ENG5027: Digital Signal Processing - Assignment 3
IIR Filters

Zeynep Oner - 2693256O
Vishwa Poswal - 2699522P
Zainab Meerza - 2361575M

University of Glasgow

QUESTION 1

Vehicle Detection Traffic Light System

Street lighting systems are used for outdoor lighting the world over. However, a lot of power is lost when street lights are always on at maximum intensity. According to Tannous et al. (2018), outdoor light systems account for 8% of electricity on a global scale. In addition to electricity consumption, as most of this power comes from non-renewable sources, it also has an impact on sustainability and climate change.

A possible method for reduction of consumed power is to introduce feedback type control in street lights to minimise operation time, and thus to reduce electricity consumption. These systems sense the presence of cars or pedestrians to adjust the intensity of light, to create a tapering effect such that the lights closest to the user operate at 100% intensity, and as the distance between the street light and the user increases, it's intensity tapers down. Lower intensity operation means lower power consumption.

According to Juntunen et al (2018), such systems could reduce electricity consumption by 70%, and have been suggested by British Standard. However, a major challenge for implementation of such systems is that the sensed signal needs to be processed in real time. This means that the sensor response needs to be ingested and processed, and the system must react to the processed signal in real time.

In this project, this challenge was tackled. A feedback circuit was created to sense vehicles and pedestrians on the road. By implementing this circuit in street lights, the distance between a car and certain lights can be calculated, and intensity of the light can be determined.

The circuit comprised of an ultrasonic sensor and an Arduino. The Arduino was controlled in real time using the Standard Firmata. The incoming signal was processed in real-time using Python. The ultrasonic sensor was used for distance measurement, and the Arduino was used to read the data and show in software. The ultrasonic sensor worked by sending an echo signal. The echo signal reflected off the surface of the nearest object and and was read by the sensor. If an object was sensed, the returned signal was 0, if not object was sensed, the returned signal was a pulse. The electronic setup and the real-time code was tested on a 1:100 replica of the road.

When the Arduino was connected to the computer and the computer was connected to a power supply, a 50Hz noise was observed in the trigger signal. The 50Hz noise was eliminated using a bandstop filter. The bandstop filter was used to reject frequencies around 50Hz. The stopband region was set from 45Hz to 55Hz because of leakage.

Additionally, based on the echo input, the street light in the model was switched ON when the car was detected and OFF when the car was not detected. However, when implemented on the road, the street light must reduce to lower intensity rather than switching OFF. This could be implemented using PWM. PWM was not implemented in the model to ensure observability, i.e., lower intensity of LED was difficult to capture in photo/video.

The Experimental Setup

The hardware setup of the project consisted of an Arduino UNO Rev3 board with a USB cable, an HC-SR04 ultrasonic sensor, an LED and series resistor. The arduino and HR-SC04 sensor have been shown in figure 1. Jumper wires were used to connect the sensor to the Arduino ports. The connection between sensor and arduino was loose due to loose connection of male-to-female wires. This was resolved by introducing a breadboard to the circuit.

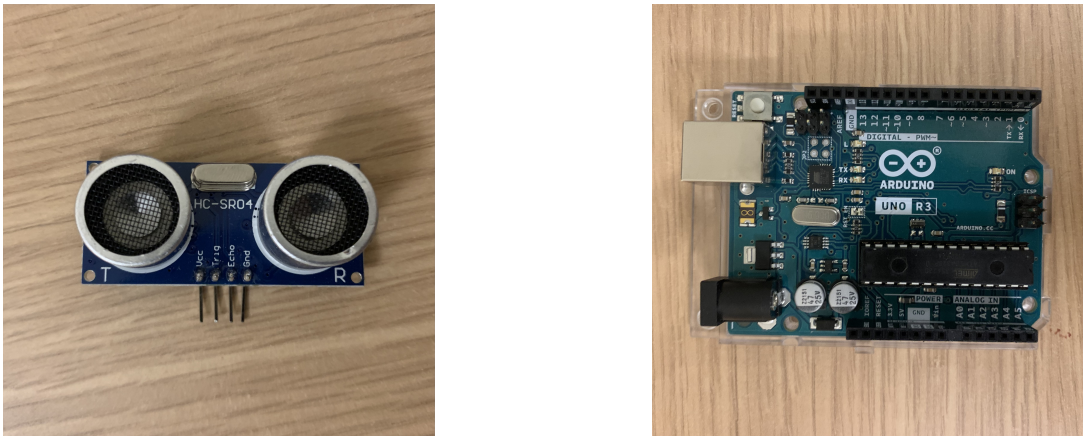

Figure 1. The Arduino UNO Rev3 board and ultrasonic sensor used for the project.

In order to filter the inputs from the sensor in real time, the Arduino board was interfaced with python using the pyFirmata2 module sourced from the author Bernd Porr from the following GitHub repository - https://github.com/berndporr/pyFirmata2

The echo pin of the sensor was connected to analog pin 0 of the Arduino, and the trigger pin was connected to analog pin 1 of the Arduino. The pins were configured in **realtime_iir_main.py** with the following lines of code.

```
# Register the callback which adds the data to the animated plot
        board.analog[0].register_callback(self.callBack1)     # Echo pin
        board.analog[1].register_callback(self.callBack2)     # Trigger pin

        # Enable the callback
        board.analog[0].enable_reporting()
        board.analog[1].enable_reporting()
```

The functions called, `callback1` and `callback2`, work to add data in real time to the plot. Additionally, these were used to calculate the actual sampling rate.

In order to test the produced circuit, a 1:100 model of the road was created. In order to do so conscientiously, average dimensions of cars, street lights, pavement, road lanes and its markings in the UK were taken and scaled down from meters to centimeters to produce this replica. This model along with the overall setup can be seen in figure 2. The same has been shown detecting the vehicle in figure 3.

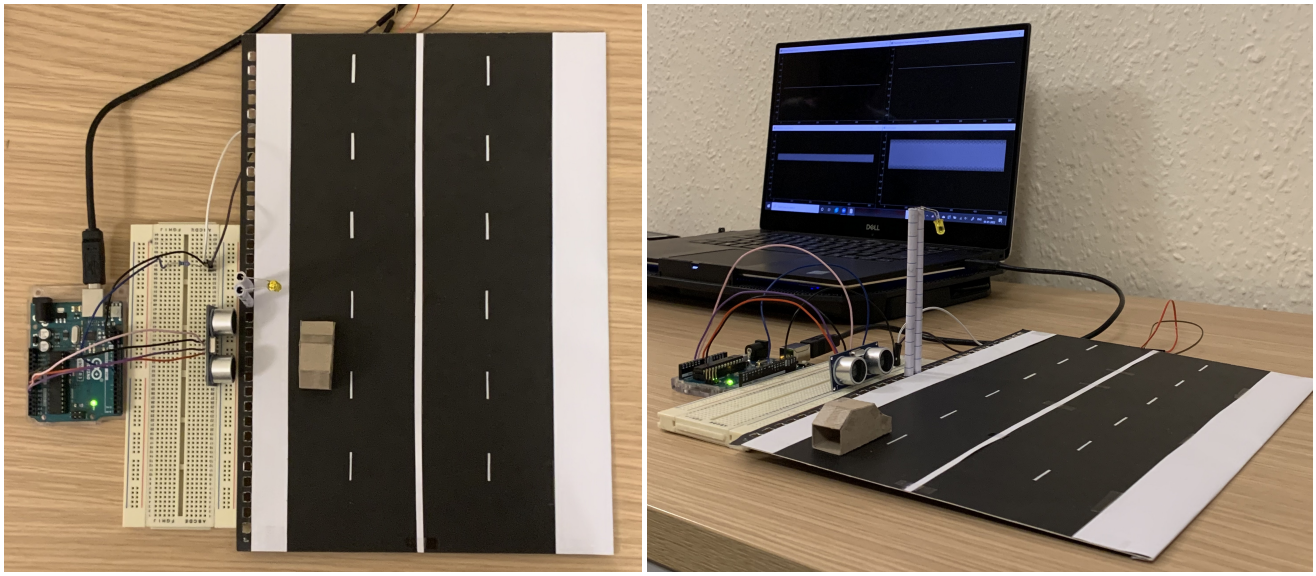

Figure 2. The overall setup of the hardware components.

Figure 3. The overall setup of the vehicle traffic light detection system in action.

The images shown in figure 3 were shot in the dark to show detection and that the LED was switched on.

YouTube Demo

A video presentation of the vehicle traffic light detection application can be found here.

Dataflow Diagrams

A dataflow diagram demonstrating the different components of the setup and the signal flow can be seen in figure 4.

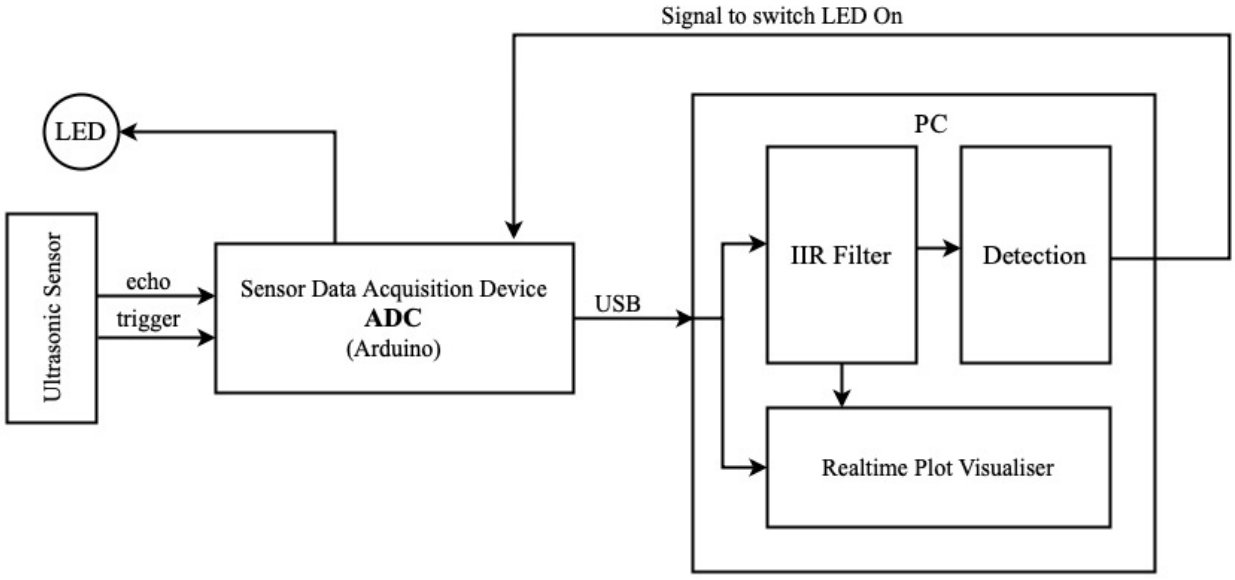

Figure 4. System Signal Block Diagram

The data-flow diagram of a 2nd order IIR filter and a chain of 2nd order filters are shown on figures 5 and 6.

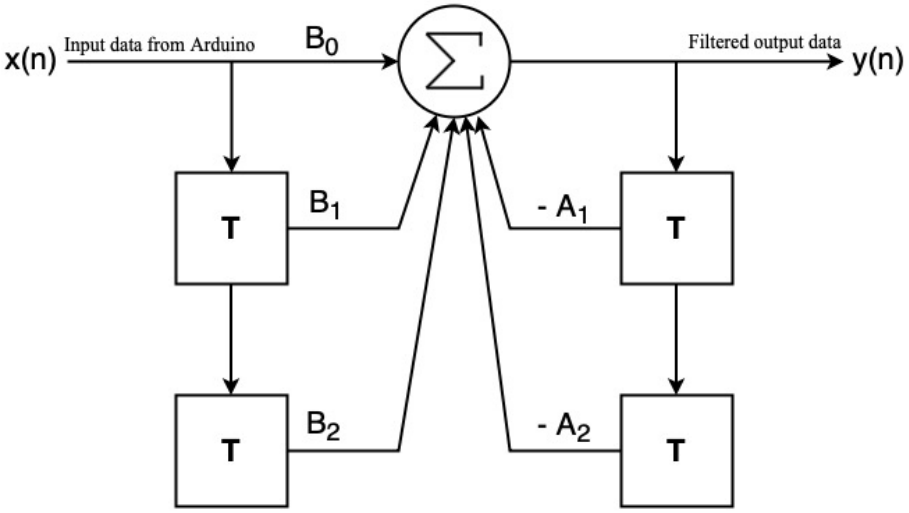

Figure 5. 2nd Order Direct Form I IIR Filter Dataflow Diagram

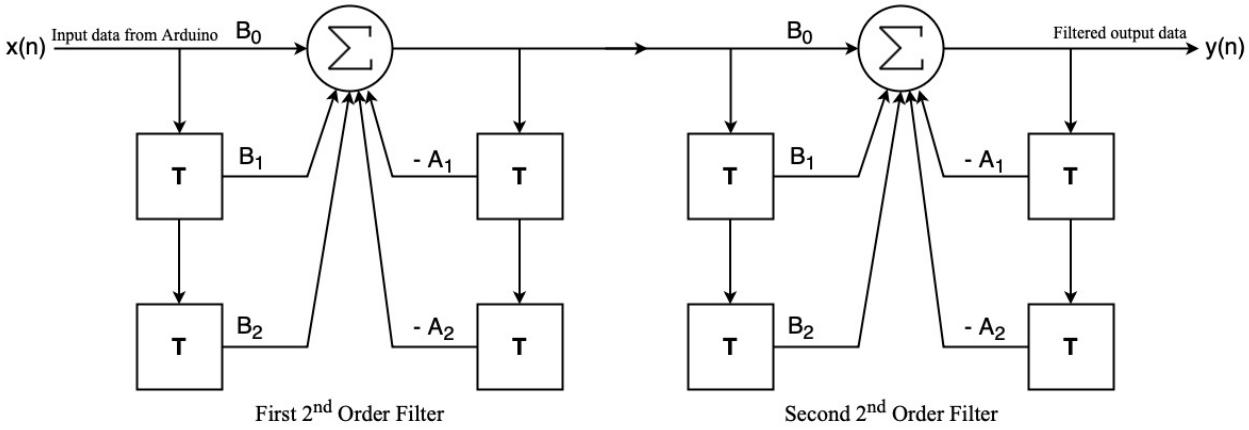

Figure 6. IIR 2nd Order Filter Chain of 2 Dataflow Diagram

Real Time Plotting

The readings have been plotted on graphs that are created by the **QtPanningPlot** class, which was provided from the example file of pyFirmata2, as this was the best approach to visualise the processing of the data in real time. The unfiltered input signal received from the arduino had an offset due to DC line interference. Additionally, the readings included unwanted high frequency peaks which were considered as noise. Hence, these components were eliminated from the input signal to obtain the filtered output. The unfiltered and the filtered response have been shown in figures 7 and 8 respectively.

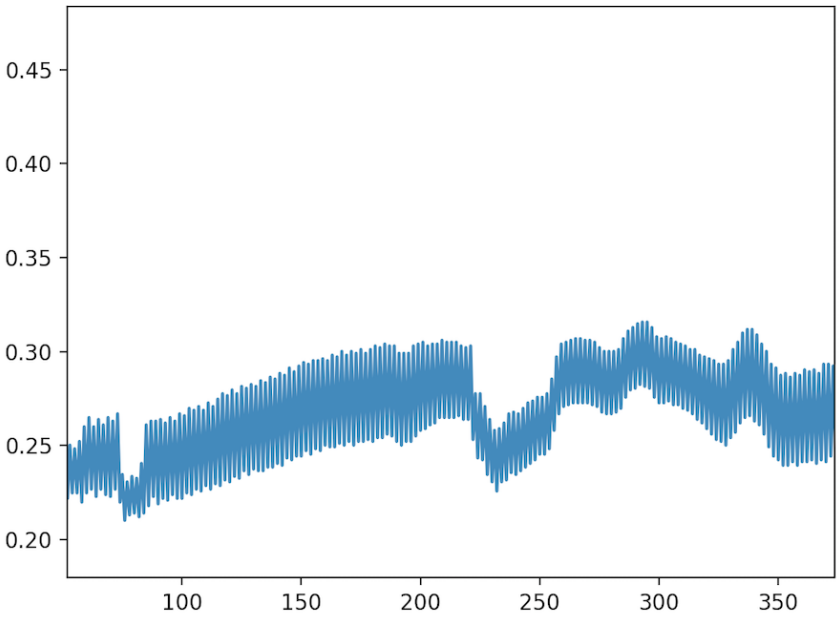

Figure 7. Readings of the sensor before filtering with an offset and noise.

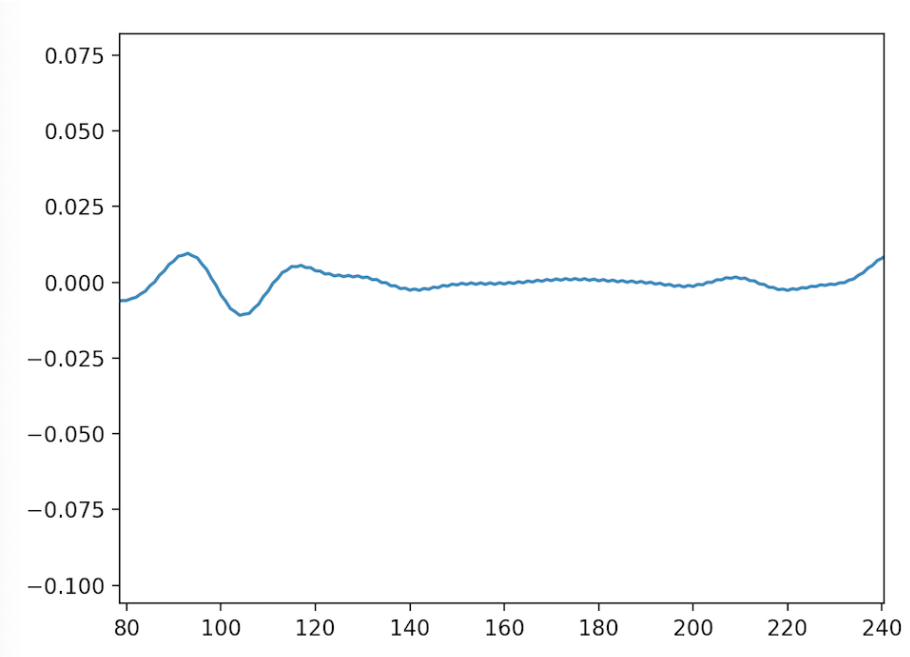

Figure 8. Readings of the sensor after filtering eliminating the offset and noise.

Generating the SOS Coefficients

To generate the IIR filter coefficients, the following python high level commands were used: butterworth and chebyshev2 from the **scipy.signal** library. Several orders, and cutoff frequencies were tested to obtain our desired response. The final filter frequency responses obtained are shown on figures 9 and 10 below.

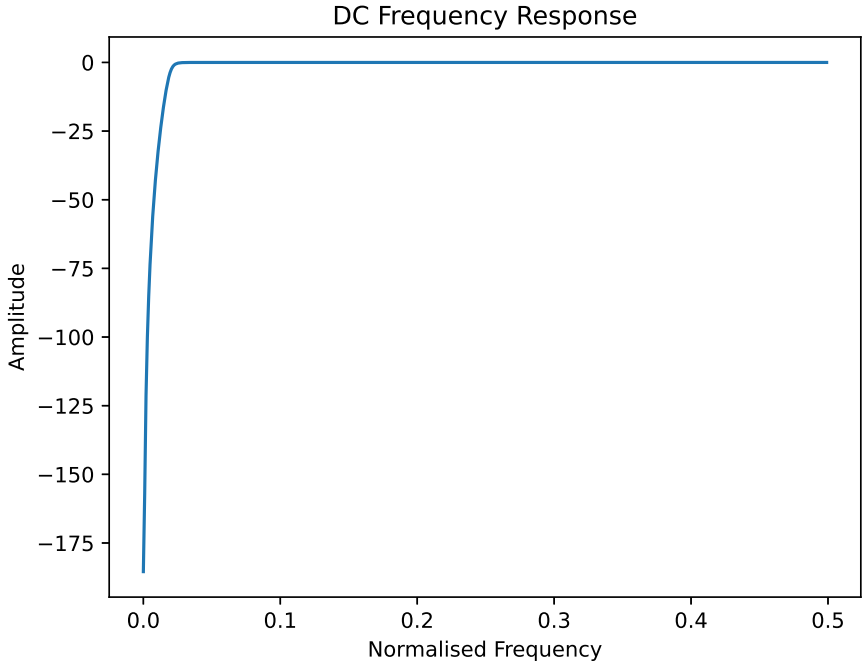

Figure 9. The frequency response of the DC Butterworth filter

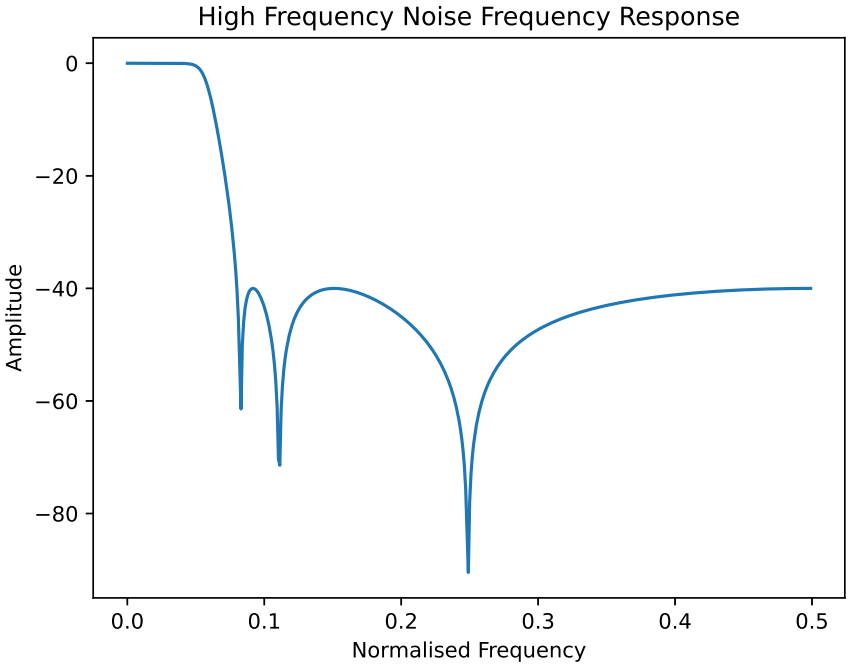

Figure 10. The frequency response of the ChebyShev2 low pass filter.

QUESTION 2

The unit test served to ensure that the filter worked as needed. An array was defined for which the filter output could be calculated, and the filter was applied. Afterwards, the filter output was calculated by hand. The two outputs were compared and if they matched, it was confirmed that the filter was working. If they did not match, it was inferred that the filter did not do what it was supposed to, assuming that there was no calculation error.

The handwritten calculation for the unit test with the input $x(n) = [6, -1, 4]$ is below:

Unit Test for DC filter

Filter 1: $b_{10} = 0.78429785$ $a_{10} = 1$
$b_{11} = -1.56859571$ $a_{11} = -1.76995414$
$b_{12} = 0.78429785$ $a_{12} = 0.7840216$

Filter 2: $b_{20} = 1$ $a_{20} = 1$
$b_{21} = -2$ $a_{21} = -1.82269493$
$b_{22} = 1$ $a_{22} = 0.83718165$

Filter 3: $b_{30} = 1$ $a_{30} = 1$
$b_{31} = -2$ $a_{31} = -1.92188606$
$b_{32} = 1$ $a_{32} = 0.93716115$

$x(n) = [6, -1, 4]$

Filter the signal one-by-one using:

$a_0 y(n) = b_0 x(n) + b_1 x(n-1) + b_2 x(n-2) - a_1 y(n-1) - a_2 y(n-2)$

$y_1(0) = b_{10} x(0) = (0.78429785)(6) = 4.70578871$
$y_2(0) = b_{20} y_1(0) = 4.70578871$
$y(0) = b_{30} y_2(0) = 4.70578871$

$y_1(1) = b_{10} x(1) + b_{11} x(0) - a_{11} y_1(0) = (0.78429785)(-1) + (-1.56859571)(6) - (-1.76995414)(4.70578871) = -1.86684475$
$y_2(1) = b_{20} y_1(1) + b_{21} y_1(0) - a_{21} y_2(0) = -1.86684475 + (-2)(4.70578871) - (-1.82269493)(4.70578871) = -2.70120466 2$
$y(1) = b_{30} y_2(1) + b_{31} y_2(0) - a_{31} y(0) = -2.70120466 2 + (-2)(4.70578871) - (-1.92188606)(4.70578871) = -3.06879223 3$

$y_1(2) = b_{10} x(2) + b_{11} x(1) + b_{12} x(0) - a_{11} y_1(1) - a_{12} y_1(0) = (0.78429785)(4) + (-1.56859571)(-1) + (0.78429785)(6) - (-1.76995414)(-1.86684475) - (0.7840216)(4.70578871) = 2.41790$
$y_2(2) = b_{20} y_1(2) + b_{21} y_1(1) + b_{22} y_1(0) - a_{21} y_2(1) - a_{22} y_2(0) = 2.41790 + (-2)(-1.86684475) + 4.70578871 - (-1.82269493)(-2.70120466 2) - (0.83718165)(4.70578871) = 1.99430599 9$
$y_3(2) = b_{30} y_2(2) + b_{31} y_2(1) + b_{32} y_2(0) - a_{31} y(1) - a_{32} y(0) = 1.99430599 9 + (-2)(-2.70120466 2) + 4.70578871 - (-1.92188606)(-3.06879223 3) - (0.93716115)(4.70578871) = 1.79455$

The results highlighted in blue are for the 2nd order filter, and the results highlighted in pink are for the chain of 2nd order filters.

The calculation went through all the filters one by one to obtain the result. The first filter's input was x, and the rest of the filters' inputs were the output of the previous filter.

The unit test function was added to the `IIR_filter` and `IIR2_filter` classes. There was a separate program that ran the unit test, and printed the result.

For the unit test, a 6th order Butterworth filter that removed the DC component was used. This filter was also used in the filtering of our signal. The 6th order filter was created with 3 2nd order filters. The 2nd order filter just took the first filter in the chain of filters.

The output obtained from the unit test program has been shown below:

```
(base) zeynepdeniz:assignment 3 Zeynep$ python -u "/Users/Zeynep/Desktop/cse/dsp/assignment 3/rununittest.py"
DC coefficients: [[ 0.78429785 -1.56859571  0.78429785  1.         -1.76995414  0.78402168]
 [ 1.         -2.          1.          1.         -1.82269493  0.83718165]
 [ 1.         -2.          1.          1.         -1.92188606  0.93716115]]
6th order filter coefficients: [ 4.70578712 -3.06879223  1.79455803]
2nd order filter coefficients: [ 4.70578712 -1.8668447   2.41790559]
```

The achieved output exactly matched with the calculation we did by hand. Thus, it was concluded that the test was a success, and the filters worked as required.

QUESTION 3

The system was setup to collect data samples from the analogue ports of the arduino with a frequency of 100 Hz. The code to verify this has been shown below and was included in `realtime_iir_main.py` inside the `callbacks` class. It printed the instantaneous sampling rate whilst the application was running. To do this, the `time()` function was used from the `time` library to read the exact time at which the reporting from the analogue pins were enabled. This time was defined as the `start_time`. When the first callback was called, the time was read again, and the `start_time` was subtracted from this in order to get the exact time that the callback was called. `delta_t` represented the execution time. A `timestamp` was used to keep a count of the number of samples that arrived.

```
    self.current_time = time.time() - self.start_time
    self.t2 = time.time()                # t2 is start of execution time
    self.delta_t = self.t2 - self.t1     # execution time
    if self.timestamp != 0:
        self.current_sr = self.current_time / (call_backs.timestamp / call_backs.
    samplingRate)
    print('The current sampling rate', self.current_sr)
    self.t1 = self.t2                    # t1 is the end of execution time
```

As shown in figure 11, the instantaneous sampling rate was printed in the terminal.

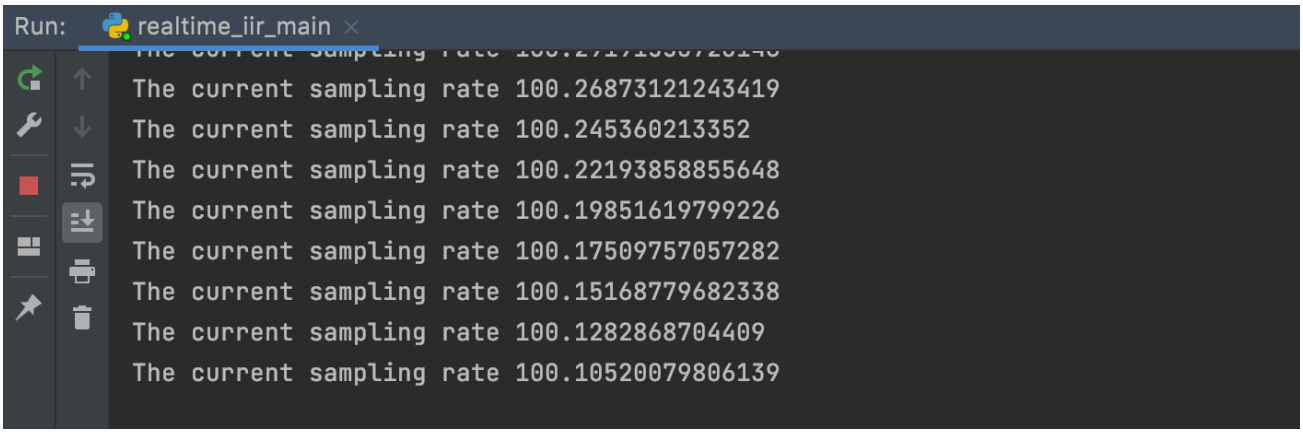

Figure 11. Terminal output printing the current sampling rate in realtime.

The results from the terminal output on Figure 11 further confirms that the sampling rate was roughly 100 Hz throughout. Further, as it was required to add this data to a plot, the instantaneous sampling rate was normalized against the set sampling rate, a new plot was created and the normalized sampling rate was plotted.

QUESTION 4

One of the challenges faced during this assignment was the data acquisition. Since electronics can be unpredictable at times, we got different results from the same code in different times of the day. It is suspected that this was caused due to loose connections and temperature difference at different times of the day. As aforementioned, a breadboard was included in the circuit to get tight connections.

Additionally, the location of our experiment changed the results. While intensity of 50 Hz noise was low in student accommodations, it was found the University library had a very large intensity of 50 Hz noise coming from the outlet. This noise in the library gave us the opportunity to clean the output. It was the best noise to be filtered and the library was the best place to conduct the experiment.

The real time signal was designed to filter the noise out of the trigger signal. The trig signal had a 50Hz noise coming from the mains. To remove this noise, an IIR filter was implemented to the trig signal. The result was a clear signal where the values were easily readable. Filtering as well as the road model were successful, as seen on the YouTube video.

Furthermore, the replica-based testing of this real time code was also successful. However, a challenge in real life implementation could be the network type communication. Controlling a series of street lights based on the signal received in one sensor could be complex and could potentially introduce delays. To tackle this challenge, a star type network is proposed.

Additionally, the range of ultrasonic sensor in air, for real life implementation was found to be 11 metres, so it may be ideal for roads which have up to 3 lanes, because lane-width in the UK was found to be 3.65 m. Furthermore, the sensitivity the of ultrasonic sensor would be efficient for wind speeds of up to 62 kmph, thus it could be implemented on roads with a speed limit of 90-100 kmph.

APPENDIX A. CODES

realtime_iir_main.py

```
import sys
import numpy as np
import pyqtgraph as pg
from pyqtgraph.Qt import QtCore, QtGui
from iirfilter import IIR_filter
import scipy.signal as sig
from pyfirmata2 import Arduino
import time

PORT = Arduino.AUTODETECT

# create a global QT application object
app = QtGui.QApplication(sys.argv)

# signals to all threads in endless loops that we'd like to run these
running = True


class QtPanningPlot:

    def __init__(self, title):
        self.win = pg.GraphicsLayoutWidget()
        self.win.setWindowTitle(title)
        self.plt = self.win.addPlot()
        self.plt.setYRange(-1, 1)
        self.plt.setXRange(0, 500)
        self.curve = self.plt.plot()
        self.data = []
        self.timer = QtCore.QTimer()
        self.timer.timeout.connect(self.update)
        self.timer.start(100)
        self.layout = QtGui.QGridLayout()
        self.win.setLayout(self.layout)
        self.win.show()

    def update(self):
        self.data = self.data[-500:]
        if self.data:
            self.curve.setData(np.hstack(self.data))

    def addData(self, d):
        self.data.append(d)


# Let's create four instances of plot windows
qtPanningPlot1 = QtPanningPlot("Echo Signal")
qtPanningPlot2 = QtPanningPlot("Unfiltered Trigger Signal")
qtPanningPlot3 = QtPanningPlot("Filtered Trigger Signal")
qtPanningPlot4 = QtPanningPlot("Normalized Instantaneous Sampling Rate")

# sampling rate
samplingRate = 100

# DC line filter to eliminate the offset
```

```
fc = 2  # cutoff frequency
sosDC = sig.butter(6, fc / samplingRate * 2, btype='high', output='sos')

# elimiating the high frequency noise
fc = 8  # cutoff frequency
sosLP = sig.cheby2(6, 40, fc / samplingRate * 2, btype='low', output='sos')

# instantiate the 2nd order chain class
filterDC = IIR_filter(sosDC)
filterLP = IIR_filter(sosLP)


# called for every new sample which has arrived from the Arduino
class callbacks:
    def __init__(self):
        self.samplingRate = 100
        self.timestamp = 0
        self.start_time = 0
        self.current_time = 0
        self.current_sr = 0
        self.delta_t = 0
        self.t2 = 0
        self.t1 = 0
        self.previous = np.zeros(5)

    def initial(self, board):
        # Set the sampling rate in the Arduino
        board.samplingOn(1000 / self.samplingRate)

        # Register the callback which adds the data to the animatedplot
        board.analog[0].register_callback(self.callBack1)
        board.analog[1].register_callback(self.callBack2)

        # Enable the callback
        board.analog[0].enable_reporting()
        board.analog[1].enable_reporting()
        self.start_time = time.time()

    def callBack1(self, data):
        # send the sample to the plot window
        # filtering the data

        self.current_time = time.time() - self.start_time
        self.t2 = time.time()               # t2 is the start time of the execution
    time
        self.delta_t = self.t2 - self.t1    # calculation of the execution time
        if self.timestamp != 0:
            self.current_sr = self.current_time / (call_backs.timestamp / call_backs.
    samplingRate)
        print('The instantaneous sampling rate', self.current_sr)
        self.t1 = self.t2                   # t1 is the end time of the execution time

        # Detection
        for i in range(len(self.previous)):
            self.previous[len(self.previous) - i - 1] = self.previous[len(self.
    previous) - i - 2]
            self.previous[0] = data
```

```
109
110         if np.sum(self.previous) < 0.001:
111             board.digital[8].write(True)
112         else:
113             board.digital[8].write(False)
114
115         qtPanningPlot4.addData(self.current_sr / self.samplingRate)
116         qtPanningPlot1.addData(data)
117
118         # keep a count of the number of samples arriving
119         self.timestamp += 1 / self.samplingRate
120
121     def callBack2(self, data):
122         # send the sample to the plot window
123         # filtering the data
124
125         output = filterDC.filter(data)
126         output = filterLP.filter(output)
127
128         # Plot unfiltered input data and filtered output
129         qtPanningPlot2.addData(data)
130         qtPanningPlot3.addData(output)
131
132
133 # Get the Arduino board.
134 board = Arduino(PORT)
135
136 # Declare callbacks class as call_backs
137 call_backs = callbacks()
138 # Start the Initial function from the class initial which run the true callback inside
        the class
139 call_backs.initial(board)
140
141 # showing all the windows
142 app.exec_()
143 finish_time = time.time()
144
145 # needs to be called to close the serial port
146 board.exit()
147
148 Execution_time = finish_time - call_backs.start_time
149 Actual_sampling_rate = Execution_time / (call_backs.timestamp / call_backs.
        samplingRate)
150
151 print('Execution Time:', Execution_time)
152 print('Actual Sampling Rate:', Actual_sampling_rate)
153 print("finished")
```

iirfilter.py

```
1 #
2 # (C) 2020 Bernd Porr, mail@berndporr.me.uk
3 # Apache 2.0 license
4 #
5 import scipy.signal as signal
6 import numpy as np
7
```

```
8
9  class IIR2_filter:
10     """2nd order IIR filter"""
11
12     def __init__(self, s):
13         """Instantiates a 2nd order IIR filter
14         s -- numerator and denominator coefficients
15         """
16         self.numerator0 = s[0]
17         self.numerator1 = s[1]
18         self.numerator2 = s[2]
19         self.denominator1 = s[4]
20         self.denominator2 = s[5]
21         self.buffer1 = 0
22         self.buffer2 = 0
23
24     def filter(self, v):
25         """Sample by sample filtering
26         v -- scalar sample
27         returns filtered sample
28         """
29         input = v - (self.denominator1 * self.buffer1) - \
30             (self.denominator2 * self.buffer2)
31         output = (self.numerator1 * self.buffer1) + \
32             (self.numerator2 * self.buffer2) + input * self.numerator0
33         self.buffer2 = self.buffer1
34         self.buffer1 = input
35         return output
36
37     def unittest(self):
38         """
39         The unit test is done by getting a random array, then filtering the
40         array. The filtering is then done by hand and the two results are
41         compared with each other to see if they match.
42         """
43         x = [6, -1, 4]  # random input to test our filter
44         y = np.zeros(len(x))
45         for i in range(len(x)):
46                 y[i] = self.filter(x[i]) # filter the output one sample at a time
47
48         return y
49
50 class IIR_filter:
51     """IIR filter"""
52
53     def __init__(self, sos):
54         """Instantiates an IIR filter of any order
55         sos -- array of 2nd order IIR filter coefficients
56         """
57         self.cascade = []
58         for s in sos:
59             self.cascade.append(IIR2_filter(s))
60
61     def filter(self, v):
62         """Sample by sample filtering
63         v -- scalar sample
64         returns filtered sample
```

```
        """
        for f in self.cascade:
            v = f.filter(v)
        return v

    def unittest(self):
        """
        The unit test is done by getting a random array, then filtering the
        array. The filtering is then done by hand and the two results are
        compared with each other to see if they match.
        """
        x = [6, -1, 4]  # random input to test our filter
        y = np.zeros(len(x))
        for i in range(len(x)):
                y[i] = self.filter(x[i]) # filter the output one sample at a time

        return y
```

REFERENCES

[1] TANNOUS, S., MANNEH, R., HARAJLI, H. & EL ZAKHEM, H. 2018. Comparative cradle-to-grave life cycle assessment of traditional grid-connected and solar stand-alone street light systems: A case study for rural areas in Lebanon. Journal of Cleaner Production, 186, 963-977.

[2] JUNTUNEN, E., SARJANOJA, E.-M., ESKELI, J., PIHLAJANIEMI, H. & ÖSTERLUND, T. 2018. Smart and dynamic route lighting control based on movement tracking. Building and Environment, 142, 472-483.

[3] AGC Lighting. (n.d.). 7 Things You Should Know About LED Street Light with Photocell. [online] Available at: https://www.agcled.com/blog/7-things-you-should-know-about-led-street-light-with-photocell.html.

[4] www.twi-global.com. (n.d.). What is Sustainability and why is it so Important? [online] Available at: https://www.twi-global.com/technical-knowledge/faqs/faq-what-is-sustainability.