Alexander Dowell

ME 457 Checkpoint 1

1. Attitude Estimation

1.1. Conceptual Roll Estimation

Accelerometer:

The accelerometer on the auto pHAT mounted on the Raspberry Pi provides accelerometer data for three axes, in a north, east, down configuration. Switch or reverse axes so that they are oriented in the right direction. The angle between the y and z accelerometer vectors is the pitch angle. to find the pitch angle use this equation:

$$\theta_{acc\_roll} = \tan\left(\frac{acc_y}{acc_z}\right)$$

This estimate will be in radians so it must be multiplied by the conversion factor, $\frac{180}{\pi}$, to get θ_{acc_roll} in degrees.

gyroscope:

The gyroscope on the auto pHAT mounted on the Raspberry Pi provides gyroscope data for three axes, in a north, east, down configuration. Switch or reverse axes so that they are oriented in the right direction. The gyroscope provides angular velocity so to find angular position, roll angle, a numerical integration method must be used on the x-axis gyroscope data. Trapezoidal rule integration method:

$$\theta_{gyro\_roll} = \int g_x(t)dt = \frac{\Delta t}{2}\sum_{k=1}^{N}(g_x(t_{k-1}) + g_x(t_k))$$

The roll angle found from the gyroscope data needs to be divided by a scalar value to be accurate. To do this roll the Pi to a known angle then divide the estimate by the known angle. this will provide the scalar value.

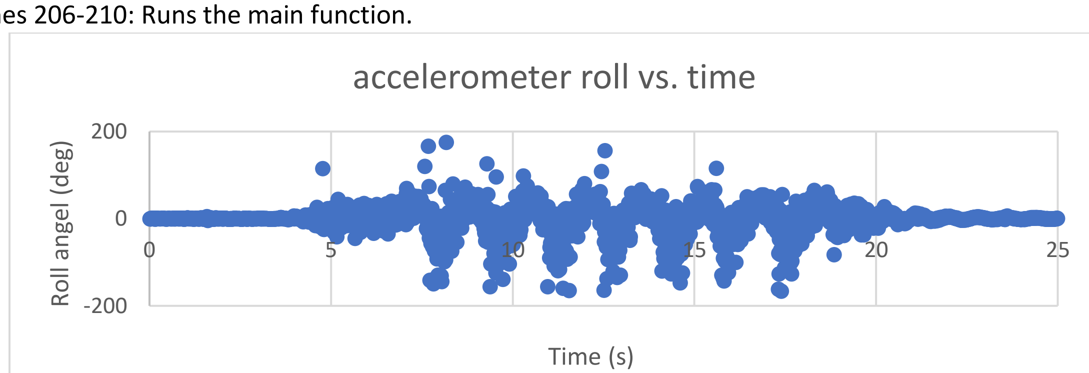
1.2. Practical Roll Estimation

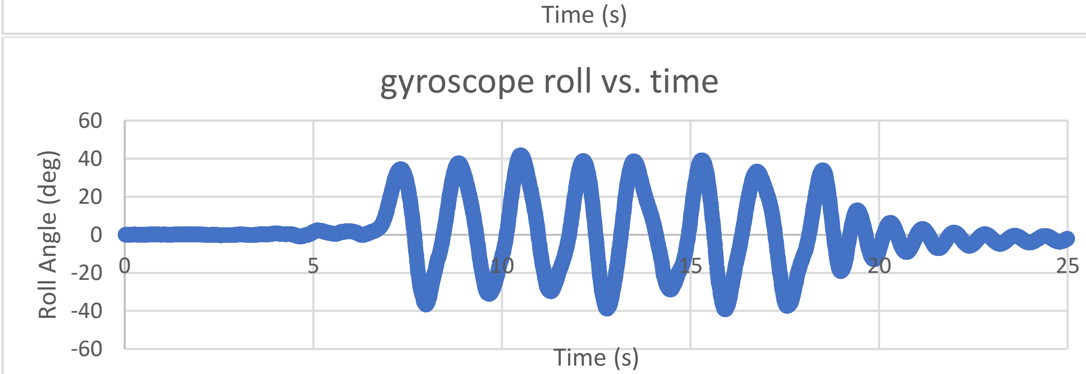
```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Alex dowell pitch and roll
"""
#import
import numpy as np
import rospy
import math
import time
import sys
import datetime
import os
import csv
from me457common.msg import IMU
from me457common.msg import Angular

#initialize input messages - IMU()
imu_data = IMU()
message = Angular()

# startup the logger node
rospy.init_node('angular', anonymous=True)

def imu_callback(message):
    global imu_data
    imu_data = message

# publisher
pub = rospy.Publisher('/angular', Angular, queue_size=1)

# set the output data rate for the log
log_odr = 100 # Hz

#---------Logfile Setup------------#
# populate the data header, these are just strings, you can name them anything
myData = ["time","accel_x","accel_y","accel_z","gyro_x","gyro_y","gyro_z","accel_pitch","accel_roll","gyro_pitch","gyro_roll","filtered_accel_pitch"
          ,"filtered_accel_roll","filtered_gyro_pitch","filtered_gyro_roll","comp_filt_pitch","comp_filt_roll"]
```

```python

# this creates a filename
# note that the path used here is an absolute path, if you want to put the log files somewhere else you will need
# to include an updated absolute path here to the new directory where you want the files to appear
fileNameBase = "/home/pi/catkin_ws/src/autophatros/scripts/logfiles/" + "alex_pitch_roll"
fileNameSuffix = ".csv"

# num is used for incrementing the file path if we already have a file in the directory with the same name
num = 1
fileName = fileNameBase + fileNameSuffix
# check if the file already exists and increment num until the name is unique
while os.path.isfile(fileName):
    fileName = fileNameBase + "_" + str(num) + fileNameSuffix
    num = num + 1

# now we know we have a unique name, let's open the file, 'a' is append mode, in the unlikely event that we open
# a file that already exists, this will simply add on to the end of it (rather than destroy or overwrite data)
myFile = open(fileName, 'a')
with myFile:
    writer = csv.writer(myFile)
    writer.writerow(myData)

zero_time = rospy.get_time()

def main():

    # establish the data rate according to the user parameter set above
    rate = rospy.Rate(log_odr)

    # initailize variables
    then = 0
    gyroy_old = 0
    gyrox_old = 0
    gyro_pitch = 0
    gyro_roll = 0
    filtered_gyroy_old = 0
    filtered_gyrox_old = 0
    filtered_gyro_pitch = 0
    filtered_gyro_roll = 0
    acc_pitch0 = 0
    acc_pitch1 = 0
    f_acc_pitch1 = 0
    f_acc_pitch0 = 0
    acc_roll0 = 0
    acc_roll1 = 0
    f_acc_roll1 = 0
    f_acc_roll0 = 0
    gyro_pitch0 = 0
    gyro_pitch1 = 0
    f_gyro_pitch0 = 0
    f_gyro_pitch1 = 0
    gyro_roll0 = 0
    gyro_roll1 = 0
    f_gyro_roll0 = 0
    f_gyro_roll1 = 0

    while not rospy.is_shutdown():

        rospy.Subscriber('imupub', IMU, imu_callback)

        # get the current time and subtract off the zero_time offset
        now = (rospy.get_time()-zero_time)

        # radians to degrees converter
        con = 180/math.pi
        # (axis orientation)negative signs are specific to pi 192.168.1.151
        ax = imu_data.accelerometer.x
        ay = imu_data.accelerometer.y
        az = imu_data.accelerometer.z
        gx = imu_data.gyroscope.x
        gy = -imu_data.gyroscope.y
        gz = -imu_data.gyroscope.z
        #pitch from the acceleromter
        acc_pitch = math.atan2(ax,az)*con

        #roll from the accelerometer
        acc roll = math.atan2(ay,az)*con
        # pitch from the gyroscope
        gyroy = gy
        gyro_pitch_raw = ((gyroy + gyroy_old)/2)*(now-then)*con
        gyro_pitch_new = gyro_pitch_raw/7515 #scalar value is specific pi 151
        gyroy_old = gyroy
        gyro_pitch = gyro_pitch + gyro_pitch_new

        # roll from the gyroscope
        gyrox = gx
        gyro_roll_raw = ((gyrox + gyrox_old)/2)*(now-then)*con
        gyro_roll_new = gyro_roll_raw/7515 #scalar value is specific pi 151
        gyrox_old = gyrox
        gyro_roll = gyro_roll + gyro_roll_new
        then = now
        myData = [now, ax, ay, az, gx, gy, gz, acc_pitch, acc_roll, gyro_pitch, gyro_roll,filtered_acc_pitch, filtered_acc_roll, filtered_gyro_pitch
                , filtered_gyro_roll, com_pitch, com_roll]

        # stick everything in the file
        myFile = open(fileName, 'a')
        with myFile:
            writer = csv.writer(myFile)
            writer.writerow(myData)

        # wait for a calculated amount of time before doing it all over again
        rate.sleep()
        message.pitch = com_pitch
        message.roll = com_roll
        message.yaw = 0

        pub.publish(message)

#run the main function
if __name__ == '__main__':
    try:
        main()
    except rospy.ROSInterruptException:
        pass
```

Alexander Dowell

The above python script takes in data from the gyroscope and accelerometer and converts it to pitch and roll angles.
Lines 1-20: Initiates this script and imports all the functions and sensor data that are needed. Then assigns the sensor data array to variable imu_data and assign the pitch, roll, and yaw angle array to variable message which will later be assigned filtered estimates.
Lines 23-30: Initiates a node, creating a call back function to make imu_data a global variable, and creating a pubisher so the angle array can be published.
Lines 33-37: Setting the sampling frequency to 100 hertz. The myData array is populated with header labels.
Lines 43-59: This section creates an excel file named pitch_roll and populates it with the myData.
Lines 61-63: First initializing the time variable, zero_time, then the main function that processes the sensor data and outputs pitch and yaw estimates is defined.
Lines 66-95: Establishes the data rate according to what it was set to in line 33. initializing all the iterative variables outside of the loop. Start while loop that will calculate pitch and roll angle then filter the signal to remove noise.
Lines 97-110: First creating a subscriber so IMU data from the sensors can used. A Now variable is initialized and the time step between values is the sampling rate set in line 33. A conversion factor is created. This conversion factor changes radian to degrees and must be applied to the accelerometer pitch and roll data if output in degrees is desired. Last is a set of variables that the raw sensor data are assigned to. The gyro y and z were in the wrong direction, so a negative sign was appliedo to reverse the axes
Lines 128 -131: Pitch and roll estimates are calculated from the accelerometer.
Lines 149-161: Pitch and roll estimates are calculated from the gyroscope.
Lines 188-195: myData is populated with current iteration values from the IMU and calculated pitch and roll values. Then current myData is put in the file pitch_roll.
Lines 198-203: rate.sleep() waits the remaining amount of time, set by the sampling frequency to begin the next iteration. Filtered pitch and roll estimates are assigned to message variable. Message variable is published
Lines 206-210: Runs the main function.





The roll angle from the accelerometer is not useful because there is too much noise. The roll angle from the gyroscope is not useful at low speeds because it is subject to drift. Filters will solve the issues.

2. Digital Filtering

2.1. Conceptual Digital Filtering

Digital filters remove certain frequencies from a signal. There are two types, Infinite impulse response (IIR) and Finite impulse response (FIR). IIR filters are for applications where linear phase is not very important and memory is limited. FIR filters are for applications where linear phase is important and good memory and computational power are available. Second order Butterworth high pass and low pass filters IIR are going to be used. Low pass filters attenuate frequencies above the cut off frequency. High pass filters attenuate frequencies below the cut off frequency. Looking at a FFT of the roll data from the gyroscope and accelerometer will identify where an acceptable cut off frequency should be. To execute a second order butterworth low and high pass filter data needs to be processed through this set of equations. The input being the data x, the sampling frequency f_s, and the cut off frequency f_c. The output being filtered data y. The set of equations:

$\gamma = \tan(\pi \times \frac{f_c}{f_s})$ $\quad D = \gamma^2 + \sqrt{2}\gamma + 1$ $\quad a_1 = 2 \times \frac{\gamma^2 - 1}{D}$ $\quad a_2 = \frac{\gamma^2 - \sqrt{2}\gamma + 1}{D}$

For low pass:

$b_0 = \frac{\gamma^2}{D}$ $\quad b_1 = 2b_0$ $\quad b_2 = b_0$

For high pass:

$b_0 = \frac{1}{D}$ $\quad b_1 = \frac{-2}{D}$ $\quad b_2 = b_0$

The filter:

$$y_n = b_0 x_n + b_1 x_{n-1} + b_2 x_{n-2} - a_1 y_{n-1} - a_2 y_{n-2}$$

2.2. Implementing a digital filter in Python

After the script has calculated pitch and roll estimates from the gyroscope and accelerometer in the while loop the following code will filter the data set.

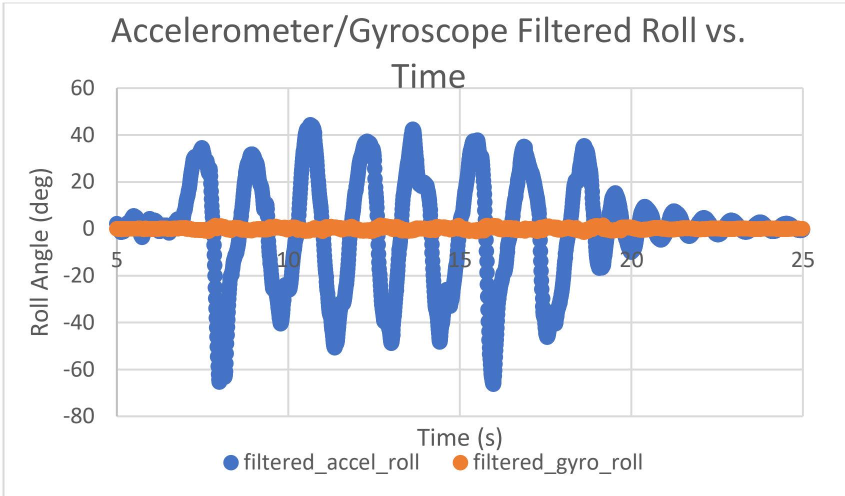
```
109        fc = 5
110        fs=100
111        alpha = math.tan(math.pi*(fc/fs))
112        D = (alpha**2) + (math.sqrt(2)*alpha) + 1
113        b0l = (alpha**2)/D
114        b1l = 2*b0l
115        b2l = b0l
116        b0h = 1/D
117        b1h = -2/D
118        b2h = b0h
119        a1 = 2 *(((alpha**2)-1)/D)
120        a2 = ((alpha**2) - (math.sqrt(2)*alpha) + 1)/D
130        #lowpass pitch from accelerometer
131        filtered_acc_pitch = (b0l*acc_pitch) + (b1l*acc_pitch1) + (b2l*acc_pitch0) - (a1*f_acc_pitch1) - (a2*f_acc_pitch0)
132        acc_pitch0 = acc_pitch1
133        acc_pitch1 = acc_pitch
134        f_acc_pitch0 = f_acc_pitch1
135        f_acc_pitch1 = filtered_acc_pitch
136
137        #lowpass roll from accelerometer
138        filtered_acc_roll = (b0l*acc_roll) + (b1l*acc_roll1) + (b2l*acc_roll0) - (a1*f_acc_roll1) - (a2*f_acc_roll0)
139        acc_roll0 = acc_roll1
140        acc_roll1 = acc_roll
141        f_acc_roll0 = f_acc_roll1
142        f_acc_roll1 = filtered_acc_roll
159        #highpass pitch from gyroscope
160        filtered_gyro_pitch = (b0h*gyro_pitch) + (b1h*gyro_pitch1) + (b2h*gyro_pitch0) - (a1*f_gyro_pitch1) - (a2*f_gyro_pitch0)
161        gyro_pitch0 = gyro_pitch1
162        gyro_pitch1 = gyro_pitch
163        f_gyro_pitch0 = f_gyro_pitch1
164        f_gyro_pitch1 = filtered_gyro_pitch
165
166        #highpass pitch from gyroscope
167        filtered_gyro_roll = (b0h*gyro_roll) + (b1h*gyro_roll1) + (b2h*gyro_roll0) - (a1*f_gyro_roll1) - (a2*f_gyro_roll0)
168        gyro_roll0 = gyro_roll1
169        gyro_roll1 = gyro_roll
170        f_gyro_roll0 = f_gyro_roll1
171        f_gyro_roll1 = filtered_gyro_roll
```

Lines 109-120: Define the variables according to the sampling frequency and cut off frequency.

Lines 130-142: Here the pitch and roll from the accelerometer runs through the low pass filter equation and reassigns values to make room for the current index in the next iteration.

Lines 159-171: Here the pitch and roll from the gyroscope runs through the high pass filter equation and reassigns values to make room for the current index in the next iteration.

Using a cut off frequency of 5 hertz and a sampling frequency of 100 hertz:



Roll from the accelerometer is clear. The lowpass filter attenuated most of the frequencies higher than the cut off frequency. The Roll from the gyroscope no longer has drift. The high pass filter attenuated most of the frequencies lower than the cut off frequency.

3. Sensor Fusion

3.1. Complimentary Filter Theory

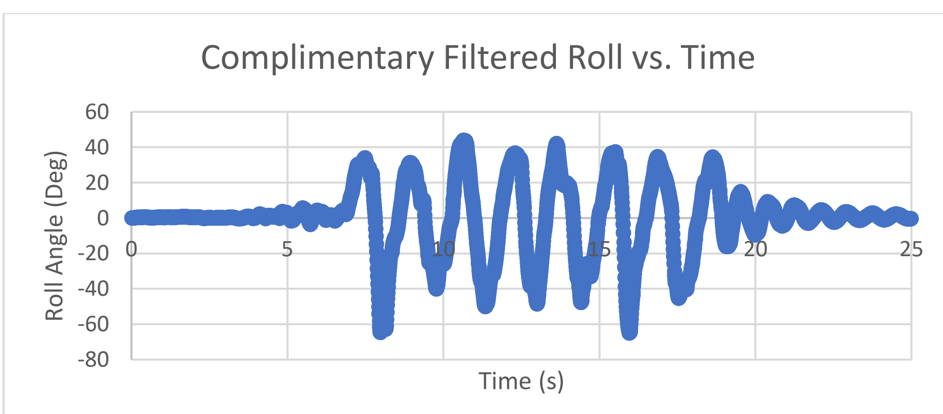
The accelerometer works well at low frequencies and the gyroscope works well at high frequencies. Using a low pass filter on the accelerometer and a high pass filter on the gyroscope both using the same cut off frequency that is located between the accelerometer and gyroscope spikes on the FFT. The sum of both filters gives filtered data across the whole frequencies range. The data above was at low frequency. That is why the low pass filtered pitch from the accelerometer is picking it up. If the frequency was above 5 hertz the accelerometer would be about zero and the high pass filtered roll from the gyroscope would be picking it up. The equation is:

$$low\ pass\ accel\ roll\ f_c\ @\ 5\ htz + high\ pass\ gyro\ roll\ f_c\ @\ 5\ htz$$
$$= filtered\ data\ across\ all\ frequencies$$

3.2. Implementing a Complimentary Filter

```
174        com_pitch = filtered_acc_pitch + filtered_gyro_pitch
175
176         # complementary roll filter
177        com_roll = filtered_acc_roll + filtered_gyro_roll
```

Lines 174-177: Now that there is a filtered roll data from the gyroscope and the accelerometer they can be summed to give the complementary filter.



Here the complimentary filter gives accurate roll estimates and takes advantage of both sensors. The accelerometer for the lower frequencies and the gyroscope for the higher frequencies.

Alexander Dowell