GC Tailbone Analysis

August 2, 2018

```
# %load main.py
#
# filename: main.py
# By: Abhay Gupta
#
# Description: Reconstruct GC's tailbone analysis in python
#

import sys
import os

p = os.path.abspath('../../../Data/Parsing Program/')
sys.path.append(p)

import print_struct as visual
import pickle
import csv
import statistics as stats
import math
import filt
import matplotlib.pyplot as plt
import integrate_IMU as integrate
import numpy as np
from numpy.linalg import inv
import stride
import difference

pickle_file = 'no_001.pkl'

with open(pickle_file, 'rb') as afile:
        data = pickle.load(afile)

# Extract only tailbone acceleration data
accel_data = data['UR']['sensorData']['tailBone']['accel']['data']

# Remove initial scrap data
accel_data['x'] = accel_data['x'][4:]
accel_data['y'] = accel_data['y'][4:]
accel_data['z'] = accel_data['z'][4:]

# Extract only tailbone gyroscope data
gyro_data = data['UR']['sensorData']['tailBone']['gyro']['data']
```

```
# Remove initial scrap data
gyro_data['x'] = gyro_data['x'][4:]
gyro_data['y'] = gyro_data['y'][4:]
gyro_data['z'] = gyro_data['z'][4:]

#Find the relative gravity vector from data
# 250->450 is the data that is relatively flat... (no movement)
mean_accel = {}
mean_accel['x'] = stats.mean(accel_data['x'][249:450])
mean_accel['y'] = stats.mean(accel_data['y'][249:450])
mean_accel['z'] = stats.mean(accel_data['z'][249:450])

gravity_strength = math.sqrt(mean_accel['x']**2 + mean_accel['y']**2 + \
                mean_accel['z']**2)

normalized_mean_accel = {}
normalized_mean_accel['x'] = mean_accel['x']/gravity_strength
normalized_mean_accel['y'] = mean_accel['y']/gravity_strength
normalized_mean_accel['z'] = mean_accel['z']/gravity_strength

"""NU"""
normalized_gravity_vector = [normalized_mean_accel['x'], normalized_mean_accel['y'], \
                normalized_mean_accel['z']]

# Key asumption here is that the gravity vector is [-1, 0, 0]
# so the x axis is upwards and downwards
# R_yzx
roll = 0; # math.atan(mean_accel['y']/mean_accel['z'])
pitch = math.atan(normalized_mean_accel['z']/normalized_mean_accel['x'])
yaw = math.atan(normalized_mean_accel['y']/math.sqrt(normalized_mean_accel['x']**2 + \
                normalized_mean_accel['z']**2))

# Indexes used for analysis
start_indices = [538, 1324, 2149, 2959];
end_indices = [1170, 2020, 2730, 3350];
start_index = start_indices[0]
end_index = min(len(accel_data['seconds']), len(gyro_data['seconds']))-5

# Cut data to relevant data
accel = {}
accel['sec'] = accel_data['seconds'][start_index:(end_index)]
accel['x'] = accel_data['x'][start_index:(end_index)]
accel['y'] = accel_data['y'][start_index:(end_index)]
accel['z'] = accel_data['z'][start_index:(end_index)]
time = accel['sec']

# Put acceleration data through a low pass filter
# No gravity vector utilized yet.... don't know why it is called it yet...
f_cuts = [0.1, 0.7] # Passband & Stopband cutoffs
sampling_rate = 100
ripple_tol = [0.001, 0.1] # Passband & Stopband tolerances
```

```
gravity_accel = {}
gravity_accel['x'],_,_,_ = filt.lowpass(accel_data['x'], f_cuts, sampling_rate, \
                ripple_tol);
gravity_accel['y'],_,_,_ = filt.lowpass(accel_data['y'], f_cuts, sampling_rate, \
                ripple_tol);
gravity_accel['z'],_,_,_ = filt.lowpass(accel_data['z'], f_cuts, sampling_rate, \
                ripple_tol);

#plt.plot(gravity_accel['x'])
#plt.show()

# Change range of data....
gravity_accel['x'] = gravity_accel['x'][start_index:end_index]
gravity_accel['y'] = gravity_accel['y'][start_index:end_index]
gravity_accel['z'] = gravity_accel['z'][start_index:end_index]

gyro = {}
gyro['sec'] = gyro_data['seconds'][start_index:end_index]
gyro['x'] = gyro_data['x'][start_index:end_index]
gyro['y'] = gyro_data['y'][start_index:end_index]
gyro['z'] = gyro_data['z'][start_index:end_index]

# Labeled as high pass... but not high passed...
gyro_hpf = {}
gyro_hpf['x'] = gyro['x']
gyro_hpf['y'] = gyro['y']
gyro_hpf['z'] = gyro['z']


# Find the angular position of sensors
angular_pos = {}
angular_pos['x'] = integrate.IMU(gyro['sec'], gyro_hpf['x'])
angular_pos['x'] = [math.pi/180*x for x in angular_pos['x']]
angular_pos['y'] = integrate.IMU(gyro['sec'], gyro_hpf['y'])
angular_pos['y'] = [math.pi/180*x for x in angular_pos['y']]
angular_pos['z'] = integrate.IMU(gyro['sec'], gyro_hpf['z'])
angular_pos['z'] = [math.pi/180*x for x in angular_pos['z']]

# Gravity vector...
# A normalized gravity vector was made earlier.... not used...., but this is used...
gravity_vector = mean_accel
gravity_vector_holder = np.empty([3,len(accel['sec'])])

accel_delta = {}
accel_delta['x'] = [0] * len(accel['sec'])
accel_delta['y'] = [0] * len(accel['sec'])
accel_delta['z'] = [0] * len(accel['sec'])
accel_delta_vector_holder = np.empty([3,len(accel['sec'])])

accel2_delta = {}
accel2_delta['x'] = [0] * len(accel['sec'])
accel2_delta['y'] = [0] * len(accel['sec'])
accel2_delta['z'] = [0] * len(accel['sec'])
```

```
accel2_delta_vector_holder = np.empty([3,len(accel['sec'])])

earth_accel_delta_vector_holder = np.empty([3,len(accel['sec'])])
earth2_accel_delta_vector_holder = np.empty([3,len(accel['sec'])])
forward_earth_accel_delta_vector_holder = np.empty([3,len(accel['sec'])])
forward_position_delta_vector_holder = np.empty([3,len(accel['sec'])])

# Main function?
#plt.plot(gravity_accel['x'])
#plt.plot(accel['x'])

for i in range(0,len(accel['sec'])):

        # Calculate rotation
        # These are the basic rotational matrixes from linar algebra
        # Roll, pitch and yaw respectively

        rotation = {}

        rotation['x'] = np.empty([3,3])
        rotation['x'][0] = [1, 0, 0]
        rotation['x'][1] = [0, math.cos(angular_pos['x'][i]), \
                        math.sin(angular_pos['x'][i])]
        rotation['x'][2] = [0, -1*math.sin(angular_pos['x'][i]), \
                        math.cos(angular_pos['x'][i])]

        rotation['y'] = np.empty([3,3])
        rotation['y'][0] = [math.cos(angular_pos['y'][i]), 0, \
                        -1*math.sin(angular_pos['y'][i])]
        rotation['y'][1] = [0, 1, 0]
        rotation['y'][2] = [math.sin(angular_pos['y'][i]), 0, \
                        math.cos(angular_pos['y'][i])]

        rotation['z'] = np.empty([3,3])
        rotation['z'][0] = \
                        [math.cos(angular_pos['z'][i]), math.sin(angular_pos['z'][i]), 0]
        rotation['z'][1] = \
                        [-1*math.sin(angular_pos['z'][i]), math.cos(angular_pos['z'][i]), 0]
        rotation['z'][2] = [0, 0, 1]

        # Why is it multiplied y to z to x? wiki says this is the order: z, y, x
        # This order is due to where the gravity
        # Matrix multiplication is not communitative
        # Gravity vector is the constant gravity on acceleromter (3x1)

        g_v = np.asarray(list(gravity_vector.values()))
        gravity_vector_holder[:,i] = rotation['y'] @ rotation['z'] @ rotation['x'] @ g_v

        # Difference between the measured acceleration and the direction of gravity
        # Should be actual acceleration value :)
        accel2_delta['x'][i] = accel['x'][i] - gravity_vector_holder[0,i]
        accel2_delta['y'][i] = accel['y'][i] - gravity_vector_holder[1,i]
        accel2_delta['z'][i] = accel['z'][i] - gravity_vector_holder[2,i]
```

```
accel2_delta_vector_holder[:,i] = \
                [accel2_delta['x'][i], accel2_delta['y'][i], accel2_delta['z'][i]]


# Rotation with removed initial gravitational vector...
updated_rotation = {}

updated_rotation['x'] = np.empty([3,3])
updated_rotation['x'][0] = [1, 0, 0]
updated_rotation['x'][1] = \
                [0, math.cos(angular_pos['x'][i]), math.sin(angular_pos['x'][i])]
updated_rotation['x'][2] = \
                [0, -1*math.sin(angular_pos['x'][i]), math.cos(angular_pos['x'][i])]

updated_rotation['y'] = np.empty([3,3])
updated_rotation['y'][0] = [math.cos(angular_pos['y'][i]+pitch), 0, \
                -1*math.sin(angular_pos['y'][i]+pitch)]
updated_rotation['y'][1] = [0, 1, 0]
updated_rotation['y'][2] = [math.sin(angular_pos['y'][i]+pitch), 0, \
                math.cos(angular_pos['y'][i]+pitch)]

updated_rotation['z'] = np.empty([3,3])
updated_rotation['z'][0] = [math.cos(angular_pos['z'][i]+yaw), \
                math.sin(angular_pos['z'][i]+yaw), 0]
updated_rotation['z'][1] = [-1*math.sin(angular_pos['z'][i]+yaw), \
                math.cos(angular_pos['z'][i]+yaw), 0]
updated_rotation['z'][2] = [0, 0, 1]

a = inv(updated_rotation['y'] @ updated_rotation['z'] @ updated_rotation['x'])

earth2_accel_delta_vector_holder[:,i] = a @ accel2_delta_vector_holder[:,i]

# Acceleration - Low passed acceleration
# Output: Noise from accelerometer
accel_delta['x'][i] = accel['x'][i] - gravity_accel['x'][i]
accel_delta['y'][i] = accel['y'][i] - gravity_accel['y'][i]
accel_delta['z'][i] = accel['z'][i] - gravity_accel['z'][i]
accel_delta_vector_holder[:,i] = \
                [accel_delta['x'][i], accel_delta['y'][i], accel_delta['z'][i]]

updated_gravity_strength = math.sqrt(gravity_accel['x'][i]**2 + \
                gravity_accel['y'][i]**2 + gravity_accel['z'][i]**2)

# Normalizing the low passed acceleration...
# This doesn't really make sense...
# you are scaling each acceleration movement, but...
# this means that there will be the same acceleration each entry... not true...

updated_normal_mean_accel = {}
updated_normal_mean_accel['x'] = gravity_accel['x'][i]/updated_gravity_strength
updated_normal_mean_accel['y'] = gravity_accel['y'][i]/updated_gravity_strength
updated_normal_mean_accel['z'] = gravity_accel['z'][i]/updated_gravity_strength
```

```
# Update pitch and yaw with low-passed data
new_ang = {}
new_ang['y'] = math.atan(updated_normal_mean_accel['z']/\
                updated_normal_mean_accel['x']) # updated pitch
new_ang['z'] = math.atan(updated_normal_mean_accel['y']/math.sqrt(\
                updated_normal_mean_accel['x']**2 + updated_normal_mean_accel['z']**2))
# updated yaw

# rotate so that the X axis points towards the earth
# weird because only pitch and yaw rotation
# was not used for initial acceleration data, only LP accel

updated_rotation['y'][0] = [math.cos(new_ang['y']), 0, -1*math.sin(new_ang['y'])]
updated_rotation['y'][1] = [0, 1, 0]
updated_rotation['y'][2] = [math.sin(new_ang['y']), 0, math.cos(new_ang['y'])]

updated_rotation['z'][0] = [math.cos(new_ang['z']), math.sin(new_ang['z']), 0]
updated_rotation['z'][1] = [-1*math.sin(new_ang['z']), math.cos(new_ang['z']), 0]
updated_rotation['z'][2] = [0, 0, 1]

updated_rotation['x'] = np.identity(3)

# I have NO CLUE why we inverted the matrix multiplication
a = inv(updated_rotation['y'] @ updated_rotation['z'] @ updated_rotation['x'])
earth_accel_delta_vector_holder[:,i] = a @ accel_delta_vector_holder[:,i]

# rotate so that Z axis is direction of travel?

if (i>1):
        j = i + 1

        # find position of non-filtered data...
        # double integration to get position
        coordinates = ['x', 'y', 'z']
        earth_position_temp = {}
        _, earth_position_temp['x'] = integrate.double(accel['sec'][:j], \
                        earth_accel_delta_vector_holder[0,:j])
        earth_position_temp['y'] = integrate.IMU(accel['sec'][:j],\
                        integrate.IMU(accel['sec'][:j],earth_accel_delta_vector_holder[1,:j]))
        earth_position_temp['z'] = integrate.IMU(accel['sec'][:j],\
                        integrate.IMU(accel['sec'][:j],earth_accel_delta_vector_holder[2,:j]))

        position_strength = math.sqrt(earth_position_temp['x'][-1]**2 + \
                        earth_position_temp['y'][-1]**2 + earth_position_temp['z'][-1]**2)

        # Normalize the distance moved? I don't see the point of this...
        normalized_mean_position = {}
        normalized_mean_position['x'] = earth_position_temp['x'][-1]/position_strength
        normalized_mean_position['y'] = earth_position_temp['y'][-1]/position_strength
        normalized_mean_position['z'] = earth_position_temp['z'][-1]/position_strength

        # Where did this come from?
        roll = math.atan(normalized_mean_position['y']/normalized_mean_position['z'])
```

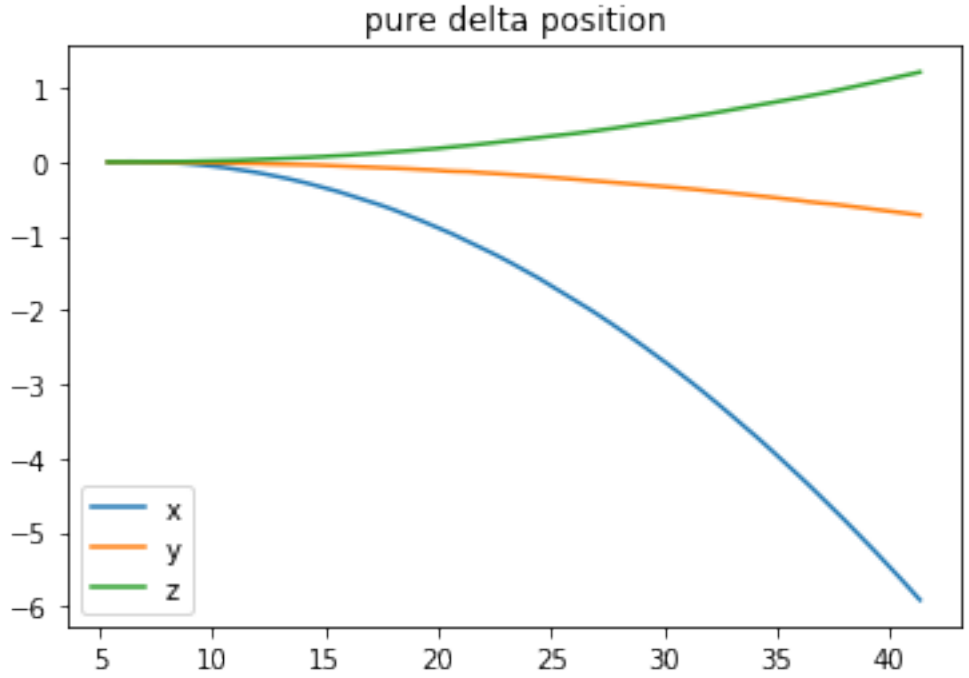
```
            pitch = math.atan(-1*normalized_mean_position['x']/math.sqrt(\
                            normalized_mean_position['y']**2 + normalized_mean_position['z']**2))

            updated_rotation['x'][0] = [1, 0, 0]
            updated_rotation['x'][1] = [0, math.cos(roll), math.sin(roll)]
            updated_rotation['x'][2] = [0, -1*math.sin(roll), math.cos(roll)]

            updated_rotation['y'][0] = [math.cos(pitch), 0, -1*math.sin(pitch)]
            updated_rotation['y'][1] = [0, 1, 0]
            updated_rotation['y'][2] = [math.sin(pitch), 0, math.cos(pitch)]

            updated_rotation['z'] = np.identity(3)

            a = inv(updated_rotation['x'] @ updated_rotation['y'] @ updated_rotation['z'])

            forward_earth_accel_delta_vector_holder[:,i] = a @ \
                            earth_accel_delta_vector_holder[:,i]
            forward_position_delta_vector_holder[:,i] = a @[earth_position_temp['x'][-1],\
                            earth_position_temp['y'][-1], earth_position_temp['z'][-1]]

""" NU START """
# The hpf is only a subtraction of 0.18?
accel_pure_delta_Hpf = {}
accel_pure_delta_Hpf['z'] = [x - 0.18 for x in accel['z']]

# Calculate the velocity and the position through integration
pure_velocity = {}
pure_velocity['z'] = integrate.IMU(accel['sec'], accel_pure_delta_Hpf['z'])

pure_position = {}
pure_position['z'] = integrate.IMU(accel['sec'], pure_velocity['z'])
""" NU END """

# Integrating the noise to get velocity and position noise
pure_delta_velocity, pure_delta_position = integrate.double(accel['sec'], accel_delta)

# hpf is same as no filter
# Storing Noise?
accel_delta_hpf = {}
accel_delta_hpf['x'] = accel_delta['x']
accel_delta_hpf['y'] = accel_delta['y']
accel_delta_hpf['z'] = accel_delta['z']

#plt.plot(accel_delta['x'])
#plt.show()

# I double checked... it is noise... wtf
# Anyways, integrate to get velocity and acceleration
velocity, position = integrate.double(accel['sec'], accel_delta_hpf)

# hpf is same as no filter...
# Gravity vector MM with noise...
earth_accel_delta_hpf = {}
```

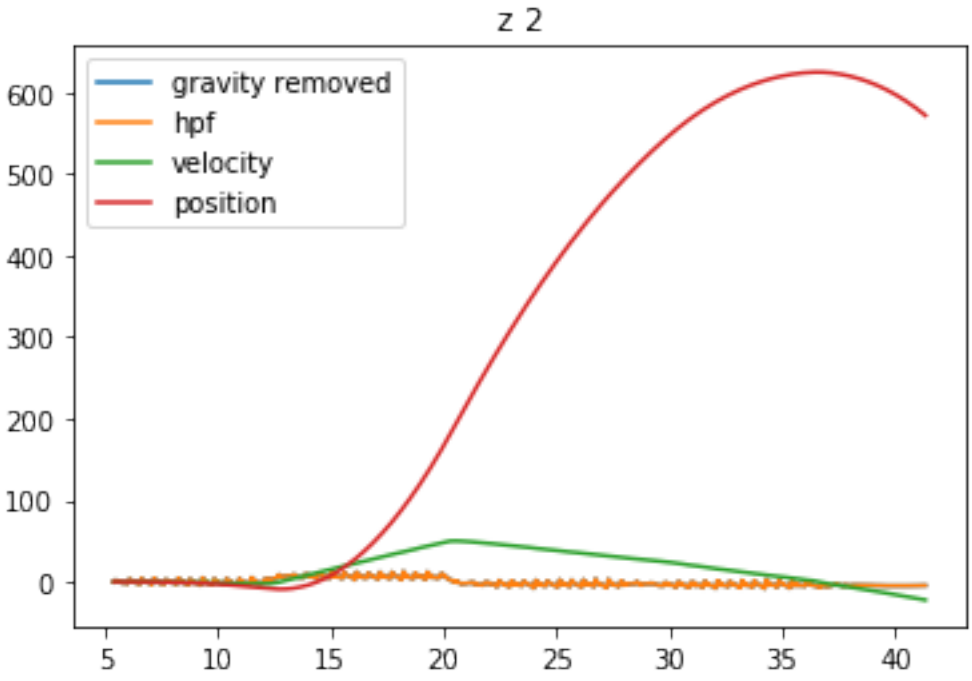
```
earth_accel_delta_hpf['x'] = earth_accel_delta_vector_holder[0,:]
earth_accel_delta_hpf['y'] = earth_accel_delta_vector_holder[1,:]
earth_accel_delta_hpf['z'] = earth_accel_delta_vector_holder[2,:]


# Find the position and velocity of noise MM gravity vector
earth_velocity, earth_position = integrate.double(accel['sec'], earth_accel_delta_hpf)
earth_velocity['z'] = [4*x for x in earth_velocity['z']]
earth_position['z'] = [4*x for x in earth_position['z']]

# Why print this variable?
forward_earth_accel_delta_vector_holder[:, i]

# Different gravity vector alignment...
# x-axis aligned with gravity MM with noise
# Multiply each vector
forward_earth_accel_delta_hpf = {}
forward_earth_accel_delta_hpf['x'] = forward_earth_accel_delta_vector_holder[0,:]
forward_earth_accel_delta_hpf['y'] = forward_earth_accel_delta_vector_holder[1,:]
forward_earth_accel_delta_hpf['z'] = forward_earth_accel_delta_vector_holder[2,:]

# get velocity
forward_earth_velocity = integrate.single(accel['sec'], forward_earth_accel_delta_hpf)
# multiply it by 4? wtf
forward_earth_velocity['z'] = [4*x for x in forward_earth_velocity['z']]
# get position
forward_earth_position = integrate.single(accel['sec'], forward_earth_velocity)

# hpf is same as regular again...
# Difference between measured acceleration and gravity
# This should be actual values :)
accel2_delta_hpf = accel2_delta.copy()
# get the velocity and position
velocity2, position2 = integrate.double(accel['sec'], accel2_delta_hpf)

# same as previous, except added rotation from initial gravitational vector throu MM
coordinates = ['x', 'y', 'z']

earth2_accel_delta_hpf = {}
for c,p in enumerate(coordinates):
        earth2_accel_delta_hpf[p] = earth2_accel_delta_vector_holder[c,:]

earth2_velocity, earth2_position = integrate.double(accel['sec'], \
                earth2_accel_delta_hpf)

# figures of all data
legend1 = ['gravity removed', 'hpf', 'velocity', 'position']

# figure 1
plt.figure()
for axis in coordinates:
        plt.plot(time, [9.81*x for x in pure_delta_position[axis]])
plt.title('pure delta position')
```

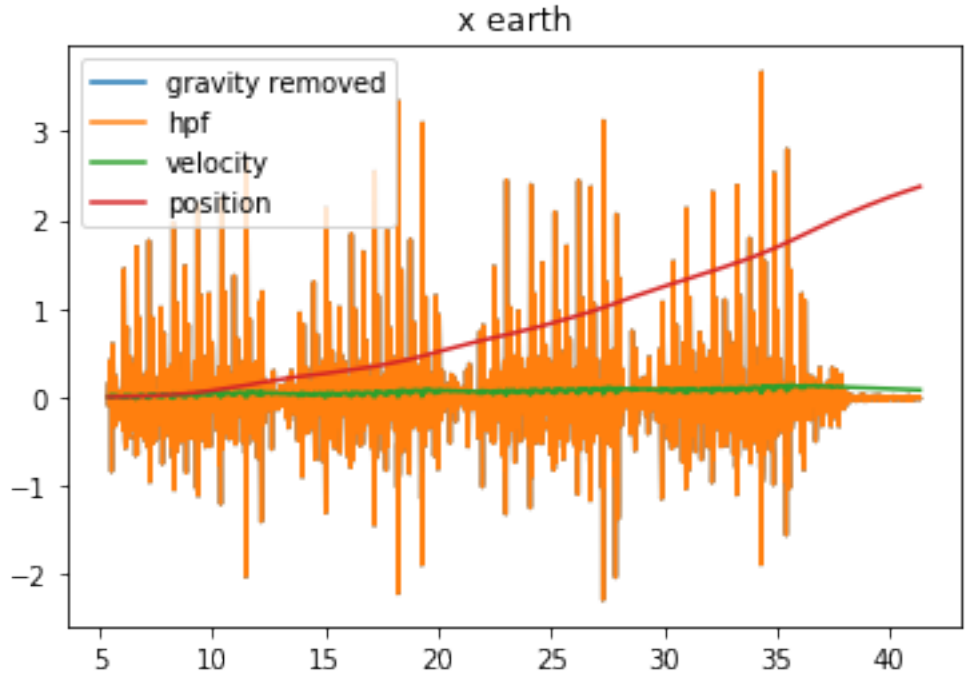
```
plt.legend(coordinates)

# figure 2
plt.figure()
plt.plot(time, [9.81*x for x in accel2_delta['z']])
plt.plot(time, [9.81*x for x in accel2_delta_hpf['z']])
plt.plot(time, [9.81*x for x in velocity2['z']])
plt.plot(time, [9.81*x for x in position2['z']])
plt.title('z 2')
plt.legend(legend1)

# figures 3-5
for c, axis in enumerate(coordinates):
        plt.figure()
        plt.plot(time, [9.81*x for x in earth_accel_delta_vector_holder[c,:]])
        plt.plot(time, [9.81*x for x in earth_accel_delta_hpf[axis]])
        plt.plot(time, [9.81*x for x in earth_velocity[axis]])
        plt.plot(time, [9.81*x for x in earth_position[axis]])
        plt.title(axis + ' earth')
        plt.legend(legend1)

# figures 6-8
for c, axis in enumerate(coordinates):
        plt.figure()
        plt.plot(time, [9.81*x for x in forward_earth_accel_delta_vector_holder[c,:]])
        plt.plot(time, [9.81*x for x in forward_earth_accel_delta_hpf[axis]])
        plt.plot(time, [9.81*x for x in forward_earth_velocity[axis]])
        plt.plot(time, [9.81*x for x in forward_earth_position[axis]])
        plt.title(axis + ' forward earth')
        plt.legend(legend1)

# figures 9-11
for c, axis in enumerate(coordinates):
        plt.figure()
        plt.plot(time, [9.81*x for x in earth2_accel_delta_vector_holder[c,:]])
        plt.plot(time, [9.81*x for x in earth2_accel_delta_hpf[axis]])
        plt.plot(time, [9.81*x for x in earth2_velocity[axis]])
        plt.plot(time, [9.81*x for x in earth2_position[axis]])
        plt.title(axis + ' earth 2')
        plt.legend(legend1)


# GC somehow initilizes HS parameters....
HS = {}
stride_velocity = {}
mean_stride_velocity = {}
for i in range(0,5):
        HS[i] = {}
        stride_velocity[i] = {}
        mean_stride_velocity[i] = {}

HS[0]['r'] = [6.17, 7.27, 8.3500, 9.41, 10.48,  11.6]
HS[0]['l'] = [5.55,  6.7, 7.7900, 8.85,  9.92, 11.02]
```

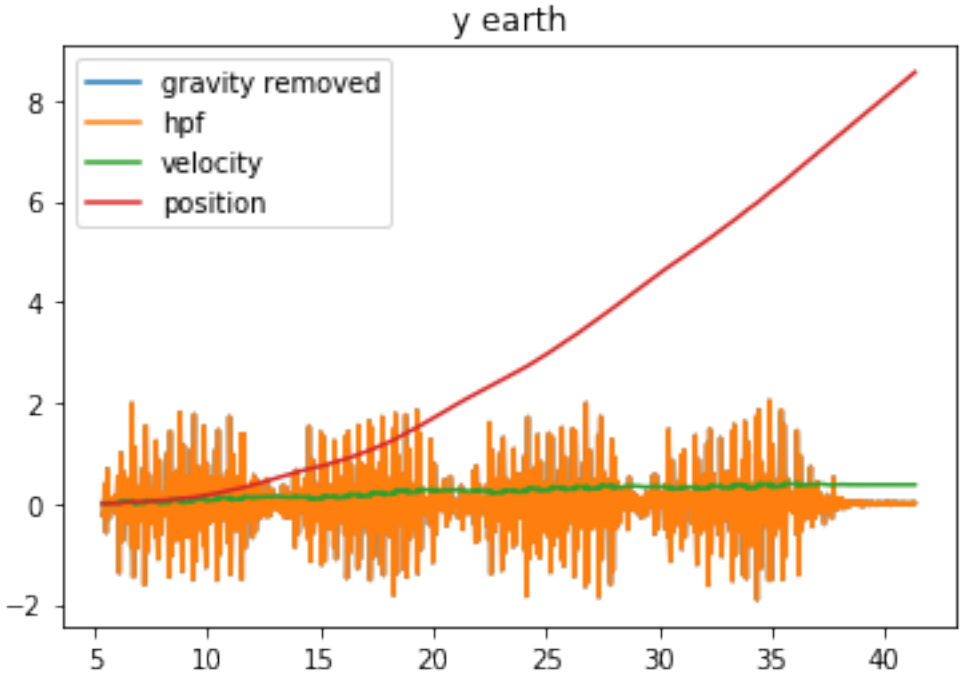
```
HS[1]['r'] = [6.13, 7.22, 8.3100, 9.37, 10.45,  11.56]
HS[1]['l'] = [5.55,  6.68, 7.7600, 8.82,  9.89, 11]
HS[2]['r'] = [15.09, 16.20, 17.24, 18.29, 19.35]
HS[2]['l'] = [16.72,  17.76, 18.81]
HS[3]['r'] = [23.07, 24.14, 25.21, 26.26, 27.37]
HS[3]['l'] = [22.52, 23.61, 24.67, 25.74, 26.81]
HS[4]['r'] = [31.05, 32.20, 33.29, 34.37, 35.49]
HS[4]['l'] = [31.63, 32.74, 33.82, 34.92]


# Calculate stride velocity by dividing the position by time
# Heel strike position/time to heel strike

for i in range(0,5):
        stride_velocity[i]['r'], mean_stride_velocity[i]['r'] = stride.vel(time, \
                        earth_position['z'], earth_velocity['z'], HS[i]['r'])
        stride_velocity[i]['l'], mean_stride_velocity[i]['r'] = stride.vel(time, \
                        earth_position['z'], earth_velocity['z'], HS[i]['l'])

# Another place where HS and TO parameters are randomly found....
TO = {}
HS.update({'r':  [4.93, 6.17, 7.27, 8.35, 9.41, 10.48, 11.6, 12.78]})
TO['r'] = [5.69, 6.83, 7.92, 8.98, 10.05, 11.16, 12.3]
HS.update({'l': [5.55, 6.7, 7.79, 8.85, 9.92, 11.02, 12.17]})
TO['l'] = [5.14, 6.31, 7.41, 8.48, 9.56, 10.63, 11.75, 12.88]

# figure 12a
orientation = ['r', 'l']
legend1 = ['right HS', 'right TO', 'left HS', 'left TO']
linestyles = ['c*', 'm*', 'bo', 'ro']

plt.figure()
plt.subplot(211)
for i in range(0,3):
        plt.plot(time, earth_accel_delta_vector_holder[i,:])
for c,i in enumerate(orientation,1):
        plt.plot(HS[i], [0] * len(HS[i]), linestyles[(c*2)-2])
        plt.plot(TO[i], [0] * len(TO[i]), linestyles[(c*2)-1])
plt.legend(coordinates + legend1)

# figure 12b
plt.subplot(212)
for axis in coordinates:
        plt.plot(time, gyro[axis])
for c,i in enumerate(orientation, 1):
        plt.plot(HS[i], [0] * len(HS[i]), linestyles[c*2-2])
        plt.plot(TO[i], [0] * len(TO[i]), linestyles[c*2-1])
plt.legend(coordinates + legend1)

# Calculations
fs = 100
f_cuts = [10/fs, 11/fs]
earth_filt = {}
```

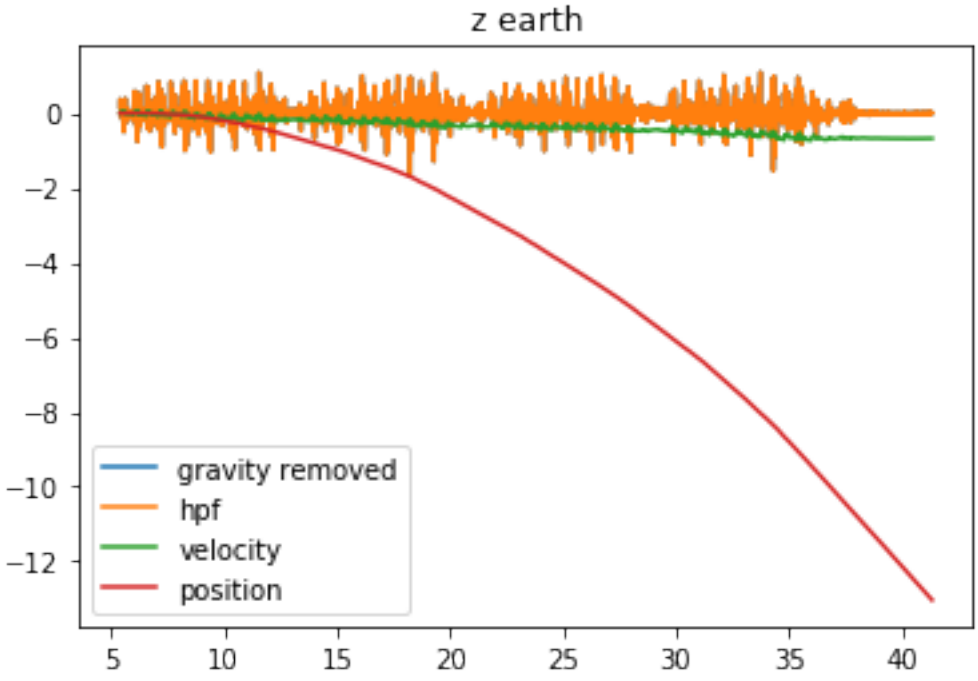
```
earth_filt['x'],_,_,_ = \
                filt.lowpass(earth_accel_delta_vector_holder[0,:], f_cuts, fs, ripple_tol)
gyro_filt = {}
gyro_filt['z'] = gyro['z']

# figure 13a
plt.figure()
plt.subplot(211)
plt.plot(time, earth_filt['x'])
plt.plot(time, difference.first(earth_filt['x'], 1))
for c,i in enumerate(orientation, 1):
        plt.plot(HS[i], [0] * len(HS[i]), linestyles[c*2-2])
        plt.plot(TO[i], [0] * len(TO[i]), linestyles[c*2-1])
plt.legend(['xaccel', 'x diff'] + legend1)

# figure 13b
plt.subplot(212)
plt.plot(gyro['sec'], [500*x for x in angular_pos['z']])
plt.plot(gyro['sec'], difference.first([500*x for x in angular_pos['z']], 20))
plt.plot(gyro['sec'], gyro_filt['z'])
for c,i in enumerate(orientation, 1):
        plt.plot(HS[i], [0] * len(HS[i]), linestyles[c*2-2])
        plt.plot(TO[i], [0] * len(TO[i]), linestyles[c*2-1])
plt.legend(['z angle', 'z gyro', 'z diff'] + legend1)




plt.show()
```

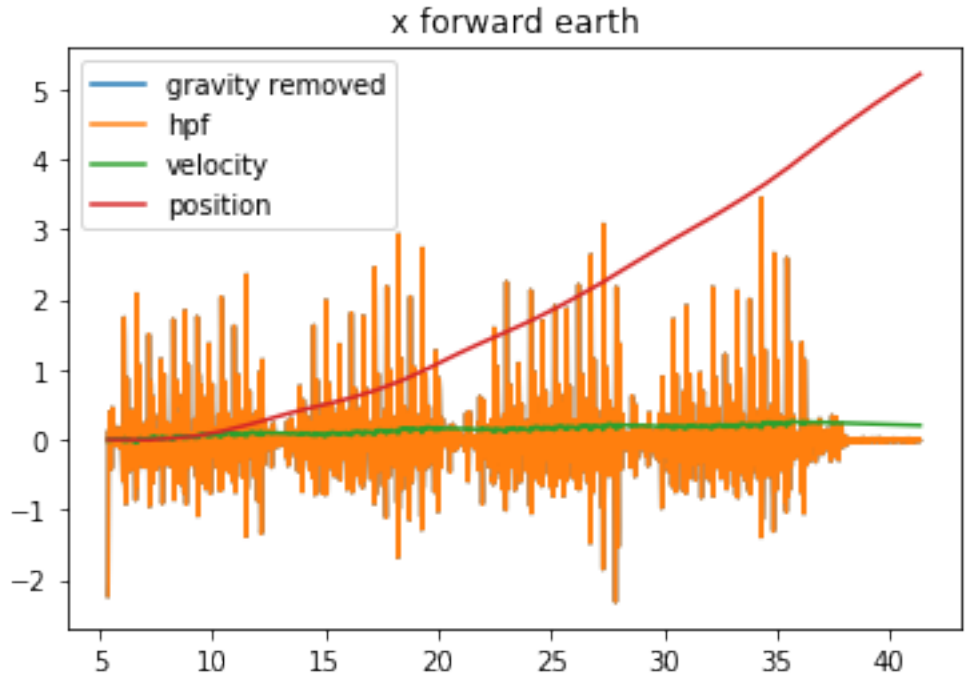


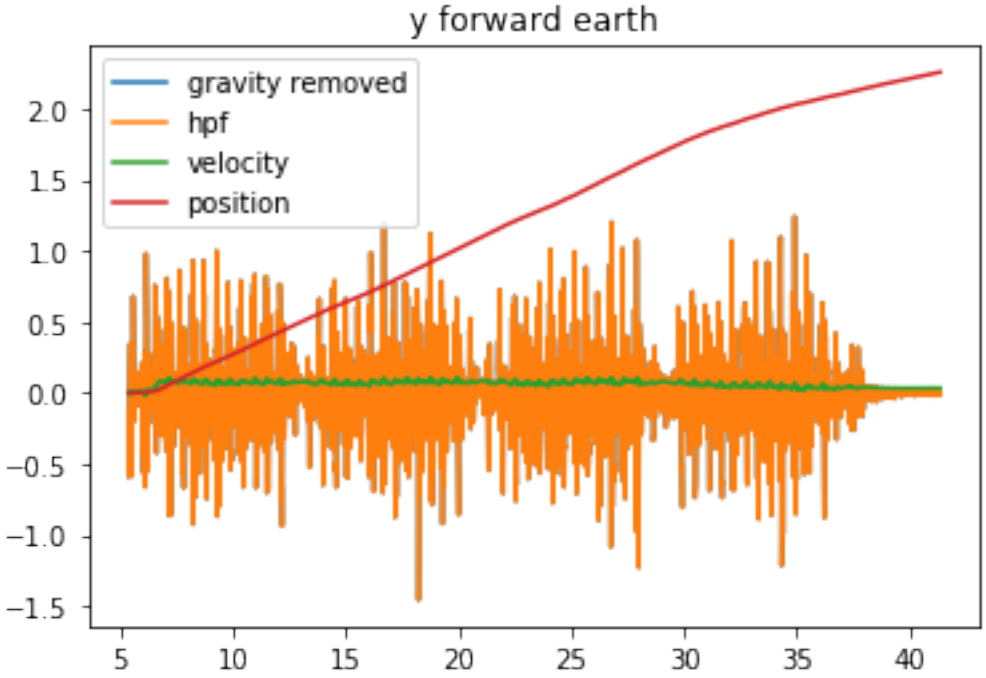

z 2
gravity removed
hpf
velocity
position
600
500
400
300
200
100
0
5
10
15
20
25
30
35
40

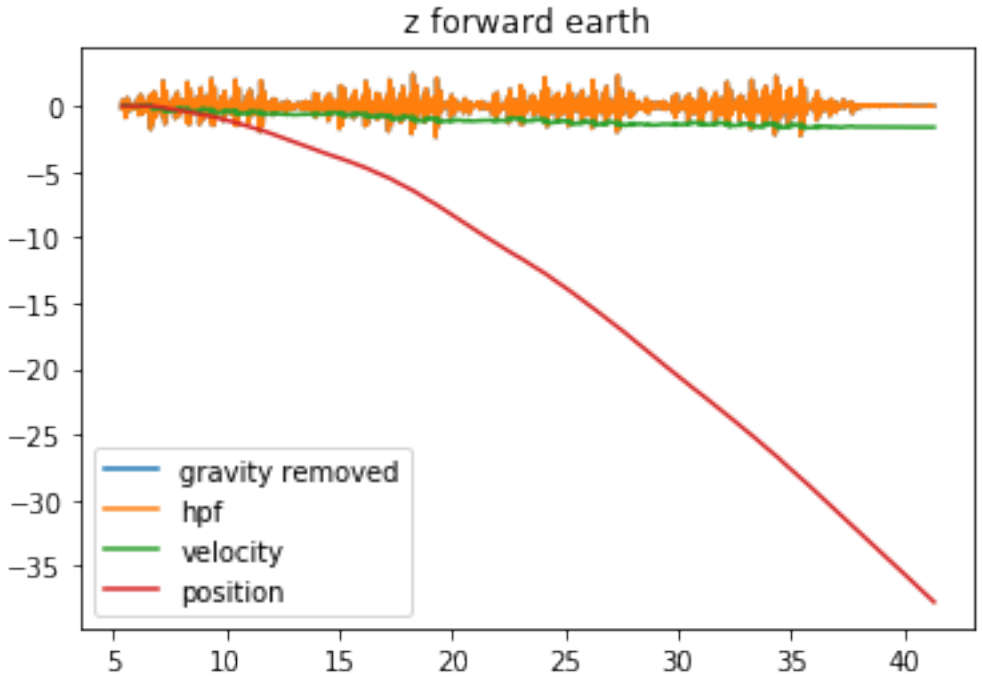

x earth
gravity removed
hpf
velocity
position
3
2
1
0
−1
−2
5
10
15
20
25
30
35
40

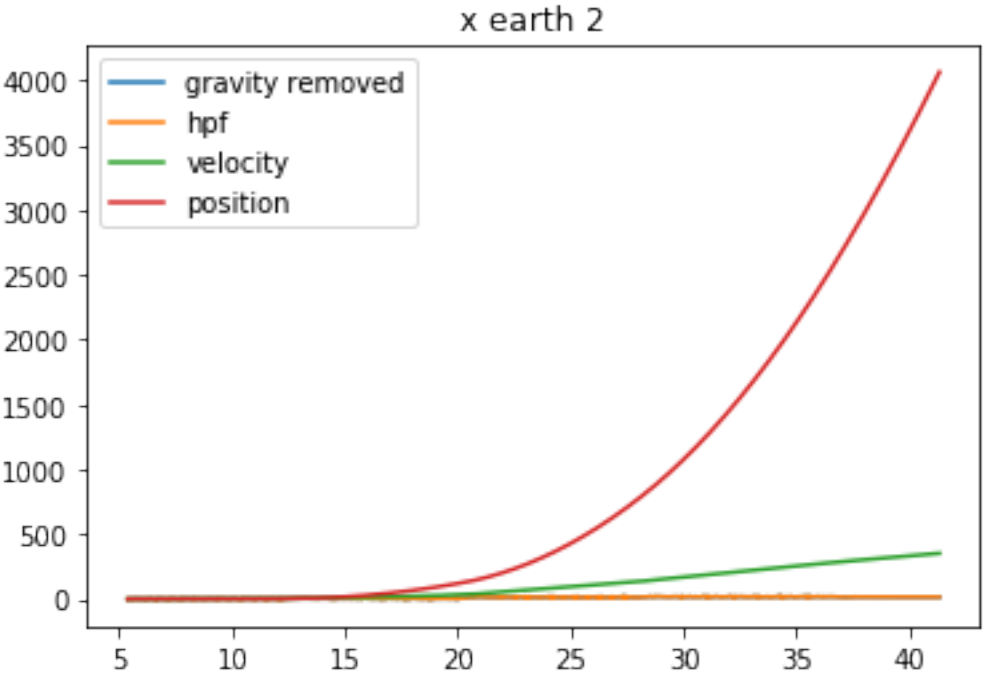

y earth
gravity removed
hpf
velocity
position
8
6
4
2
0
−2
5
10
15
20
25
30
35
40

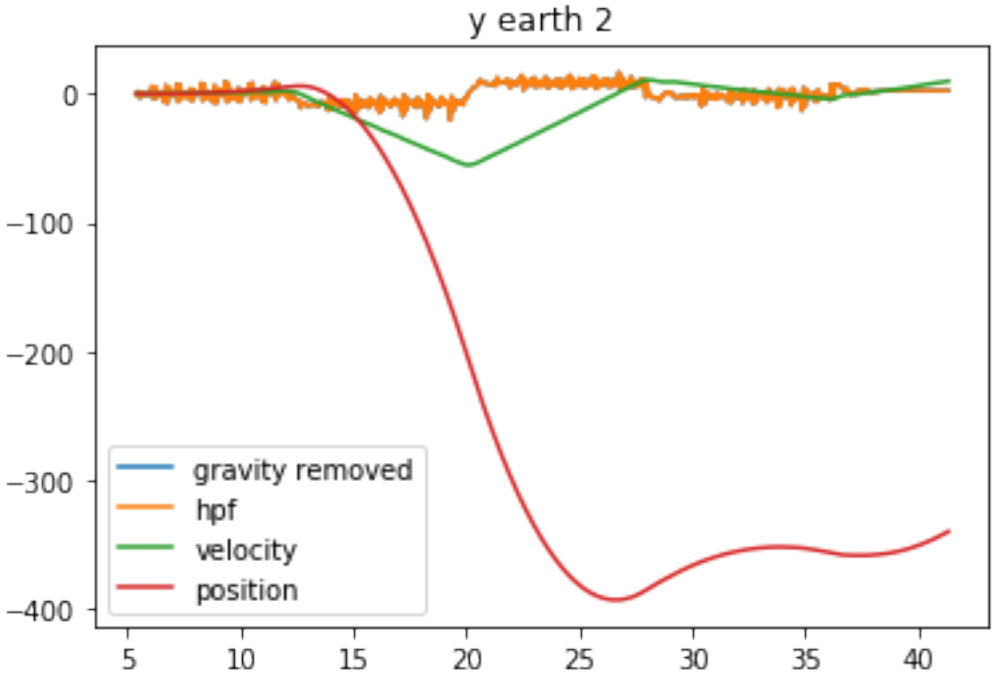

z earth
0
−2
−4
−6
−8
−10
−12
gravity removed
hpf
velocity
position
5
10
15
20
25
30
35
40

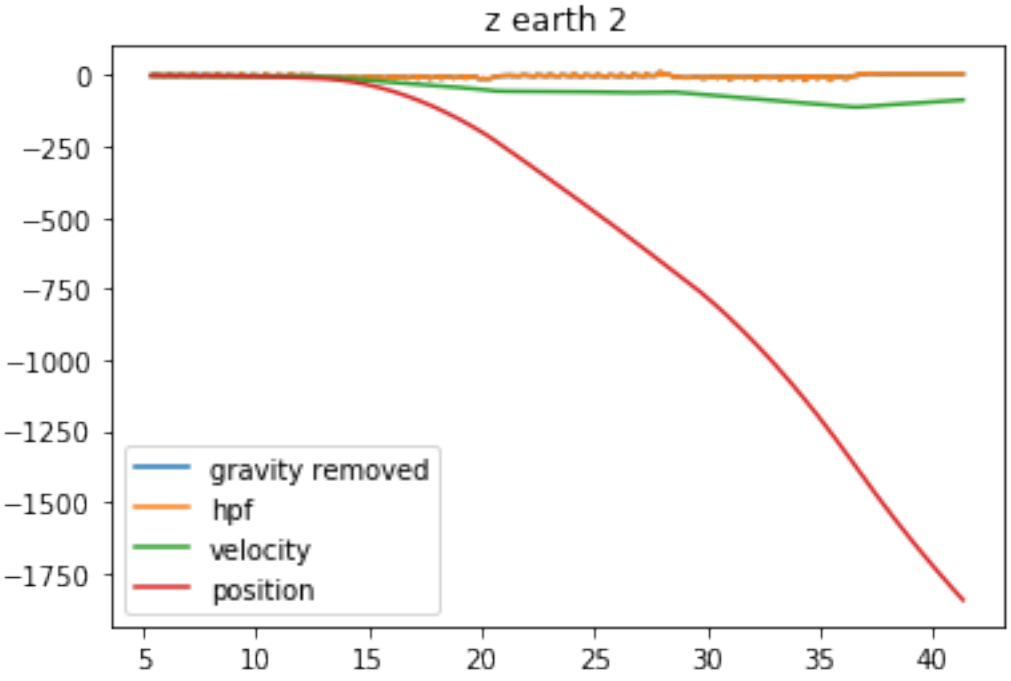

x forward earth
gravity removed
hpf
velocity
position
5
4
3
2
1
0
−1
−2
5
10
15
20
25
30
35
40

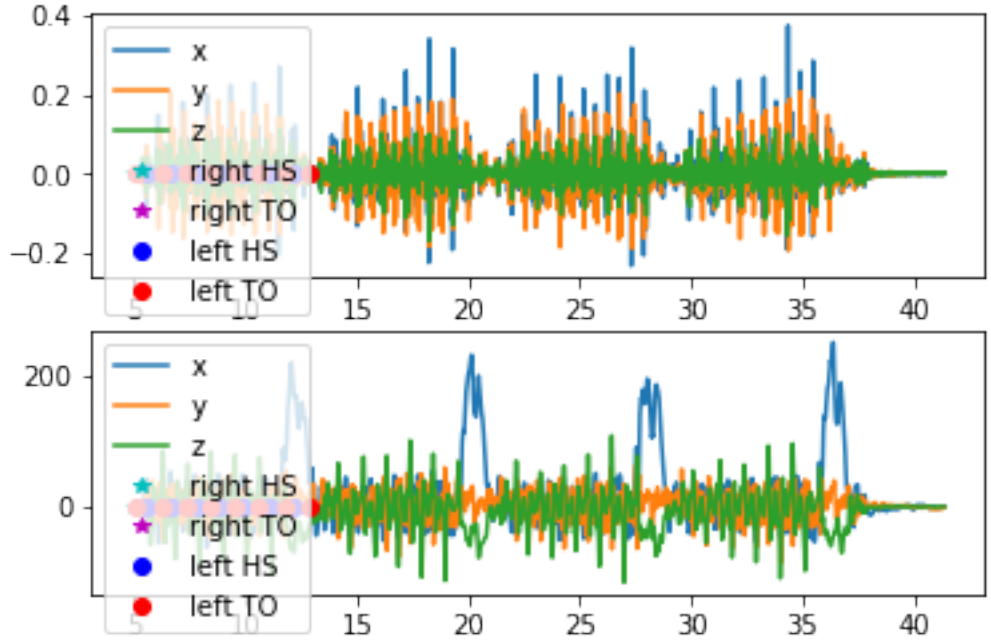

y forward earth
gravity removed
hpf
velocity
position
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
5
10
15
20
25
30
35
40

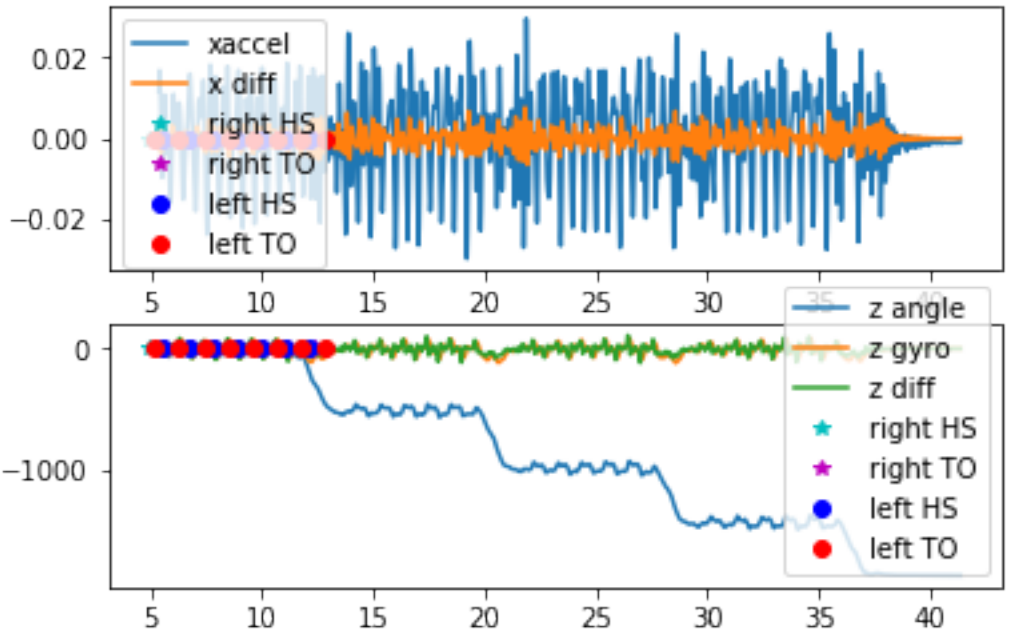

z forward earth
gravity removed
hpf
velocity
position
0
−5
−10
−15
−20
−25
−30
−35
5
10
15
20
25
30
35
40


x earth 2
gravity removed
hpf
velocity
position
4000
3500
3000
2500
2000
1500
1000
500
0
5
10
15
20
25
30
35
40


y earth 2
0
−100
−200
−300
−400
5
10
15
20
25
30
35
40
gravity removed
hpf
velocity
position


z earth 2
0
−250
−500
−750
−1000
−1250
−1500
−1750
5
10
15
20
25
30
35
40
gravity removed
hpf
velocity
position


x
y
z
right HS
right TO
left HS
left TO
0.4
0.2
0.0
−0.2
200
0
5
10
15
20
25
30
35
40


xaccel
x diff
right HS
right TO
left HS
left TO
0.02
0.00
−0.02
z angle
z gyro
z diff
0
−1000
5
10
15
20
25
30
35
40