set but not used [-Wunused-but-set-variable]
 { unsigned int result;
                ^~~~~~
VL53L1X.cpp:130:11: warning: unused variable 'address' [-Wunused-variable]
   uint8_t address = 1 + offset; //Start at memory location 0x01, add offset
           ^~~~~~~
VL53L1X.cpp: In member function 'bool VL53L1X::newDataReady()':
VL53L1X.cpp:188:12: warning: variable 'result' set but not used [-Wunused-but-set-variable]
   uint16_t result;
            ^~~~~~
pi@raspberrypi:~/Raspberry $
```

Figure 10 - Rebuilding "Range"

Part II: Experimental work

Much of the following text and figures are directly taken (and occasionally adapted) from the following excellent online article by Joshua Hrisko (2019): https://makersportal.com/blog/2019/4/10/arduino-vl53l1x-time-of-flight-distance-measurement [2].

Introduction

Time of flight (ToF) approximates the time it takes a travelling wave to come into contact with a surface and reflect back to the source. Time of flight has applications in automotive obstacle detection, resolving geographic surface composition, and computer vision and human gesture recognition. In the application here, the VL53L1X ToF sensor will be used to track the displacement of a ping pong ball falling down a tube. We can predict the acceleration and behaviour of a falling ping pong ball by balancing the forces acting on the ball, and ultimately compare the theory to the actual displacement tracked by the time of flight sensor.

Time of flight theory

Time of flight uses the basic principles of Newtonian physics by assuming perfectly elastic contact with a surface. Most inexpensive ToF sensors use amplitude modulation to emit a singular or series of pulses at a particular frequency. For a laser[1], the frequency is usually in the infrared spectrum to ensure that the measurement is not visible to the human eye, just as an ultrasonic pulse is above the audible range. For an object at a distance x away from the light emitter (laser), we can approximate the reflection time for the light to hit the object and return back to the emitter using distance = speed x time:

$$2x = c\Delta T$$

where x is the distance to the object ($2x$ since the signal has to reflect back), c is the speed of the wave (light here), and ΔT is the time it takes to travel $2x$. This process is also shown in Figure 13.

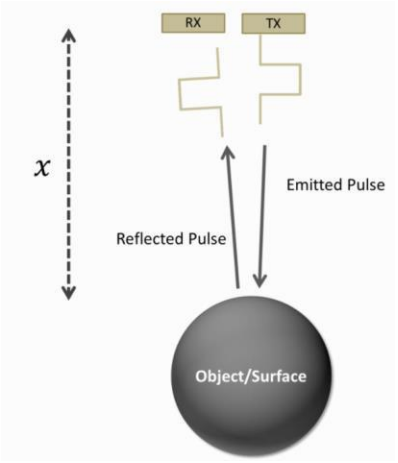

Figure 13 – Laser time of flight; from [2]

The width of the signal pulse limits the minimum detectable distance, and is often the working limit of the ToF module. This is a limitation of hardware which produces and reads short pulses of light. For example, pulse widths in the nano or pico second range are typically capable of resolving distances in the meter and millimeter ranges, respectively. The pulses are repeated hundreds or thousands of times to increase the sensor's ability to measure longer distances. Therefore, the longer the desired distance, the longer the pulse 'train' needs to be. This means that either the individual pulses need to be shorter or the pulse train needs to be longer to allow for longer reflections. More on ToF sensors and their limitations can be found here [3]. The VL53L1X has a maximum range of 4m.

[1] LASER: Light Amplification by the Stimulated Emission of Radiation

SAFETY - IMPORTANT

Some of this section is reproduced directly from ST Microelectronics' VL53L1X datasheet.

The VL53L1X contains a laser emitter and corresponding drive circuitry. As a precaution, **remember never to look directly into the 'line of sight' of the laser emitter, or point it at your own or anybody else's eyes. The laser output power must not be increased by any means and no optics should be used with the intention of focusing the laser beam.**

According to the manufacturer, ST Microelectronics, the laser output is designed to remain within Class 1 laser safety limits under all reasonably foreseeable conditions including single faults in compliance with IEC 60825-1:2014 (third edition).

The laser output remains within Class 1 limits as long as:

- the ST Microelectronics' recommended device settings (driver settings) are used
- the right operating conditions are respected (particularly the maximum timing budget, as described in the VL53L1X API user manual UM2356).

In short, while this is a safe device, please respect it and don't mess with it (for example, don't attempt to reverse engineer it or control aspects of its performance which are not designed to be controlled by the user), don't look directly at the laser emitter and don't try to focus the laser through any optics.

Experimental setup

The basic setup for the experiment consists of a ping pong ball, VL53L1X sensor[2], and a ~30cm acrylic tube. You will drop the ball from or near the top of the tube, and the VL53L1X will record the time and displacement of the ball as it travels down the tube due to gravity. This data will be displayed on screen via your Pi-Zero. The setup of the ball, tube, and ToF sensor is shown in Figure 14. (Note that even though the VL53L1X pulse emitter and receiver are shown as if pointing out of the page, in reality you will have them pointing down, at the ping pong ball, to track its progress.) You do not have your own acrylic tube; use a common one provided in the lab, making sure to put on a pair of disposable gloves first.

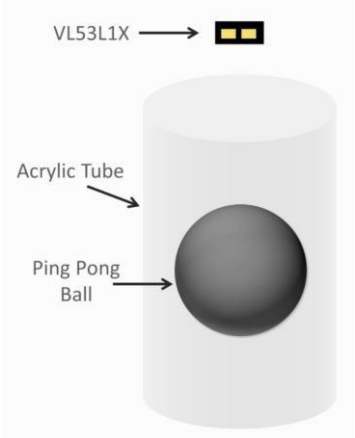

Figure 14 – Experimental setup; from [2]

Carrying out your measurements

You should drop the ping pong ball down the tube, while pointing the VL53L1X-SATEL emitter/receiver downwards with your other hand, held in a constant position above the tube. The VL53L1X can take

[2] This datasheet may be helpful, although it is for the VL53L1X (as opposed to the VL53L1X-SATEL we're using) so some points are different e.g. the pinout.

measurements at anything from 20 to 50 Hz (but does it achieve this in practice with the program you're using? see below…) so you should get a good few measurements in the short time it takes the ping pong ball to fall and bounce until it is at rest at the bottom of the tube.

Have several goes; gather several rounds of data and save it. You are going to use **five** of your takes to plot five graphs of displacement versus time, and incorporate these in your eventual Assignment 2 Report. You will need to compare your graphs to the theory given below.

As with the earlier work, you should also take relevant screenshots of your terminal in MobaXterm showing the range data (but include the full PC desktop with taskbar showing date/time, as before) – as evidence of your progress.

The measurement rate

With the C++ demo program you've been using, the VL53L1X is operating at a currently unknown rate. Although the formal Assignment 2 Brief hasn't been released yet, it will help you get good marks in the assignment if you can eventually edit and re-make the demo program to calculate the measurement rate and print it to the command line interface. Another challenge that will help your marks is to edit the program so that a useful and reliable as possible measurement time is printed with each range measurement.

Saving your measurements and beginning to compose your Assignment 2 Report

At the moment, you've probably been saving the output of the VL53L1X by using the 'save terminal output' function within MobaXterm, or by simply copying and pasting it from the terminal. Although Assignment 2 itself hasn't been released yet, it will help you get good marks in the assignment if you can find a way to manage data-saving to a text file by means of C++ code. (If you get some inspiration for this from an online source, that's fine – just make sure to properly reference it in your Report.)

IMPORTANT: Shutting down your Pi-Zero SAFELY before disconnection

For a proper shutdown, you need to observe the following command and procedure:

```
sudo shutdown -h now
```

- **This command shuts down your Pi.**
- **WAIT for the LED to stop flickering on the Pi, then wait 5 more seconds before unplugging.**
- **Using this command guards against accidentally interrupting background processes – a bit like shutting down your PC properly rather than just removing the battery...**

Theory: Drag Profile of a ball falling down a tube

Applying Newton's law leads to the following:

$$\sum F = F_g - F_D - F_B$$

where F_g is the force due to gravity, F_D is the drag force, and F_B is the buoyant force, as illustrated in Figure 15.

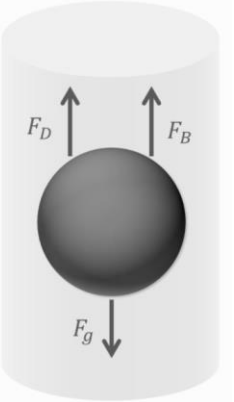

Figure 15 – Forces on the ping pong ball; from [2]

The equation above can be expanded and written in its explicit form:

$$m\frac{dv}{dt} = mg - \frac{1}{2}\rho_f A C_D v^2 - \rho_f g V$$

where m is the mass of the ball, v is the velocity of the ball, g is gravitational acceleration, ρ_f is the density of air, A is the aerodynamic area of the ball susceptible to drag, C_D is the drag coefficient, and V is the volume of the ball. We can further rearrange the differential equation above and simplify a few parameters:

$$\frac{dv}{dt} + \alpha v^2 - \beta = 0$$

with the constants α and β defined as:

$$\alpha = \frac{1}{2} \cdot \frac{\rho_f A C_D}{\rho_S V}$$
$$\beta = (1 - \frac{\rho_f}{\rho_s}) \cdot g$$

The differential equation has a solution of the form:

$$v(t) = \sqrt{\frac{\beta}{\alpha}} \tanh\left(\sqrt{\alpha\beta} \cdot (C_0 + t)\right)$$

As an initial condition, we know the ball is not moving, so we can quantify the constant C_0:

$$v(0) = \sqrt{\frac{\beta}{\alpha}}\tanh(\sqrt{\alpha\beta} \cdot C_0) = 0$$

$$C_0 = 0$$

Therefore, our final velocity expression is:

$$v(t) = \sqrt{\frac{\beta}{\alpha}}\tanh(\sqrt{\alpha\beta} \cdot t)$$

Now, in order to approximate distance, we need to integrate the velocity, otherwise known as dx/dt:

$$v(t) = \frac{dx}{dt} = \sqrt{\frac{\beta}{\alpha}}\tanh(\sqrt{\alpha\beta} \cdot t)$$

This looks easier to integrate than the original differential equation, and we can obtain an expression for vertical displacement as a function of time, for a falling ping pong ball in a tube:

$$x(t) = \frac{1}{\alpha}\ln(cosh(\sqrt{\alpha\beta} \cdot t))$$

For future reference…

Whilst both sensor units interface via the I2C bus, there seems to be an issue with the pull-up resistors used on the I2C bus in that the Raspberry Pi Zero W has pull-up resistors of 1.8 kΩ and the accelerometer unit has pull-up resistors of 4.7 kΩ to give a combined resistance of 1.3 kΩ. While the value of 1.3 kΩ is within specification (anything between 1.0 kΩ and 10 kΩ should be acceptable) prior tests have shown that it does not work reliably using the cables provided, therefore (pending further testing) another I2C bus will likely be configured when we come to interfacing the IMU.

References

[1] Siegwart, R., Nourbakhsh, I.R. and Scaramuzza, D., 2011. Introduction to autonomous mobile robots. MIT press.
[2] Hrisko, J. (2019, online). 'Arduino + VL53L1X Time of Flight Distance Measurement'. Available from *https://makersportal.com/blog/2019/4/10/arduino-vl53l1x-time-of-flight-distance-measurement*. [Accessed 10th December 2021.]
[3] Sarbolandi, H., Plack, M. and Kolb, A., 2018. Pulse based time-of-flight range sensing. Sensors, 18(6), p.1679.

Version history:

Dr. D. McIntosh & Dr. D. McGuiness, Dec. 2021; revised script and incorporation of ping pong ball experiment.
Prof. J.S. Smith, 2019-20; original script including interfacing instructions.