DSP Assignment 3 IIR filtering Report

Partheepan Shiyamsunthar, Benjamin Frazer, Cameron Richards*

January 3, 2022

Contents

*2531525S@student.gla.ac.uk, 2704250F@student.gla.ac.uk, 2391317R@student.gla.ac.uk

1 Introduction

This project aims to deliver an accelerometer based angel measurement device capable of returning the angle between a vector drawn between the device and the centre of the earth and three orthogonal coordinate vectors and as such providing a measurement of the tilt of the sensor in 3d space.

The high-level design objectives for this project are summarised as follows:

- return angle measurements for pitch and yaw
- ensure smooth measurements (un-disturbed by noise)
- make responsive measurements (if the user changes the angle the change in measurement is perceived as instantaneous)

2 Working Principles

The tilt sensor is predicated on the knowledge that the gravity vector will always be pointed directly down towards the centre of the earth. If the assumption is then made that the device is otherwise at rest, we may infer the pitch and roll of the device by observing the projection of the gravity vector on the respective axes.

The sensor used to measure acceleration is the ADXL335, this is a three axis accelerometer which outputs its values via DAC as three analogue signals. These analogue signals are digitised by the Arduino which is acting purely as an acquisition device. Real-time data is then passed python using the firmatta protocol, where data processing such as filtering and angle measurement occurs. Figure 1 shows the high-level structure of the tilt sensor.

3 Filter design objectives

The high-level design objectives can be translated into requirements for the filter design, namely:

- Eliminate sensor noise
- Eliminate environmental noise where possible i.e. vibrations
- Good transient behaviour for DC changes i.e. fast response, no overshoot

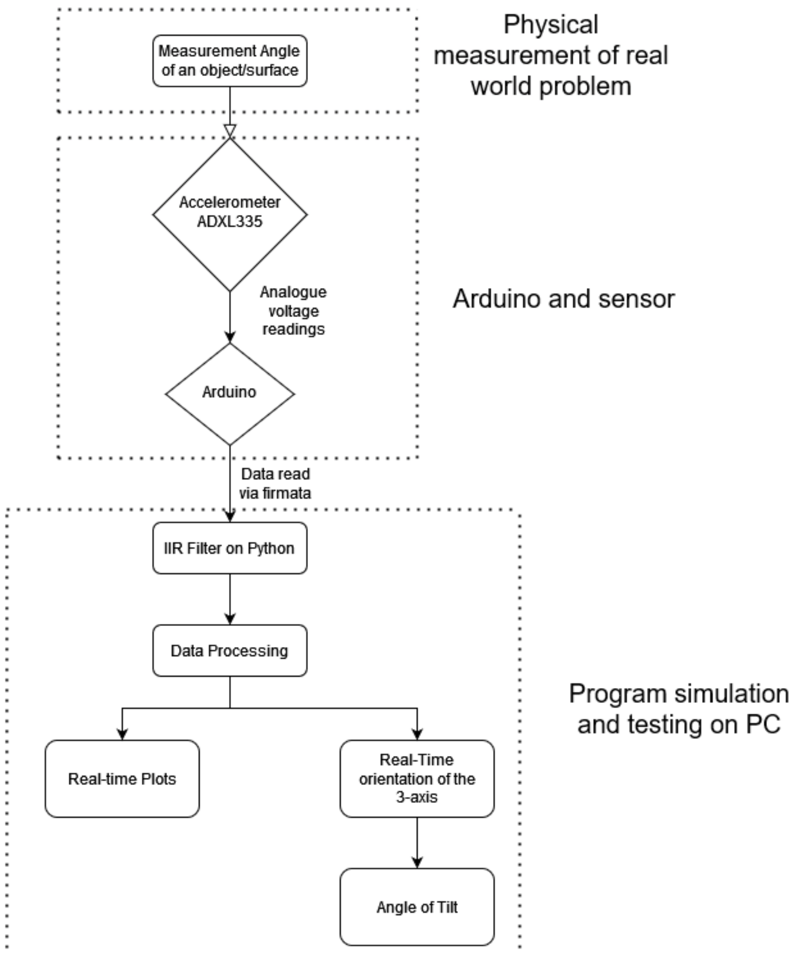

Figure 1: Tilt sensor High-level Dataflow Diagram

Acceleration at rest will be a dc value, as such a low-pass filter is required.

The primary requirement on the filter implementation is whether the filter can be realised in real time. Since python is an interpreted language, the speed of execution is fundamentally limited and as such the amount of computation required for filtering should be minimised. In this regard, IIR filters are certainly the most optimal, since on a per computation basis they offer better performance than FIR filters. Higher order IIR filters will of course still come at the cost of more computation and as such, the order of the filters should be kept minimal as a design goal.

In order to address the design goal of "responsiveness" we must first identify a metric which can suitably quantify this characteristic in the filter design. Since an instantaneous change in orientation will appear as a step change in the acceleration of a given accelerometer channel, it seems suitable to use the settling time as a metric for this goal. For the purpose of this report settling time is defined as the time taken for the step response to be bounded by ± 10

3.1 Analogue Prototype

The first design decision to be made is that of the analogue filter which will form the basis of the the IIR filter design the following constitute the leading candidates:

Table 1: Qualitative comparison of candidate analogue prototype filters

filter type	Overshoot	transition width	rise time	pass-band Flatness	Stop-band Att
Chebychev	poor	best	poor	poor	good
Butterworth	poor	medium	poor	best	good
Bessel	best	poor	best	poor	good

Acceleration measured due to gravity is at steady state a dc value, as such the filter flatness in the pass band is not of interest, since even the distortion at DC caused by the Chebychev will not affect the angle measurement as long as it is consistent between channels. Further more, it is assumed that sensor noise will be flatly distributed in the frequency domain meaning there is no pressing requirement for a sharp transition width. With the aforementioned considerations mentioned it is clear that the benefits of the Chebychev and the Butterworth filters are largely negated by the design requirements.

The requirement for good transient characteristics penalises the Butterworth and Chebychev filters since both suffer from overshoot while the Bessel filter doesn't. The Bessel filter also has a faster settling time when compared to the other analogue designs.

3.2 Sensor noise

The first factor influencing the choice of filter cut-off is the distribution of filter noise. The noise content of the accelerometer data is confirmed by taking a recording of the filter at rest and taking a Fourier transform for a single channel. [1]

As we can see from Figure 2, sensor noise is relatively evenly distributed in the frequency domain. with only a slight concentration of harmonics around 100Hz. This means that there is no particular requirement to have a steep transition width since we are only concerned with unity response at DC and have roughly

[1] After investigation noise is similar in all three channels but only one is plotted here

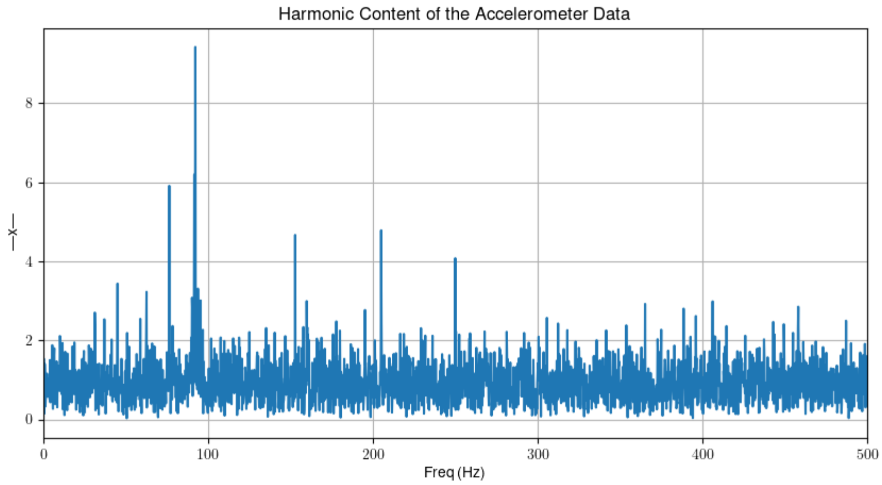

Figure 2: Fourier transform of raw accelerometer data showing sensor noise

two decades of roll-off before the dominant frequencies at 100Hz at which point even a first order bessel will suffice.

3.3 Settling Time vs Cut-off Frequency

Since the objective of the filter is to reject all but DC but also remain responsive (settling time) we must consider if and how these two requirements interact. As shown in Figure 3 there exists a fundamental trade-off between settling time and cut-off frequency.

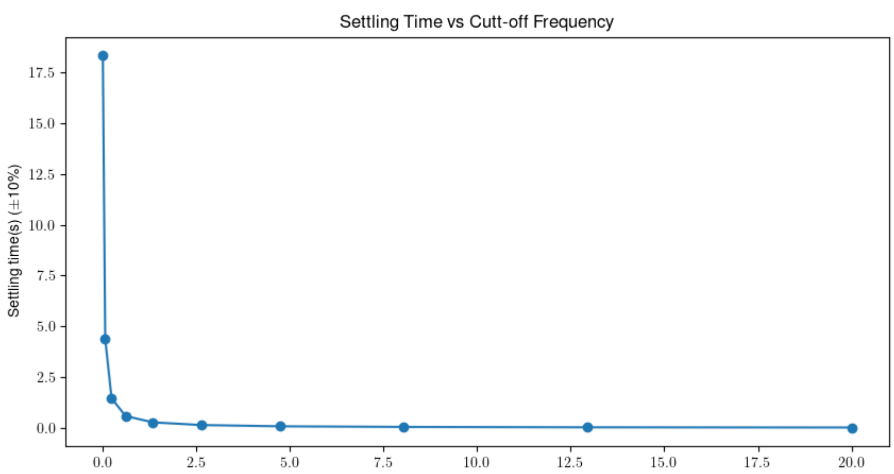

Figure 3: Trade-off between settling time and cut-off frequency for a second order Bessel filter

This plot was generated numerically by applying a step filter designs with different cutoff frequencies and logging the settling time (for Bessel this is simply the first intersection between the step response and line $y = 0.9$) denoted in Figure 4. The cut-off frequencies are exponentially spaced between 0.2 and 20 Hz in order to have good coverage of the corner between the two asymptotes.

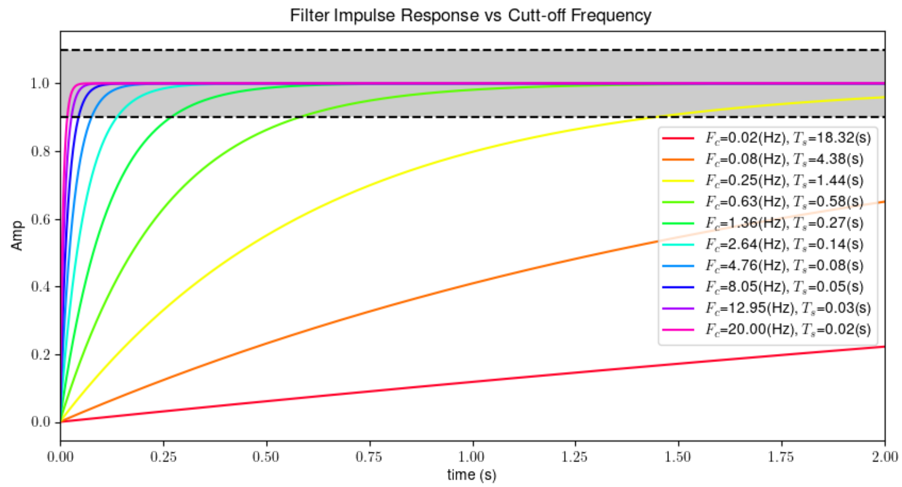

Figure 4: Impulse, frequency domain responses of second order Bessel filters with various cutoff frequencies.

For the final design, a cut-off frequency of 5 Hz will be used since this preforms adequately at removing noise while incurring little to no penalty in responsiveness (settling time).

3.4 Filter order

A further degree of freedom in the design is that of the filter order used. Typically the impact of higher filter order is faster transition width, which, as mentioned previously is only a very minor consideration here, while the increased computational cost associated with this incur a substantial penalty. It is plausible however that increased filter order might have an impact on the rise time of the filter. Figure 5 shows the trade-off plots for different orders of Bessel Filter.

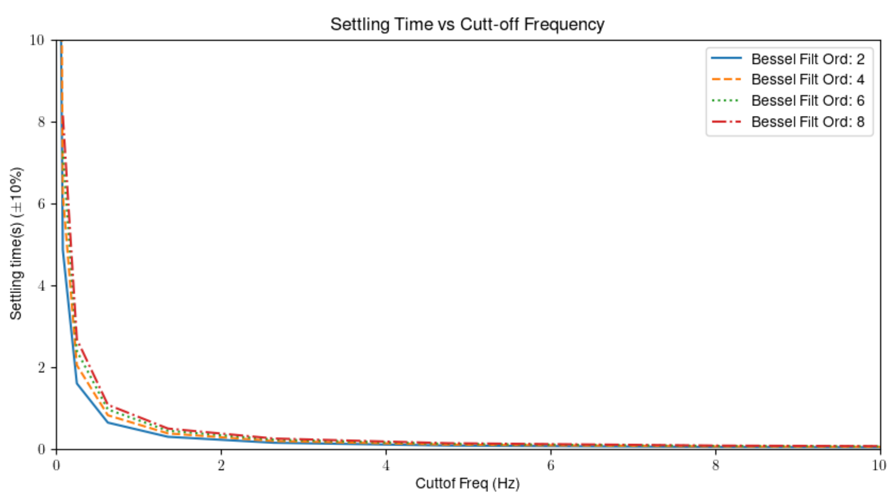

Figure 5: Trade-off between settling time and cut-off frequency for multiple orders Bessel filter orders

From this figure we can see that increasing filter order has a detrimental impact on the settling time trade-off curve. It is a simple decision therefore to limit the filter order to first order given that there is little benefit decreased transition width. This plot was again generated numerically using exponentially spaced cut-off frequencies between 0 and 20Hz but as well as even filter orders between 2 and 8. The corresponding impulse and frequency responses of these filters are shown below.

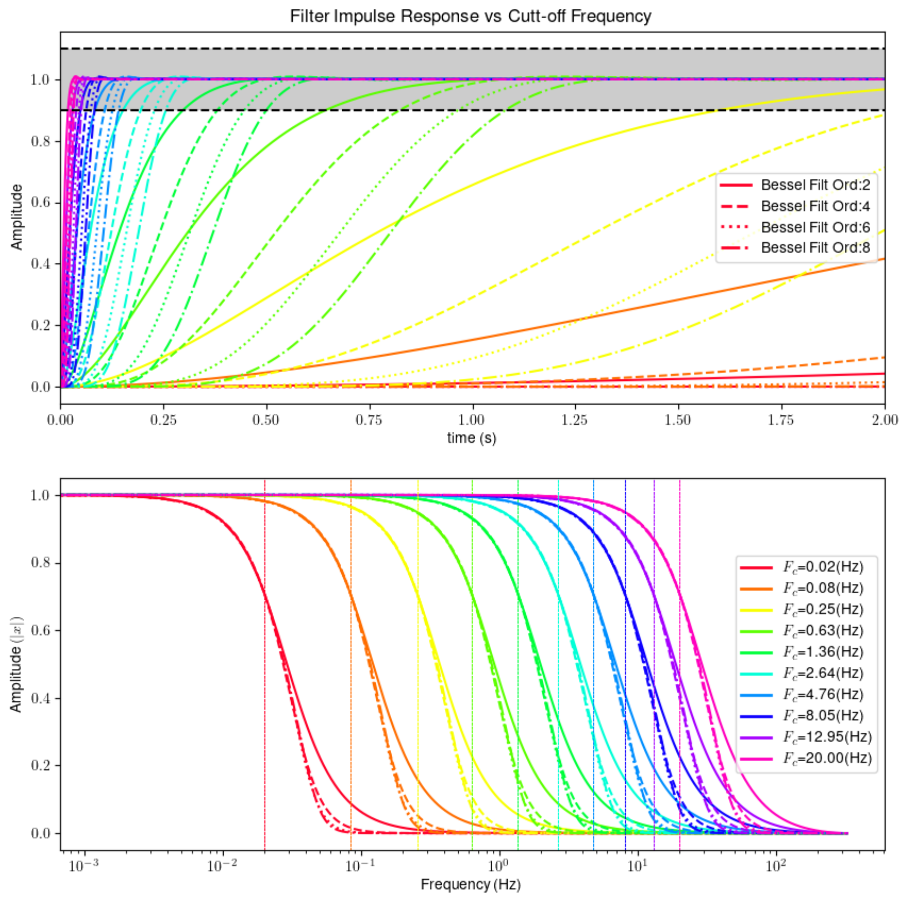

Figure 6: Impulse, frequency domain responses of second order Bessel filters with various cutoff frequencies and filter orders.

3.5 Final design

The Final Filter design parameters are given in Table 2. [2]

[2]Filter coefficients will be presented in the following sections, since this is a function of sample rate.

Table 2: final filter design parameters

Analogue Prototype	Filter order	Cut-off Frequency (Hz)
Bessel	1st	5

4 Implementation

4.1 Sample rate Verification

To get the sampling rate the number of samples recorded of a period of time were counted and then divided by the period length:

$$fs = NT \tag{1}$$

The period that samples were counted for was chosen based on the knowledge that the measurement period must be much larger than the sample rate period. The system should run at 1kHz sample rate so will have a 1ms sampling period. If the measurement period is too large the sample rate won't be calculated often enough to be practical. Therefore, a measurement period of 500ms was selected to balance both these factors.

In practice to get the half second timing the animation callback feature of matplotlib which is used to update the plot every ˜100ms. A counter system was used so that the sample rate is calculated every five times the plot is updated. However, the animation callback does not have precise timing so the "time" module was used to precisely measure the time between the sample rate calculations.

This method was implemented and tested single input channel at 1kHz where the measured value stayed between 950Hz and 1050Hz. However, when more input channels are added the sample rate starts to reduce. This may be because either the Arduino or the python script can't run fast enough to process that many samples.

4.2 Filter design

The final design coefficients were derived using the scipy signal library bessel design command source-code shown below:

```python
from scipy import signal
fs = 650 # sample rate
fn = fs/2
fc = 2 # cutoff frequency (Hz)
sos = signal.bessel(1, fc / fn, "lowpass", output="sos", norm="mag")
```

This design function outputs filter coefficients grouped into bi-quads. The basic structure of such a biquad is shown in Figure 7. This is the coefficient naming convention that will be used when discussing and presenting filter designs in this report.
Since the achieved sample rate was substantially lower than the requested 1kHz, the sample rate used to derive the normalised cut-off frequency was modified to account for this.
This results in the following filter coefficients:

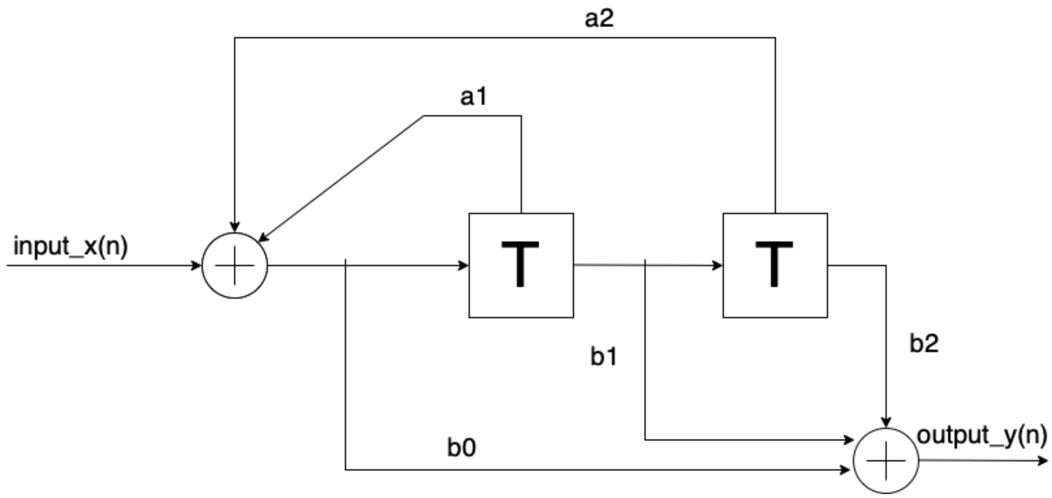

Figure 7: Direct form II Biquad IIR filter showing coefficient naming convention

Table 3: Final Filter Design Coefficients

b0	b1	b2	a0	a1	a2
9.67419e-3	9.57419e-3	0	1	0.98085162	0

4.3 Filter Unit Testing

Unit testing is conducted by first generating a generic high and lowpass design and then hand calculating the expected output by using the difference equation for a second order IIR filter.

The transfer function for a second order biquad filter is given in the z domain by:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{a_0 + a_1 z^{-1} + a_2 z^{-2}} \tag{2}$$

This yields the following difference equation in the time-domain:

$$v[n] = x[n] - a_1 \times y[n-1] - a_2 \times y[n-2] \tag{3}$$
$$y[n] = v[n] \times b_0 + b_1 \times y[n-1] + b_2 \times y[n-2] \tag{4}$$

For the purpose of simplifying notation in the hand calculations the following substitutions will be made:

$$y[n-1] = T1 \tag{5}$$
$$y[n-2] = T2 \tag{6}$$

These difference equations allow the expected output of the filter to be calculated recursively.

4.3.1 Generic Low-Pass Unit test design:

This is a generic 2nd order IIR Filter with a cut-off at a normalized frequency of 0.1

Table 4: Contains the filter coefficients for the low pass unit test filter

b0	b1	b2	a0	a1	a2
0.06745527	0.13491055	0.06745527	1	1.1429805	-0.4128016

4.3.2 Low-Pass hand calculations

Hand calculation for the lowpass filter when given an input signal x(n) = [1,3,5]:

```
T1 = T2 = 0

Input = x(n) - (a1*T1) - (a2*T2)

y(n) = [input]*bo + b1*T1 + b2*T2

After each iteration, T1 = input & T2 = T1

y(1)    = [x(1) - (a1*T1) - (a2*T2)]*b0 + b1*T1 + b2*T2
      = [1 - (0) - (0)] * 0.06745527 + 0 + 0
      = 0.06745527
      = 0.0675

T2 = T1 = 0
T1 = input = 1


y(2)    = [x(2) - (a1*T1) - (a2*T2)]*b0 + b1*T1 + b2*T2
      = [3 - (-1.1429805 * 0.06745527) - (0)] * 0.06745527 + 0.13491055 * 1 + 0
      = [4.1429805] * 0.06745527 + 0.13491055 * 1 + 0
      = 0.4143764182
      = 0.4144

T2 = T1 = 1
T1 = input = 4.1429805

y(3)    = [x(3) - (a1*T1) - (a2*T2)]*b0 + b1*T1 + b2*T2
      = [5 - (-1.1429805 * 4.1429805) - ( 0.4128016 * 1)] * 0.06745527 + 0.13491055
              * 4.1429805 + 0.06745527 * 1
      = [9.322544323]  * 0.06745527 + 0.13491055 * 4.1429805 + 0.06745527 * 1
      = 1.255241792
      = 1.2552

T2 = T1 = 4.1429805
T1 = input = 9.322544323
```

Therefore our output y(n) = [0.0675, 0.4144, 1.2552]

4.3.3 Generic High-Pass Unit test design:

This is a 2nd order IIR Filter with a cut-off at a normalized frequency of 0.3.

Table 5: Contains the filter coefficients for the high pass unit test filter

b0	b1	b2	a0	a1	a2
0.206572	-0.413144	0.206572	1	0.369527	- 0.195815

Compare these output values with the filter output when used in the unittest.py program.

4.3.4 Highpass hand calculations

Hand calculation for the highpass filter when given an input signal x(n) = [6,-8,3]:

```
T1 = T2 = 0

Input = x(n) - (a1*T1) - (a2*T2)

y(n) = [input]*bo + b1*T1 + b2*T2

After each iteration, T1 = input & T2 = T1

y(1)    = [x(1) - (a1*T1) - (a2*T2)]*b0 + b1*T1 + b2*T2
      = [6 - (0) - (0)] * 0.20657208 + 0 + 0
      = 1.23943248
      = 1.2394

T2 = T1 = 0
T1 = input = 6

y(2)    = [x(2) - (a1*T1) - (a2*T2)]*b0 + b1*T1 + b2*T2
      = [-8 - (0.36952738 *6) - (0)] * 0.20657208 + (-0.41314417 * 6)+ 0
      = [-10.21716428]  * 0.20657208 + (0.41314417 * 6)+ 0
      = -4.589445897
      = -4.5894

T2 = T1 = 6
T1 = input = -10.21716428

y(3)    = [x(3) - (a1*T1) - (a2*T2)]*b0 + b1*T1 + b2*T2
      = [3 - (0.36952738 * -10.21716428) - (0.19581571 * 6)] * 0.20657208+ (-0.41314417
             * -10.21716428) + 0.20657208 * 6
      = [5.600627687] *  0.20657208+ (-0.41314417 * -10.21716428) + 0.20657208 * 6
      = 6.617527647
      = 6.6175
```

```
T2 = T1 = -10.21716428
T1 = input = 5.600627687

Therefore our output y(n) = [1.2394, -4.5894, 6.6175]
```

4.4 Angle measurement

The three angels defining the orientation of the device are simply defined as the angel between the orientation vector and the three orthogonal coordinate axes. Since this formula is invariant to the magnitude of the vectors involved, the unit vector is used for simplicity (denoted as $\hat{x}$).

$$xangle = cos^{-1}\frac{\vec{V}_{xyz} \cdot \hat{x}}{|\hat{x}| \times |\vec{V}_{xyz}|} \tag{7}$$

The source code for the is shown below, this calculation is implemented for all three angels in a for loop. [3]

```python
def calcAngles(vec):
    """returns the angles of a 3d vector relative to the orthoginal unit vectors"""
    angle = np.zeros(3)
    referenceFrame = []
    referenceFrame.append([1.0,0.0,0.0])
    referenceFrame.append([0.0,1.0,0.0])
    referenceFrame.append([0.0,0.0,1.0])
    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)
```

4.5 Setup

Figure 8 shows the setup used for testing the tilt sensor. The Arduino board is connected to the ADXL335 breakout board via three jumper wires, while power and ground is supplied from the arduino onboard power pins.

5 Results

A full demonstration of the filter operating in real time can be found here (full URL in appendices):

[3] The final angle used in the display is chosen based on which corresponds to the longest side of the breadboard, as such tilt is only returned for one axis in the demonstration video

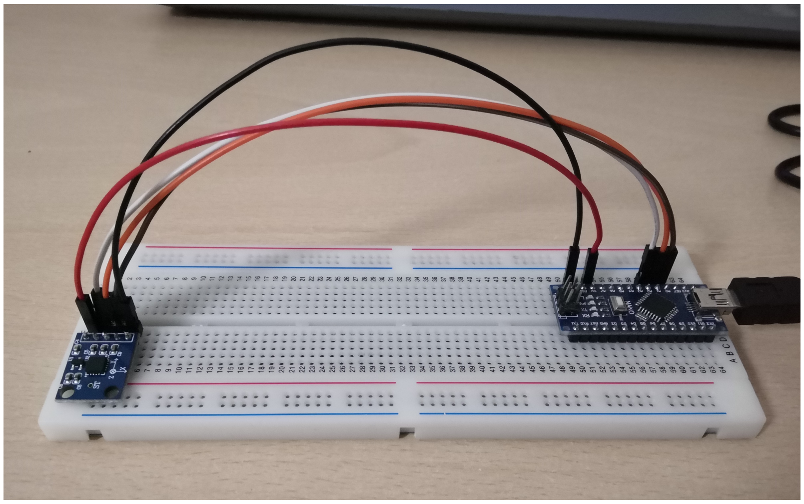

Figure 8: Tilt sensor Hardware setup

- Demonstrating Angle Measurement

- Demonstrating Responsiveness

Figures 9 and 10 show the effect of filtering on the raw accelerometer data. It can be seen that the filter acts to smooth the data while also introducing a time lag.

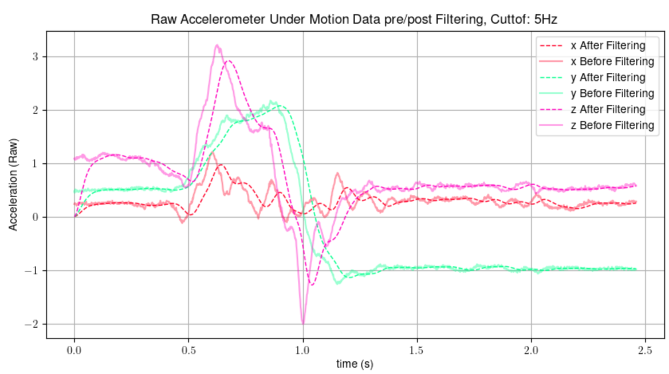

Figure 9: Raw accelerometer data before and after filtering in motion

From the video demonstration it can be seen that the tilt sensor returns accurate and stable angle measurements. Further more the tilt sensor reacts quickly to changes in angle.

6 Design Review

Given that the testing of the filter revealed that it provided accurate angle measurement, wile remaining responsive, it can be said that the initial design objectives of the project have been met.

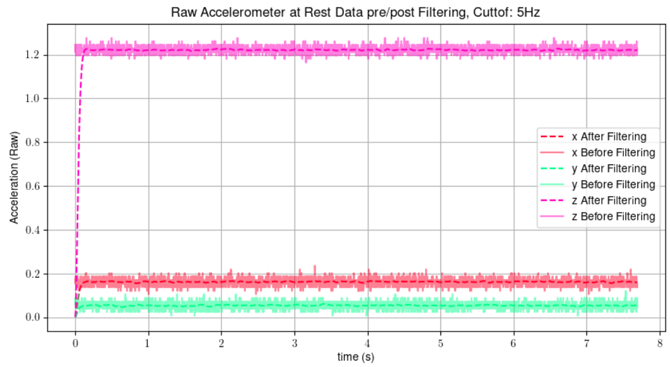

Figure 10: Raw accelerometer data before and after filtering at rest

One potential weakness of the design is the hard coded nature of the filter design while the sample rate seems constrained by the hardware in use and may as such be variable between users. This however should not substantially affect the usage of the filter since changes in sample rate for a given set of filter coefficient will merely act to move the effective cut-off frequency of the filter which, as discussed in prior sections is not a critical component of the design.

7 Future work

7.1 Digital Comms

One possible improvement could come in the form of a digital interface rather than analogue voltages between the arduino such as the I2C protocol. This was initially attempted but abandoned due to difficulties in using and debugging I2C with the firmatta library. the benefit of such a comms protocol would be eliminating a potential source of noise in the wires, and not being limited to the analogue resolution of the arduino ADCs which is substantially lower than those on the accelerometer itself.

7.2 Dynamic recalculation of filter coefficient

To combat the aforementioned drift of cut-off frequency due to hardware dependent sample rate, the filter could be recalculated "on the fly" based on the measure sample rate.

8 Appendices

8.1 Links

8.1.1 Project Source code

8.1.2 Filter Source code

8.1.3 Angle Measurment Demo - **https://youtu.be/p1zuRrrh6NQ**

8.1.4 Responsiveness Demo - **https://youtu.be/6KL1qg4h9OE**

8.2 Code

8.2.1 realtime_iir_main.py

```python
""" 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 = 650                    # BF 3/01/22 - changed to "true" sample rate Max 1000
# Nyquist
fn = fs / 2
fc = 5

""" IIR Filter Design """
# Noise Removal
sos = signal.bessel(1, fc / fn, "lowpass", output="sos", norm="mag") # BF 3/01/22 - changed to bessel

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=[-0.25,0.25], channels=3)
orientation_plot = rtp.RealtimeVectorPlot()

""" Sample Process Function """
def addX(data):
    # Zeroing from measurements
    data -= 0.335
    f = x_filter.filter(data)

```

```
    sample_plot.addSample(data, f, channel=0)
    orientation_plot.addSample(f, channel=0)

def addY(data):
    # Zeroing from measurements
    data -= 0.340
    f = y_filter.filter(data)

    sample_plot.addSample(data, f, channel=1)
    orientation_plot.addSample(f, channel=1)

def addZ(data):
    # Zeroing from measurements
    data -= 0.340
    f = z_filter.filter(data)

    sample_plot.addSample(data, f, channel=2)
    orientation_plot.addSample(f, 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()
```

8.2.2 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:
```

```python
            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 = []

    fi = iir_filter.IIR_filter(sos)

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

    return sos_check


"""Unit test class function for a lowpass and highpass filters"""


class TestStringMethods(unittest.TestCase):

    def test_lowpass_2nd_order_filter(self):
        cutoff_freq = 0.1  # define the normalized cutoff frequency
        filter_order = 2  # state the filter order
        filter_type = 'lowpass'  # state the type of filter used
        test_input = [1, 3, 5]  # define the input signal coefficients

        hand_calculated_values = [0.0675, 0.4144, 1.2552]  # array that contains the hand calculated values for the
        # above specification

        filter_calculated_values = getCoefficients(cutoff_freq, filter_order, filter_type,
                                                                         test_input)  # array contains the IIR filter output values for the
        # same specification
```

```python

        self.assertEqual(hand_calculated_values,
                         filter_calculated_values)  # compare both arrays to check if they aer similar

    def test_lowpass_sos_filter(self):
        cutoff_freq = 0.1  # define the normalized cutoff frequency
        filter_order = 2  # state the filter order
        filter_type = 'lowpass'  # state the type of filter used
        test_input = [1, 3, 5]  # define the input signal coefficients

        hand_calculated_values = [0.0675, 0.4144, 1.2552]  # array that contains the hand calculated values for the
        # above specification

        filter_sos_calculated_values = getSOSCoefficients(cutoff_freq, filter_order, filter_type,
                                                          test_input)  # array contains the sos influenced IIR filter
        # output values for the same specification

        self.assertEqual(hand_calculated_values,
                         filter_sos_calculated_values)  # compare both arrays to check if they aer similar

    def test_highpass_2nd_order_filter(self):
        cutoff_freq = 0.3  # define the normalized cutoff frequency
        filter_order = 2  # state the filter order
        filter_type = 'highpass'  # state the type of filter used
        test_input = [6, -8, 3]  # define the input signal coefficients

        hand_calculated_values = [1.2394, -4.5894, 6.6175]  # array that contains the hand calculated values for the
        # above specification

        filter_calculated_values = getCoefficients(cutoff_freq, filter_order, filter_type,
                                                   test_input)  # array contains the IIR filter output values for the
        # same specification

        self.assertEqual(hand_calculated_values,
                         filter_calculated_values)  # compare both arrays to check if they aer similar

    def test_highpass_sos_filter(self):
        cutoff_freq = 0.3  # define the normalized cutoff frequency
        filter_order = 2  # state the filter order
```

```
        filter_type = 'highpass'  # state the type of filter used
        test_input = [6, -8, 3]  # define the input signal coefficients

        hand_calculated_values = [1.2394, -4.5894, 6.6175]  # array that contains the hand calculated values for the
        # above specification

        filter_sos_calculated_values = getSOSCoefficients(cutoff_freq, filter_order, filter_type,
                                                              test_input)  # array contains the sos influenced IIR filter
        # output values for the same specification

        self.assertEqual(hand_calculated_values,
                         filter_sos_calculated_values)  # compare both arrays to check if they aer similar

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)
```

8.2.3 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.zeros(3)
    referenceFrame = []
    referenceFrame.append([1.0,0.0,0.0])
    referenceFrame.append([0.0,1.0,0.0])
    referenceFrame.append([0.0,0.0,1.0])
    for i in range(3):
        if DoDebug:
            print(f"unitVec{i} = {referenceFrame[i]}")
```

```python
            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)
```

8.2.4 realtime_iir_main.py

```python
""" 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
```

```python
fs = 650                # BF 3/01/22 - changed to "true" sample rate Max 1000
# Nyquist
fn = fs / 2
fc = 5

""" IIR Filter Design """
# Noise Removal
sos = signal.bessel(1, fc / fn, "lowpass", output="sos", norm="mag") # BF 3/01/22 - changed to bessel

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=[-0.25,0.25], channels=3)
orientation_plot = rtp.RealtimeVectorPlot()

""" Sample Process Function """
def addX(data):
    # Zeroing from measurements
    data -= 0.335
    f = x_filter.filter(data)

    sample_plot.addSample(data, f, channel=0)
    orientation_plot.addSample(f, channel=0)

def addY(data):
    # Zeroing from measurements
    data -= 0.340
    f = y_filter.filter(data)

    sample_plot.addSample(data, f, channel=1)
    orientation_plot.addSample(f, channel=1)

def addZ(data):
    # Zeroing from measurements
    data -= 0.340
    f = z_filter.filter(data)

    sample_plot.addSample(data, f, channel=2)
    orientation_plot.addSample(f, 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()
```

8.2.5 realtime_plot.py

```
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.animation import FuncAnimation
from matplotlib.widgets import Button
import matplotlib.pyplot as plt

import numpy as np
import time

from calcAngles 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
```

```python
        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([])
        self.ax.set_yticks([])
        self.ax.set_zticks([])

        self.anim = FuncAnimation(self.fig, self.update, interval=100)
        self.q = self.ax.quiver(0, 0, 0, 0, 0, 1)
        self.x = self.ax.quiver(0, 0, 0, 1, 0 ,0, color="red")
        self.y = self.ax.quiver(0, 0, 0, 0, 1 ,0, color="green")
        self.z = self.ax.quiver(0, 0, 0, 0, 0 ,1, color="blue")

        self.label = self.ax.text(0, 0, 0, "test", transform=self.ax.transAxes)
        self.update_delay = 0

        # For calibration
        self.offset_angles = np.zeros(3)
        axes = plt.axes([0.81, 0.05, 0.1, 0.075])
        self.button = Button(axes, "Calibrate")
        self.button.on_clicked(self.setOffset)

    def update(self, x):
        m = np.sqrt(self.vec[0] * self.vec[0] + self.vec[1] * self.vec[1] + self.vec[2] * self.vec[2])
        if m != 0:
            self.q.remove()
```

```
            self.q = self.ax.quiver(0, 0, 0, self.vec[0] / m, self.vec[1] / m, self.vec[2] / m, color="black")

            # Angle Calcs
            if self.update_delay % 5 == 0:
                # Angle between the vector and the x plane
                angle = calcAngles(self.vec)
                ma = f"Measured Angles (deg), x: {angle[0]:.1f}, y: {angle[1]:.1f}, z: {angle[2]:.1f}\n"
                angle = np.abs(angle - self.offset_angles)
                fa = f"Final Angle (deg):  {angle[2]:.1f}"
                self.label.set_text(ma + fa)
            self.update_delay += 1

    def setOffset(self, x):
        self.offset_angles = calcAngles(self.vec)

    def addSample(self, v, channel=0):
        self.vec[channel] = v
```