DSP Assignment 3 IIR filtering Report

Ben Frazer*

January 2, 2022

1 Introduction BEN:PETER:MAIN

☐ Present the problem

☐ describe the working principles of an accelerometer Bases tilt sensor

☐ design objectives

2 Working Principles BEN:PETER:MAIN

⊟ high level data flow diagrams

☐ Underlying principles

3 Filter design objectives BEN:MAIN

☐ justify filter design requirements

- translate the high-level project design objectives into filter design objectives

☐ discuss key design metrics

☐ discuss importance of rise time as a metric for "responsiveness"

☐ filter dataflow diagram + discuss

☐ present discuss final design objective for filter design stage

*2704250F@student.gla.ac.uk

- cutoff frequency
    - ??
- filter type
    - Butterworth
    - because??
- order
    - second order?
    - no cascading because we only want a lowpass response
    - because we want our code to execute as fast as possible
    - we are not as concerned about steep cutoffs

3.1 Sensor noise — Ben

☐ discuss noise in accelerometers in general

☐ potential additional noise sources (ADCs/EMF)

☐ mention that noise tends to be relatively speaking higher frequency and uniformly distributed in the frequency domain

3.2 Procedure for Quantifying sensor noise — Ben

☐ present approach for quantifying noise

☐ discuss problems due to dc spectral leakage

☐ eliminate using time-domain first order polynomial fit

3.3 Harmonic content due to sensor movement — Ben

☐ we want to reject this

☐ discuss how sensor motion will disturb the measurement of angle

- ☐ (present figure) show the harmonic content of the sensor under motion vs at rest
  - comparison plots should be at the same sample rate and length (can be achieved by cropping the longer "at rest" recording)
  - identify and comment on the frequency range of these harmonics
- ☐ comment on the trade off between settling time and rejection of noise/ disturbances and settling time
- ☐ lower priority than sensor noise

3.4 Settling Time — Ben

- ☐ discuss how settling time is a metric for responsiveness
- ☐ point out the fundamental 2 between filter cutoff frequency and

3.5 influence of Filter order — Ben

- ☐ Filter order directly influences the maximum achievable transition width
- ☐ Attenuation after the cuttof is not an important metric since disturbances are transient and noise is flatly distributed
- ☐ As the cuttof tends towards lower frequencies, the relative "flatness" of the pass-band is diminished resulting in exponential increase in settling time for a given increase in frequency
- ☐ Present plot showing different filter orders
- ☐ transition width becomes important because it also affects settling time
- ☐ It is effectively a free degree of freedom that allows a limited improvement to settling time at the cost of additional computation
- ☐ only important when cutoff frequency bottoms out

3.6 Final design — Ben

4 Implementation — Ben:Cammeron:main

- ☐ discuss how sample rate was checked

☐ discuss unit testing

- include here the tables of filter coefficients used (refer to dataflow diagram)

☐ maybe discuss how angle measurements were derived

☐ present images of the setup

4.1 Filter design — Cammeron

☐ just a brief overview of the high-level design commands used

☐ Reiterate the cutoff frequency decided above

4.2 Filter Unit Testing — Cammeron

4.3 Sample rate Verification — Cammeron

4.4 Angle measurement — Ben

5 Results — Cammeron:main

Treat this section as presentation of results

☐ present plot of the raw vs filtered data

⊟ discuss the achieved sample rate

☐ link to youtube video

☐ with reference to the youtube video + cammeron's pics with the protractor discuss success

6 Design Review — Ben:main

☐ discuss limitations

- influence of sample rate on cutoff frequency

7 Future work — BEN:MAIN

☐ improvements

  ☐ have the filter design recalculate based on average sample-rate

  ☐ more instantaneous response using a gyroscope + sensor fusion

8 conclusion — PETER:CAMMERON:BEN:MAIN

☐ challenges faced

  ☐ problems with firmatta I2C (documented progress)

☐ reiterate that we were successful

9 Appendices — BEN

☒ include unit test code

☒ include links section:

  – the git repo

  – the youtube video

  – link to bernds source code

☐ description of the link

9.1 Code

9.1.1 realtime_iir_main.py

```
""" Modules """
from pyfirmata2 import Arduino
from scipy import signal
import numpy as np

import realtime_plot as rtp
import iir_filter as iir

```

```
""" Constants """
# Arduino Sample Rate
fs = 1000   # Max 1000
# Nyquist
fn = fs / 2

""" IIR Filter Design """
# Noise Removal
sos = signal.butter(1, 1 / fn, "lowpass", output="sos")

x_filter = iir.IIR_filter(sos)
y_filter = iir.IIR_filter(sos)
z_filter = iir.IIR_filter(sos)

""" Real Time Plotters """
sample_plot = rtp.RealtimePlots(fs, 2, ["X", "Y", "Z"],
 ↪  sample_limits=[-5,5], channels=3)
orientation_plot = rtp.RealtimeVectorPlot()

""" Convert Normalized Voltage to Acceleration """
def v2a(n_volt):
    # Normalised voltage to voltage and Re-centre (3.3V / 2 = 0g)
    volts = (n_volt * 5) - (3.3 / 2)
    # Convert to acceleration (300mV per g)
    return volts / 0.3

""" Sample Process Function """
def addX(data):
    acc = v2a(data)
    f_acc = x_filter.filter(acc)

    sample_plot.addSample(acc, f_acc, channel=0)
    orientation_plot.addSample(f_acc, channel=0)

def addY(data):
    acc = v2a(data)
    f_acc = y_filter.filter(acc)

    sample_plot.addSample(acc, f_acc, channel=1)
    orientation_plot.addSample(f_acc, channel=1)

def addZ(data):
    acc = v2a(data)
    f_acc = z_filter.filter(acc)

    sample_plot.addSample(acc, f_acc, channel=2)
    orientation_plot.addSample(f_acc, channel=2)

""" Aurdino Data Aquisition """
```

```
board = Arduino(Arduino.AUTODETECT)
board.samplingOn(1000/fs)
board.analog[0].register_callback(addX)
board.analog[0].enable_reporting()
board.analog[1].register_callback(addY)
board.analog[1].enable_reporting()
board.analog[2].register_callback(addZ)
board.analog[2].enable_reporting()

""" Show Real Time Plots """
rtp.plt.show()
board.exit()
```

9.1.2 rununittest.py

```
import unittest
import iir_filter
import scipy.signal as signal

"""Function used to collect the IIR filter output influenced by the
    specifications"""


def getCoefficients(w, order, system_type, input_signal):
    coeff = []
    irr_result = []
    b, a = signal.butter(order, 2 * w, system_type)

    for i in b:
        coeff.append(i)

    for i in a:
        coeff.append(i)

    f = iir_filter.IIR2_filter(coeff)

    for i in range(len(input_signal)):
        irr_result.append(round(f.filter(input_signal[i]), 4))

    return irr_result


"""Function used to collect the IIR filter output influenced by SOS at
    given specifications"""


def getSOSCoefficients(w, order, system_type, input_signal):
    sos = signal.butter(order, 2 * w, system_type, output='sos')
    sos_check = []
```

```
33
34      fi = iir_filter.IIR_filter(sos)
35
36      for i in range(len(input_signal)):
37          sos_check.append(round(fi.filter(input_signal[i]), 4))
38
39      return sos_check
40
41
42  """Unit test class function for a lowpass and highpass filters"""
43
44
45  class TestStringMethods(unittest.TestCase):
46
47      def test_lowpass_2nd_order_filter(self):
48          cutoff_freq = 0.1  # define the normalized cutoff frequency
49          filter_order = 2  # state the filter order
50          filter_type = 'lowpass'  # state the type of filter used
51          test_input = [1, 3, 5]  # define the input signal coefficients
52
53          hand_calculated_values = [0.0675, 0.4144, 1.2552]  # array
    ↪ that contains the hand calculated values for the
54          # above specification
55
56          filter_calculated_values = getCoefficients(cutoff_freq,
    ↪ filter_order, filter_type,
57                                                     test_input)  #
    ↪ array contains the IIR filter output values for the
58          # same specification
59
60          self.assertEqual(hand_calculated_values,
61                           filter_calculated_values)  # compare both
    ↪ arrays to check if they aer similar
62
63      def test_lowpass_sos_filter(self):
64          cutoff_freq = 0.1  # define the normalized cutoff frequency
65          filter_order = 2  # state the filter order
66          filter_type = 'lowpass'  # state the type of filter used
67          test_input = [1, 3, 5]  # define the input signal coefficients
68
69          hand_calculated_values = [0.0675, 0.4144, 1.2552]  # array
    ↪ that contains the hand calculated values for the
70          # above specification
71
72          filter_sos_calculated_values = getSOSCoefficients(cutoff_freq,
    ↪ filter_order, filter_type,
73                                                            test_input)
    ↪ # array contains the sos influenced IIR filter
74          # output values for the same specification
```

```
75
76          self.assertEqual(hand_calculated_values,
77                           filter_sos_calculated_values)  # compare both
    ↪ arrays to check if they aer similar
78
79      def test_highpass_2nd_order_filter(self):
80          cutoff_freq = 0.3  # define the normalized cutoff frequency
81          filter_order = 2  # state the filter order
82          filter_type = 'highpass'  # state the type of filter used
83          test_input = [6, -8, 3]  # define the input signal
    ↪ coefficients
84
85          hand_calculated_values = [1.2394, -4.5894, 6.6175]  # array
    ↪ that contains the hand calculated values for the
86          # above specification
87
88          filter_calculated_values = getCoefficients(cutoff_freq,
    ↪ filter_order, filter_type,
89                                                     test_input)  #
    ↪ array contains the IIR filter output values for the
90          # same specification
91
92          self.assertEqual(hand_calculated_values,
93                           filter_calculated_values)  # compare both
    ↪ arrays to check if they aer similar
94
95      def test_highpass_sos_filter(self):
96          cutoff_freq = 0.3  # define the normalized cutoff frequency
97          filter_order = 2  # state the filter order
98          filter_type = 'highpass'  # state the type of filter used
99          test_input = [6, -8, 3]  # define the input signal
    ↪ coefficients
100
101         hand_calculated_values = [1.2394, -4.5894, 6.6175]  # array
    ↪ that contains the hand calculated values for the
102         # above specification
103
104         filter_sos_calculated_values = getSOSCoefficients(cutoff_freq,
    ↪ filter_order, filter_type,
105                                                           test_input)
    ↪ # array contains the sos influenced IIR filter
106         # output values for the same specification
107
108         self.assertEqual(hand_calculated_values,
109                          filter_sos_calculated_values)  # compare both
    ↪ arrays to check if they aer similar
110
111 try: # BF 20/12/21 - check if i am being run in the ipython shell
112     __IPYTHON__
```

```
except:
    __IPYTHON__ = False

if not __IPYTHON__:
    if __name__ == '__main__':
        unittest.main()
else: # BF 20/21/21 - This code allows the unit test to run in the IPyhton shell
    unittest.main(argv=['first-arg-is-ignored'], exit=False)
```

9.1.3 calcAngles.py

```
#!/usr/bin/env ipython
import unittest
import numpy as np
import numpy.linalg as ln


def calcAngles(vec, DoDebug=False):
    """returns the angles of a 3d vector relative to the orthoginal unit vectors"""
    angle = np.array([0,0,0])
    referenceFrame = []
    referenceFrame.append([1,0,0])
    referenceFrame.append([0,1,0])
    referenceFrame.append([0,0,1])
    for i in range(3):
        if DoDebug:
            print(f"unitVec{i} = {referenceFrame[i]}")
        part1=np.dot(vec,referenceFrame[i])/(ln.norm(vec)*ln.norm(referenceFrame[i]))
        angle1 = np.arccos(part1)
        angle[i] = np.rad2deg(angle1)
        #angle1 = np.arcsin(abs(vec[i])/np.linalg.norm(vec))
        #angle[i] = 90 - np.rad2deg(angle1)
        if DoDebug:
            print(f"angle[{i}] = {angle[i]}")
    return angle


###################### Tests #################################
testVec = [99.1,99.1,99.1]
class TestCalcAngles(unittest.TestCase):

    def test1(self):
        testvec = [1,1,1]
        angles = calcAngles(testVec)
        didPass = np.testing.assert_array_almost_equal([54,54,54],angles)
```

```
        #print(didPass)



try: # BF 20/12/21 - check if i am being run in the ipython shell
    __IPYTHON__
except:
    __IPYTHON__ = False

if not __IPYTHON__:
    if __name__ == '__main__':
        unittest.main()
else: # BF 20/21/21 - This code allows the unit test to run in the IPyhton shell
    unittest.main(argv=['first-arg-is-ignored'], exit=False)
```

9.1.4 realtime_iir_main.py

```
""" Modules """
from pyfirmata2 import Arduino
from scipy import signal
import numpy as np

import realtime_plot as rtp
import iir_filter as iir

""" Constants """
# Arduino Sample Rate
fs = 1000   # Max 1000
# Nyquist
fn = fs / 2

""" IIR Filter Design """
# Noise Removal
sos = signal.butter(1, 1 / fn, "lowpass", output="sos")

x_filter = iir.IIR_filter(sos)
y_filter = iir.IIR_filter(sos)
z_filter = iir.IIR_filter(sos)

""" Real Time Plotters """
sample_plot = rtp.RealtimePlots(fs, 2, ["X", "Y", "Z"], sample_limits=[-5,5], channels=3)
orientation_plot = rtp.RealtimeVectorPlot()

""" Convert Normalized Voltage to Acceleration """
def v2a(n_volt):
    # Normalised voltage to voltage and Re-centre (3.3V / 2 = 0g)
    volts = (n_volt * 5) - (3.3 / 2)
```

```
    # Convert to acceleration (300mV per g)
    return volts / 0.3

""" Sample Process Function """
def addX(data):
    acc = v2a(data)
    f_acc = x_filter.filter(acc)

    sample_plot.addSample(acc, f_acc, channel=0)
    orientation_plot.addSample(f_acc, channel=0)

def addY(data):
    acc = v2a(data)
    f_acc = y_filter.filter(acc)

    sample_plot.addSample(acc, f_acc, channel=1)
    orientation_plot.addSample(f_acc, channel=1)

def addZ(data):
    acc = v2a(data)
    f_acc = z_filter.filter(acc)

    sample_plot.addSample(acc, f_acc, channel=2)
    orientation_plot.addSample(f_acc, channel=2)

""" Aurdino Data Aquisition """
board = Arduino(Arduino.AUTODETECT)
board.samplingOn(1000/fs)
board.analog[0].register_callback(addX)
board.analog[0].enable_reporting()
board.analog[1].register_callback(addY)
board.analog[1].enable_reporting()
board.analog[2].register_callback(addZ)
board.analog[2].enable_reporting()

""" Show Real Time Plots """
rtp.plt.show()
board.exit()
```

9.1.5 realtime_plot.py

```
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.animation import FuncAnimation
import matplotlib.pyplot as plt
import numpy as np
import time
import calcAngles
""" Rolling Buffer """
class RollingBuffer:
```

```
    def __init__(self, size):
        self.size = size
        self.plot_buffer = np.zeros(self.size)
        self.data_buffer = []

    def update(self):
        self.plot_buffer = np.append(self.plot_buffer,
    ↪ self.data_buffer)
        self.plot_buffer = self.plot_buffer[-self.size:]
        self.data_buffer = []
        return self.plot_buffer

    def add(self, v):
        self.data_buffer.append(v)

""" Real-Time Plotter Variable Channel """
class RealtimePlots:
    def __init__(self, fs, window_time, labels, sample_limits=[-0.5,
    ↪ 0.5], channels=1):
        # Buffer
        self.buffer_size = fs * window_time
        self.r_buffers = []
        self.f_buffers = []

        # Figure Plot
        self.fig, self.ax = plt.subplots(nrows=2)

        # Sample Plot
        self.ax[0].plot([0,
    ↪ self.buffer_size-1],[0,0],"r--",label="Zero")
        self.ax[0].set_ylim(sample_limits[0], sample_limits[1])
        self.ax[0].set_title("Un-Filtered")
        self.r_lines = []

        # Filter Plot
        self.ax[1].plot([0,
    ↪ self.buffer_size-1],[0,0],"r--",label="Zero")
        self.ax[1].set_ylim(sample_limits[0], sample_limits[1])
        self.ax[1].set_title("Filtered")
        self.f_lines = []

        # Plot Buffers
        for i in range(channels):
            self.r_buffers.append(RollingBuffer(self.buffer_size))
            line, = self.ax[0].plot(self.r_buffers[i].update(),
    ↪ label=labels[i])
            self.r_lines.append(line)

            self.f_buffers.append(RollingBuffer(self.buffer_size))
```

```
            line, = self.ax[1].plot(self.f_buffers[i].update(),
    ↪  label=labels[i])
            self.f_lines.append(line)
        self.ax[1].legend(loc=4)

        self.anim = FuncAnimation(self.fig, self.update, interval=100)
        self.update_count = 0

        # Sample Rate
        self.sample_count = 0
        self.label = self.ax[0].text(0, sample_limits[0], "Sample
    ↪  Rate: -", ha="left", va="bottom", fontsize=15)
        self.last = 0

    def update(self, x):
        # Buffer
        for i in range(len(self.r_lines)):
            self.r_lines[i].set_ydata(self.r_buffers[i].update())
            self.f_lines[i].set_ydata(self.f_buffers[i].update())

        # Sample Rate Calc
        if self.update_count % 5 == 0: # Reduce updates for
    ↪  performance
            current_time = time.time()
            sample_rate = self.sample_count / (current_time -
    ↪  self.last)
            self.last = current_time
            self.sample_count = 0

            self.label.set_text(f"Fs: {sample_rate:.1f}Hz")
        self.update_count += 1

    def addSample(self, v, f, channel=0):
        if channel == 0:
            self.sample_count += 1
        self.r_buffers[channel].add(v)
        self.f_buffers[channel].add(f)

""" Real-Time Vector Plotter """
class RealtimeVectorPlot:
    def __init__(self):
        self.vec = np.zeros(3)

        self.fig, self.ax =
    ↪  plt.subplots(subplot_kw=dict(projection="3d"))
        self.ax.set_xlim(-1.0, 1.0)
        self.ax.set_ylim(-1.0, 1.0)
        self.ax.set_zlim(-1.0, 1.0)
        self.ax.set_xticks([])
```

```
97          self.ax.set_yticks([])
98          self.ax.set_zticks([])
99
100         self.anim = FuncAnimation(self.fig, self.update, interval=100)
101         self.q = self.ax.quiver(0, 0, 0, 0, 0, 1)
102         self.x = self.ax.quiver(0, 0, 0, 1, 0 ,0, color="red")
103         self.y = self.ax.quiver(0, 0, 0, 0, 1 ,0, color="green")
104         self.z = self.ax.quiver(0, 0, 0, 0, 0 ,1, color="blue")
105
106         self.tLastUpdate = 0
107         self.updatePeriod_s = 1
108         self.label = self.ax.text(0, 0, 0, "test",
    ↪ transform=self.ax.transAxes)
109         self.angle = np.array([0,0,0])
110
111     def update(self, x):
112         m = np.max(np.abs(self.vec))
113         if m != 0:
114             self.q.remove()
115             self.q = self.ax.quiver(0, 0, 0, self.vec[0] / m,
    ↪ self.vec[1] / m, self.vec[2] / m, color="black")
116             #angle calcs
117             now = time.time()
118             if (now - self.tLastUpdate)>self.updatePeriod_s:
119                 #angle between the vector and the x plane
120                 self.angle = calcAngles.calcAngles(self.vec)
121                 self.label.set_text(f"angle (deg) x:{self.angle[0]},
    ↪ y:{self.angle[1]}, z:{self.angle[2]}")
122                 self.tLastUpdate = now
123
124     def addSample(self, v, channel=0):
125         self.vec[channel] = v
```

9.2 Links

9.2.1 Project Source code — Ben

9.2.2 Filter Source code — Ben

9.2.3 Project demonstration YouTube Video — Cammeron