Assignment 3

Digital Signal Processing IIR Filters

Group 65 – Proximity Sensor

Rohit Avadhani (2696886A)

Varsha Shiburaj(2664373S)

Léon Rémusat (2692920R)

Question 1:

Below are attached photos of the setup and dataflow diagrams. The camera used during the real-time processing YouTube video is a MacBook Pro mid-2012 720p FaceTime HD camera [1]. Additional research on the technical specifications of that camera showed that the framerate was 30 frame per second (fps) [2].

The YouTube clip can be found at: https://www.youtube.com/watch?v=o3QWh1pppyk

1.1. Photos of the setup.

The setup consists of a simple integrated MacBook Pro camera for which the printed circuits are displayed below.

Figure 1 Macbook Pro 2012 Camera

1.2. Dataflow diagrams

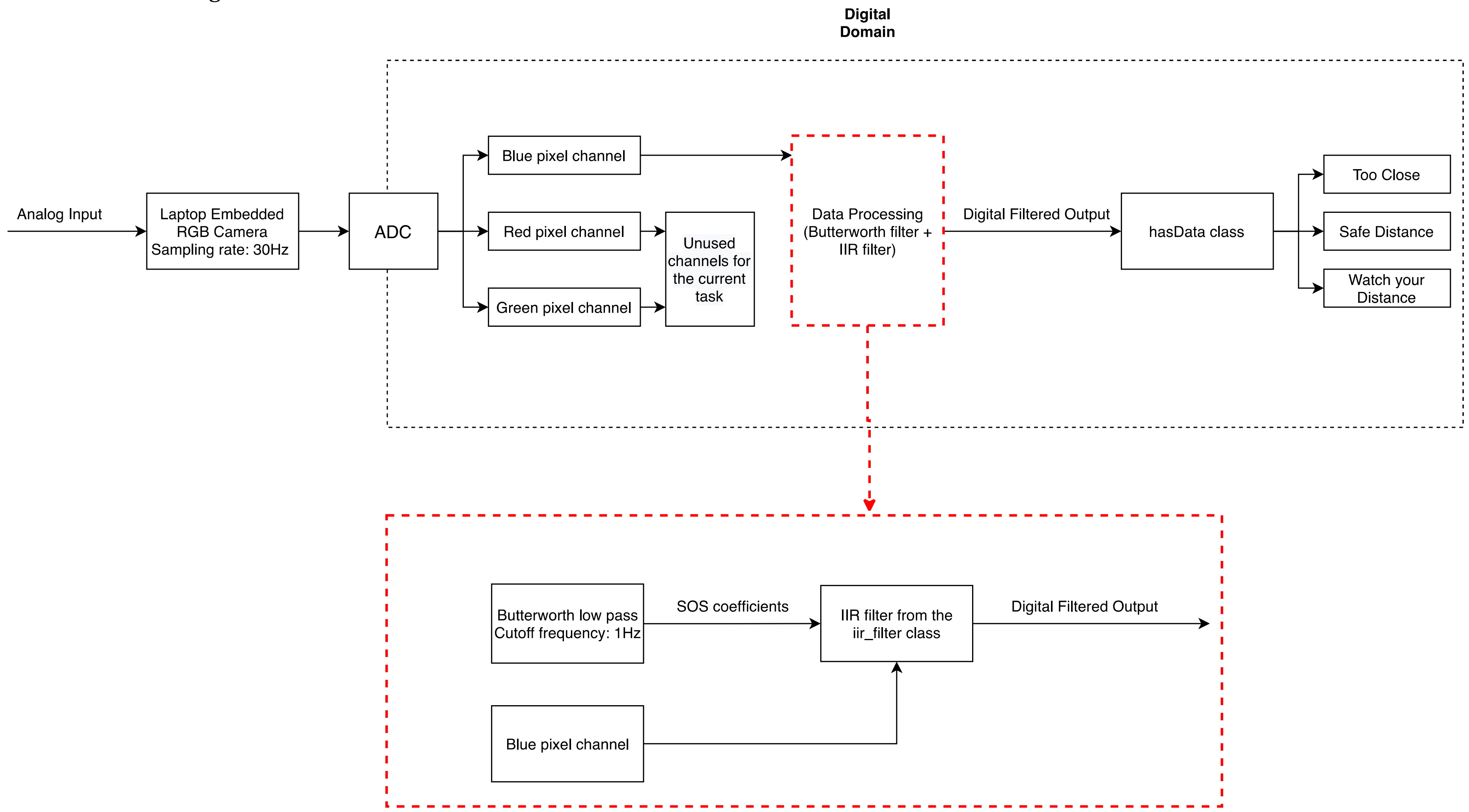


Figure 2 Dataflow diagram

1.3.

To find the type of filter needed using scipy.signal.butter, the RGB camera output was recorded for 30 seconds to have sufficient data points and stored into a .dat file (using demo.py with modified lines to allow data storing). Applying the fast Fourier transform (FFT) command, the spectrum of raw data could be displayed and thus analyzed to observe the presence of noise (from sensors, etc.).

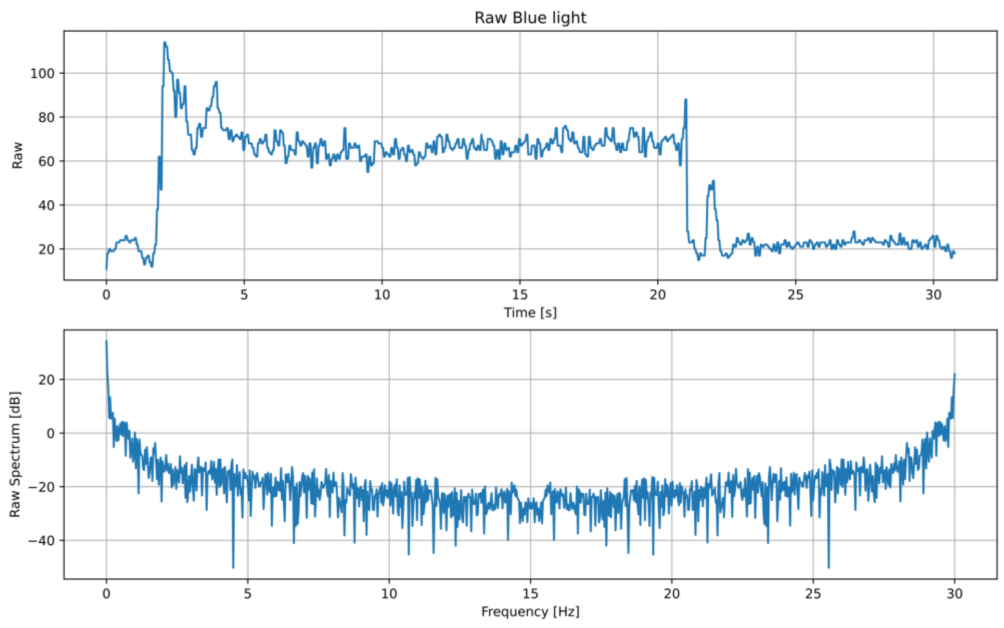


Figure 3 Raw data against time (top) and frequency spectrum (bottom)

From the top subplot in Figure 2, we observe a high frequency noise. This is confirmed by the frequency spectrum plot displaying peaks from 0 Hz to 15 Hz (at M/2). Hence a low pass filter is needed to remove the peaks shown on the bottom subplot in Figure 2. The cut-off frequency f1 can be found iteratively by comparing the raw data plot (time domain) to the filtered data (time domain). The aim is to remove high frequency noise but without removing the necessary part of the signal. First, cutoff frequency needs to be normalized to Nyquist to use the digital Butterworth filter and SOS coefficients [3]. The coefficients are then generated using the following command (with fs: sampling rate, f1: cutoff frequency):

```
sos1 = signal.butter(4, f1/fs*2, 'lowpass', output='sos')
```

The filter response is shown below (see next page).

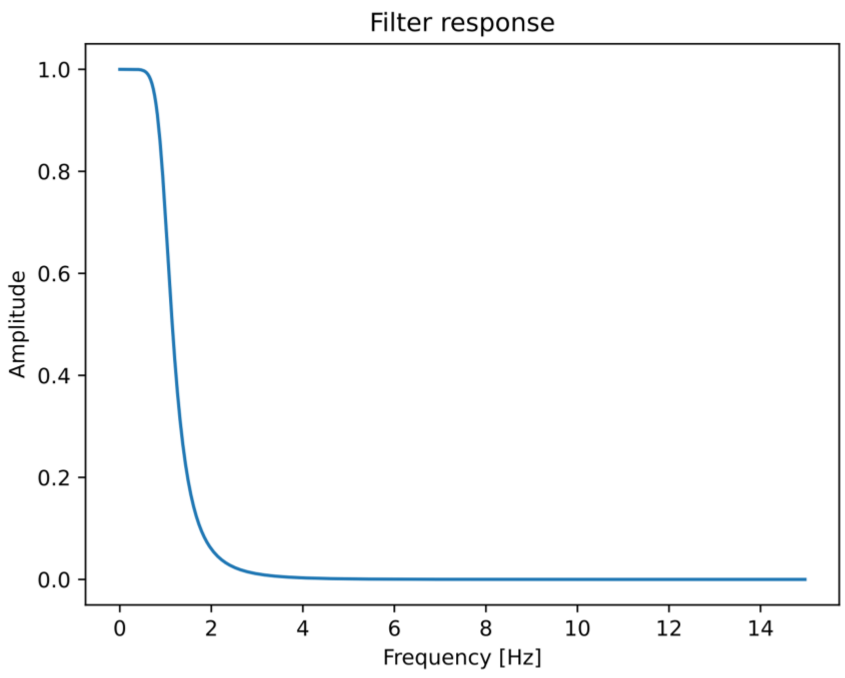


Figure 4 Filter response

The resulting cutoff frequency was set to fc= 1 Hz. Below is shown the filtered data both in the time domain and the frequency domain.

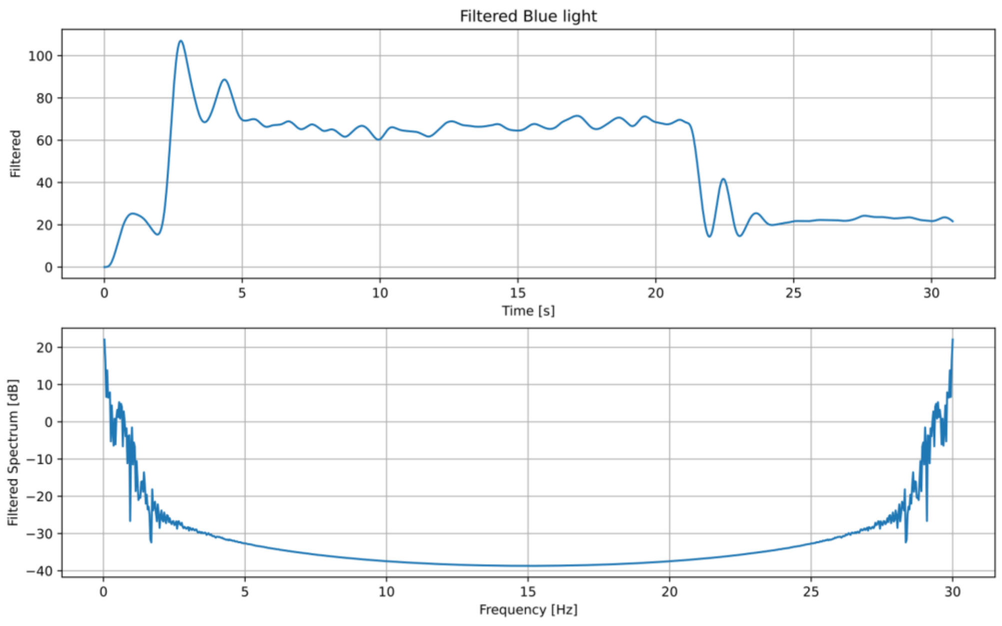


Figure 5 Filtered data against time (top) and frequency spectrum (bottom)

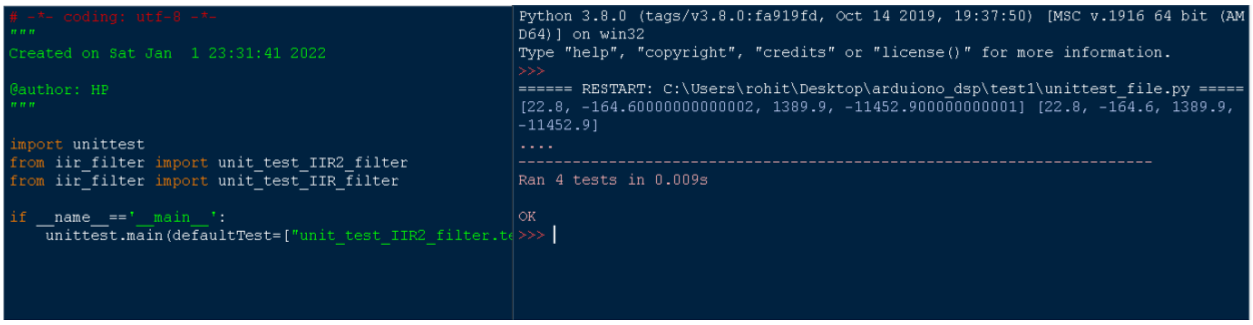
By comparing Figure 3 to Figure 2, we observe a much cleaner blue light signal without any ripples. This is translated into the frequency domain by the removal of high frequency peaks.

Question 2:

Real-time filtering is done by considering the SOS coefficients in real-time. We consider a sampling frequency of 30Hz and a cut-off frequency for our low pass filter in signal.butter at 1 Hz and the order of the filter is set at 4. A further explanation is provided in question 4 in terms of critical analysis on real-time filtering as well as in the YouTube clip.

For the unit test, we are taking two test cases for 2nd order filter and the chain of 2nd order filters. To make sure the values obtained from output1_test and output1 (for example) are as close to each other, we set a threshold of 1e-4. If the difference between the values is less than 1e-4, the filters pass the unit test. The same principle works for the rest of the cases. The idea is to make sure the threshold difference is maintained.

The unit test is run from a different file, rununittest.py. The unit test was successful (returned dot(.) for each test case).



```
# -*- coding: utf-8 -*-
"""
Created on Sat Jan  1 23:31:41 2022

@author: HP
"""

import unittest
from iir_filter import unit_test_IIR2_filter
from iir_filter import unit_test_IIR_filter

if __name__=='__main__':
    unittest.main(defaultTest=["unit_test_IIR2_filter.te
```

```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:37:50) [MSC v.1916 64 bit (AM
D64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
====== RESTART: C:\Users\rohit\Desktop\arduiono_dsp\test1\unittest_file.py =====
[22.8, -164.60000000000002, 1389.9, -11452.900000000001] [22.8, -164.6, 1389.9,
-11452.9]
....
----------------------------------------------------------------------
Ran 4 tests in 0.009s

OK
>>> |
```

Figure 6

Question 3:

The real-time sampling rate of the acquisition has been plotted and a snippet of the plot when it was run real-time is given below as an example. It can be noted that the values are around 30 calls per second. When tested on different devices, deviations from this value were observed which might be due to the differences in hardware.

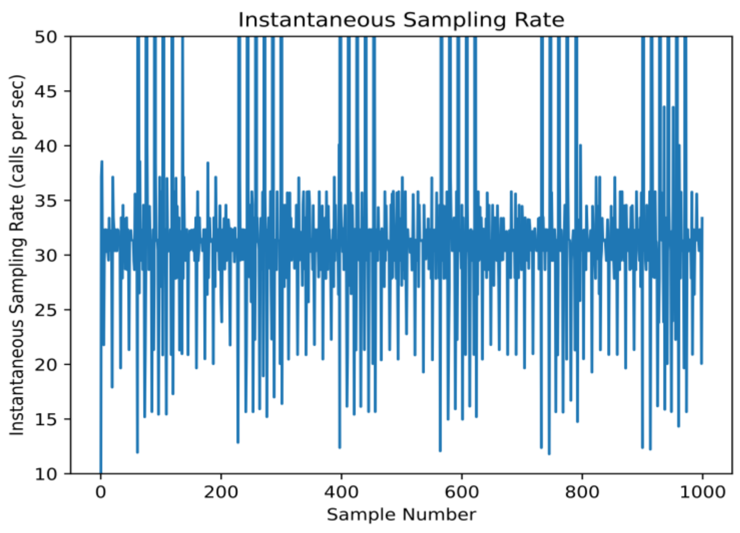


Figure 7

Question 4:

To compare the filtered signal with the original one, it is crucial to first understand how an RGB camera works. On the three channels of the camera, each signal will get perturbed when its color appears in front of the webcam. The color sensitivity is coded on 8 bits with a maximum number of 256 steps. Having three channels, it means a total of 256*256*256=16777216 colors can be created.

The current experiment was carried out in a room with white walls and wearing a blue shirt. The proximity sensor returns three messages which correspond to the distance from the user to the paint in a museum (assuming the camera is fixed on the paint's wall). The blue channel has been used in this experiment so the blue level will vary depending on how far the user is staying from the webcam. When the user is too close, the blue level ranges between 60 – 100. When the user is in a region which is neither safe nor too close, the blue level is below 25. Anywhere else the user is staying at a safe distance. These thresholds were found iteratively, it is highly likely that they cannot be exactly reproduced in a new environment (different colors of the walls or clothes, lightening, etc.)

Now comparing the real-time raw signal to the filtered on (see Figure X), we observe that noise has been significantly reduced (less ripples) and therefore it makes it easier to set a threshold to for the proximity sensor. In particular, the gap between "Too Close" and "Watch your distance" is very narrow. The noisy peaks make this classification impossible because the overshoots are larger than the classification range.

However, there remain some flaws of using this approach. It has been found that a blue of level of 40 can be reached either at a safe distance, or when moving from "Watch your distance" to "Too Close". The system does not memorize from which state the user is coming from and therefore at that particular blue level it remains impossible to say if this is a safe zone or if the user is too close. One way of tackling this problem would be to use cameras placed at different locations to determine the user's position and additionally combine the 3 RGB channels.

Bibliography

[1] Apple Inc., "MacBook Pro (13-inch, Mid 2012) - Technical Specifications," 14 04 2021. [Online]. Available: https://support.apple.com/kb/sp649?locale=en_GB. [Accessed 12 2021].

[2] StackExchange, "What is the framerate of the macbook pro mid 2012 non retina facetime hd camera?," StackExchange, 03 2015. [Online]. Available: https://apple.stackexchange.com/questions/174993/what-is-the-framerate-of-the-macbook-pro-mid-2012-non-retina-facetime-hd-camera. [Accessed 12 2021].

[3] The SciPy community., scipy.signal.butter , 05 2021. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.butter.html. [Accessed 12 2021].

Appendix

realtime_iir_main.py

```
#!/usr/bin/python3

import numpy as np

import matplotlib.pyplot as plt
import matplotlib.animation as animation

import webcam2rgb
import iir_filter
from scipy import signal
import time
#import csv


class RealtimePlotWindow:

    def __init__(self, channel: str, set_lim=None):
        # create a plot window
        self.fig, self.ax = plt.subplots()
        plt.title(f"Channel: {channel}")
        # that's our plotbuffer
        self.plotbuffer = np.zeros(500)
        # create an empty line
        self.line, = self.ax.plot(self.plotbuffer)
        self.set_lim = set_lim
        # axis
        if self.set_lim is None:
            self.ax.set_ylim(0, 1)
        else:
            self.ax.set_ylim(self.set_lim)
        # That's our ringbuffer which accumluates the samples
        # It's emptied every time when the plot window below
        # does a repaint
        self.ringbuffer = []
        # add any initialisation code here (filters etc)
        # start the animation
        self.ani = animation.FuncAnimation(self.fig, self.update,
interval=100)
        #self.filename = open("results.csv","w")
        #self.csvwriter = csv.writer(self.filename)


    # updates the plot
    def update(self, data):
        # add new data to the buffer
        self.plotbuffer = np.append(self.plotbuffer, self.ringbuffer)
        # only keep the 500 newest ones and discard the old ones
        self.plotbuffer = self.plotbuffer[-500:]
        self.ringbuffer = []
        # set the new 500 points of channel 9
        self.line.set_ydata(self.plotbuffer)
        if self.set_lim is None:
            self.ax.set_ylim(0, max(self.plotbuffer)+1)
        else:
            self.ax.set_ylim(self.set_lim)
        return self.line,
```

```python
    # appends data to the ringbuffer
    def addData(self, v):
        self.ringbuffer.append(v)


realtimePlotWindowBlue = RealtimePlotWindow("Blue Channel - Raw")
realtimePlotWindowBFil= RealtimePlotWindow("Blue Channel - Filtered")
realtimePlotWindowSampRate = RealtimePlotWindow("Sampling Rate of
Acquisition", (0, 100))
#realtimePlotWindowRed = RealtimePlotWindow("Red")


fs = 30
duration = 1/fs
f1 = 1
sos1 = signal.butter(4, f1/fs*2, 'lowpass', output='sos')


iir1 = iir_filter.IIR_filter(sos1)

#create callback method reading camera and plotting in windows
s = time.time()
#avg, count = 0, 0

def hasData(retval, data):
    global s
    #global avg, count
    e = time.time()
    te = e-s
    #avg *= count
    #avg += 1/te
    #count += 1
    #avg /= count
    # print(np.round(1/te, 3), np.round(avg, 2))
    s = e

    b = data[0]
    #g = data[1]
    #r = data[2]
    realtimePlotWindowBlue.addData(b)
    y = iir1.filter(b)
    realtimePlotWindowBFil.addData(y)
    realtimePlotWindowSampRate.addData(1/te)
    #print("Sampling rate of acquisition",1/te)

    if (y <25):
        print ('Watch your distance     ')
    elif (40< y <130):
        print('Too close     ')
    else:
        print('Safe distance     ')

#create instances of camera
camera = webcam2rgb.Webcam2rgb()
#start the thread and stop it when we close the plot windows
camera.start(callback = hasData, cameraNumber=0)
print("Camera Sampling Rate(Webcam2RGB).....", camera.cameraFs(), "Hz")
plt.show()

camera.stop()
print('finished')
```

runun ittest.py

```
# -*- coding: utf-8 -*-
"""
Created on Sat Jan  1 23:31:41 2022

@author: Group 65
"""

import unittest
from iir_filter import unit_test_IIR2_filter
from iir_filter import unit_test_IIR_filter

if __name__=='__main__':

unittest.main(defaultTest=["unit_test_IIR2_filter.test1","unit_test_IIR2_fi
lter.test2","unit_test_IIR_filter.test3","unit_test_IIR_filter.test4"])
```

iir filter.py

```
#
# (C) 2020 Bernd Porr, mail@berndporr.me.uk
# Apache 2.0 license
#
import numpy as np

class IIR2_filter:
    """2nd order IIR filter"""

    def __init__(self,s):
        """Instantiates a 2nd order IIR filter
        s -- numerator and denominator coefficients
        """
        self.numerator0 = s[0]
        self.numerator1 = s[1]
        self.numerator2 = s[2]
        self.denominator1 = s[4]
        self.denominator2 = s[5]
        self.buffer1 = 0
        self.buffer2 = 0

    def filter(self,v):
        """Sample by sample filtering
        v -- scalar sample
        returns filtered sample
        """
        input = v - (self.denominator1 * self.buffer1) - (self.denominator2
* self.buffer2)
        output = (self.numerator1 * self.buffer1) + (self.numerator2 *
self.buffer2) + input * self.numerator0
        self.buffer2 = self.buffer1
        self.buffer1 = input
        return output

class IIR_filter:
    """IIR filter"""
    def __init__(self,sos):
        """Instantiates an IIR filter of any order
        sos -- array of 2nd order IIR filter coefficients
        """
        self.cascade = []
        for s in sos:
            self.cascade.append(IIR2_filter(s))

    def filter(self,v):
        """Sample by sample filtering
        v -- scalar sample
        returns filtered sample
        """
        for f in self.cascade:
            v = f.filter(v)
        return v
```

webcam2rgb.py

```
import cv2
import numpy as np

import threading



class Webcam2rgb():

    def start(self, callback, cameraNumber=0, width = None, height = None,
fps = None, directShow = False):
        self.callback = callback
        try:
            self.cam = cv2.VideoCapture(cameraNumber + (cv2.CAP_DSHOW if
directShow else cv2.CAP_ANY))
            if not self.cam.isOpened():
                print('opening camera')
                self.cam.open(0)

            if width:
                self.cam.set(cv2.CAP_PROP_FRAME_WIDTH,width)
            if height:
                self.cam.set(cv2.CAP_PROP_FRAME_HEIGHT,height)
            if fps:
                self.cam.set(cv2.CAP_PROP_FPS, fps)
            self.running = True
            self.thread = threading.Thread(target = self.calc_BRG)
            self.thread.start()
            self.ret_val = True
        except:
            self.running = False
            self.ret_val = False

    def stop(self):
        self.running = False
        self.thread.join()

    def calc_BRG(self):
        while self.running:
            try:
                self.ret_val = False
                self.ret_val, img = self.cam.read()
                h, w, c = img.shape
                brg = img[int(h/2),int(w/2)]
                self.callback(self.ret_val,brg)
            except:
                self.running = False

    def cameraFs(self):
        return self.cam.get(cv2.CAP_PROP_FPS)
```