sensor_fusion_3_22

March 22, 2021

[25]:
```
import math
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

TEST_NAME = "euler_angles"

GRAVITY = 9.80665
RAD_TO_DEG = 180 / math.pi
dt = 1/960
```

[26]:
```
# read test params from CSVP
csvp = open(f"data/{TEST_NAME}.csvp")

# create params array
params = []
for line in csvp: params.append(eval(line))
params = np.array(params)

print(params)
```

```
[   1 3813    0  300    0    1    0  960   96    8 2000   92   92   63
   63   14   14   86   86    0    0    0    0    0    0    0    0    0
    0    0    0]
```

[27]:
```
# read data from CSV
data = pd.read_csv(f"data/{TEST_NAME}.csv", names=["AccelX", "AccelY",␣
↪"AccelZ", "GyroX", "GyroY", "GyroZ", "MagX", "MagY", "MagZ"],␣
↪index_col=False)

sample_rate = params[7]

# add time axis to data set
time = np.arange(0, len(data)/sample_rate, 1/sample_rate)
data.insert(0, "Time", time)

# sign data
data = data.applymap(lambda x: x-65535 if x > 32767 else x)
```

```python
# apply accel sensitivity
acc_cols = ["AccelX", "AccelY", "AccelZ"]
acc_sens = params[9]
data[acc_cols] = data[acc_cols].applymap(lambda x: x * acc_sens * GRAVITY / 32768)

# apply gyro sensitivity
gyro_cols = ["GyroX", "GyroY", "GyroZ"]
gyro_sens = params[10]
data[gyro_cols] = data[gyro_cols].applymap(lambda x: x * gyro_sens / 32768)

# apply mag sensitivity
mag_cols = ["MagX", "MagY", "MagZ"]
mag_sens = 4800
data[mag_cols] = data[mag_cols].applymap(lambda x: x * mag_sens / 8192)

# create new mag dataframe by removing all NaNs
mag_data = data[mag_cols + ["Time"]].dropna()

# for some reason, the first mag data point is always erroneous, so remove it
mag_data = mag_data.iloc[1:]

# calculate offsets for each sensor (first 0.5s of data)
acc_offsets = data[acc_cols].head(480).mean()
gyro_offsets = data[gyro_cols].head(480).mean()
mag_offsets = mag_data[mag_cols].mean()

# apply offsets to each sensor (remove sensor bias)
# TODO: hold off on accel until actual IMU calibration is implemented
for i, axis in enumerate(gyro_cols):
    data[axis] = data[axis].map(lambda x: x - gyro_offsets[i])

for i, axis in enumerate(mag_cols):
    data[axis] = data[axis].map(lambda x: x - mag_offsets[i])

print(acc_offsets, gyro_offsets, mag_offsets)
print(data.head(20))
```

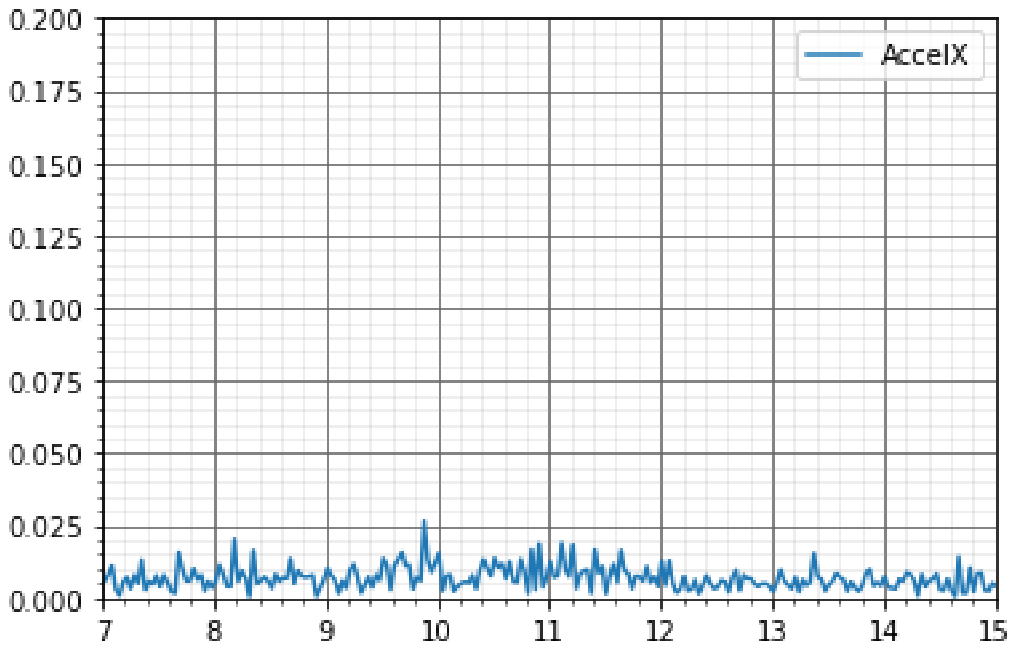
```
AccelX    0.579721
AccelY   -0.262405
AccelZ   -9.943548
dtype: float64 GyroX    -1.288223
GyroY    -3.983561
GyroZ    14.923859
dtype: float64 MagX   -316.491174
MagY    132.152067
MagZ   -164.916010
```

```
dtype: float64
        Time    AccelX    AccelY     AccelZ      GyroX      GyroY      GyroZ  \
0   0.000000 -0.299275 -0.936133 -10.685322  -0.848007   2.518717 -12.238312
1   0.001042  1.431733 -0.639252 -10.120290   6.903458   0.809733   9.673309
2   0.002083  0.555455  0.957681 -10.168174   4.095840  -8.467611  -0.763702
3   0.003125  0.052672 -1.094150  -9.512163   1.166153 -13.655599 -16.938019
4   0.004167  0.167594 -0.179565  -9.890447  -3.716660   3.983561  -1.068878
5   0.005208  0.435745 -0.869095 -10.915165   0.922012  -1.936849   5.217743
6   0.006250  0.699107 -0.940921  -9.454702  -0.542831  -1.814779  17.180634
7   0.007292  1.115698  0.359130 -11.657368   3.180313  -7.674154   1.006317
8   0.008333  1.288080 -1.127669  -9.742007   0.738907   0.077311  -8.637238
9   0.009375  0.541090  1.278504  -9.512163  -1.519394  -8.589681 -13.336945
10  0.010417  0.708684 -0.711078  -9.028534   1.532364  -9.566243  12.969208
11  0.011458  0.924162 -0.849942  -9.818621  -3.106308  -1.143392  -8.820343
12  0.012500  0.967257  0.569820 -10.053253  -0.481796  -4.988607   3.264618
13  0.013542  0.885855  0.316035  -9.220071  -8.111191   4.838053 -11.261749
14  0.014583  0.612916  0.201113  -9.449914  -0.115585 -17.561849   6.011200
15  0.015625  1.010353 -0.284910  -9.224859  -0.848007   1.420085  13.640594
16  0.016667  0.397437 -1.319205  -9.052476  -2.007675  -1.387533  -2.899933
17  0.017708  0.086191 -0.505177 -10.316615  10.931778   5.692546  -3.876495
18  0.018750  0.119710  0.459687  -9.325415  -1.824570  -3.157552  -3.266144
19  0.019792  0.814029  0.612916  -9.943119   2.020645   1.969401  -3.998566

         MagX       MagY       MagZ
0   34.655237 -60.667692  45.970698
1         NaN        NaN        NaN
2         NaN        NaN        NaN
3         NaN        NaN        NaN
4         NaN        NaN        NaN
5         NaN        NaN        NaN
6         NaN        NaN        NaN
7         NaN        NaN        NaN
8         NaN        NaN        NaN
9         NaN        NaN        NaN
10 -26.868201  20.777620 -27.271490
11        NaN        NaN        NaN
12        NaN        NaN        NaN
13        NaN        NaN        NaN
14        NaN        NaN        NaN
15        NaN        NaN        NaN
16        NaN        NaN        NaN
17        NaN        NaN        NaN
18        NaN        NaN        NaN
19        NaN        NaN        NaN
```

[39]:

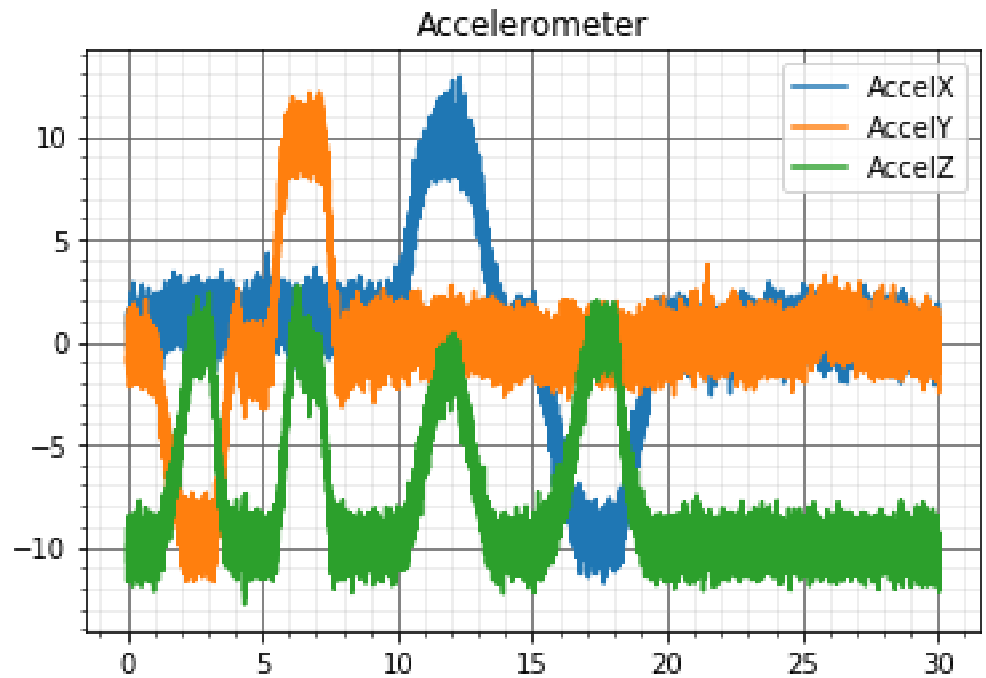
```
# denoising accelerometer
```

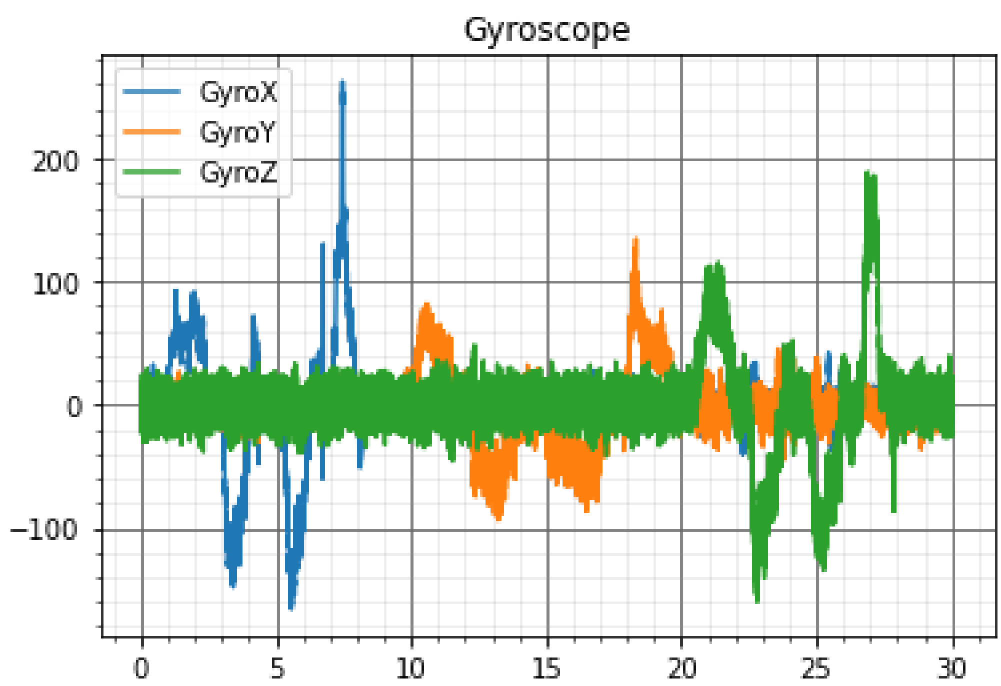
```
# calculate fft for AccelX
fourierTransform = np.fft.fft(data["AccelX"])/len(data["AccelX"])

tpCount = len(data["AccelX"])
values = np.arange(tpCount)
timePeriod = tpCount/sample_rate
frequencies = values/timePeriod

plt.plot(frequencies, abs(fourierTransform), label="AccelX")

# display the plot
plt.xlim(7,15)
plt.ylim(0,0.2)
plt.grid(b=True, which='major', color='#666666', linestyle='-')
plt.minorticks_on()
plt.grid(b=True, which='minor', color='#999999', linestyle='-', alpha=0.2)
plt.legend()
plt.show()

# apply simple moving average
# data[acc_cols] = data[acc_cols].rolling(window=100).mean().
 ↪fillna(data[acc_cols].iloc[49])
```

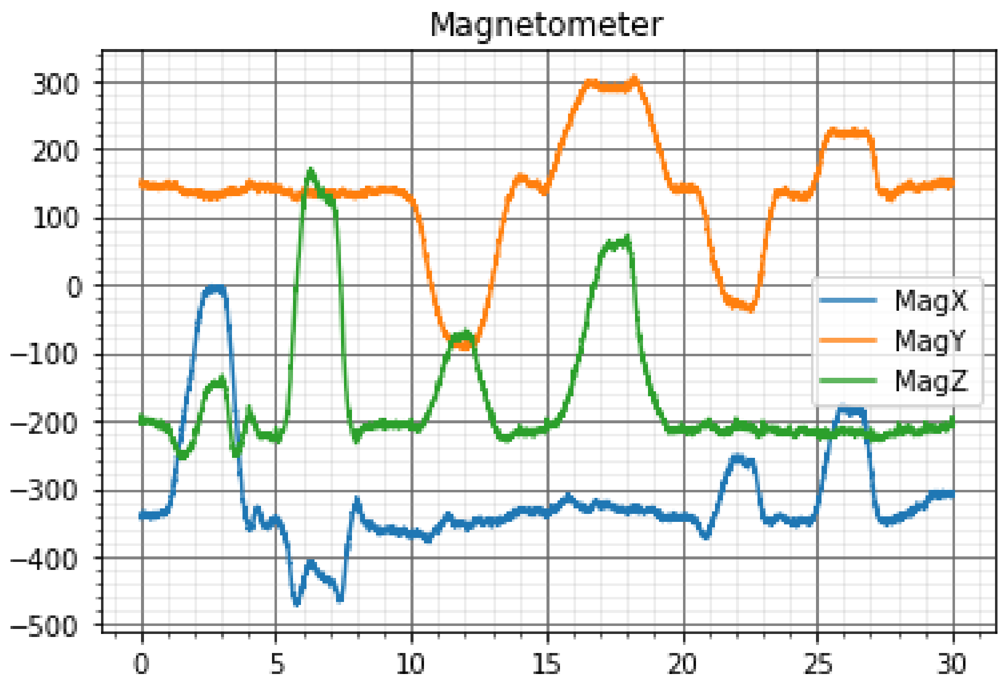


[29]:
```
# plot acceleration
plt.plot(data["Time"], data["AccelX"], label="AccelX")
```

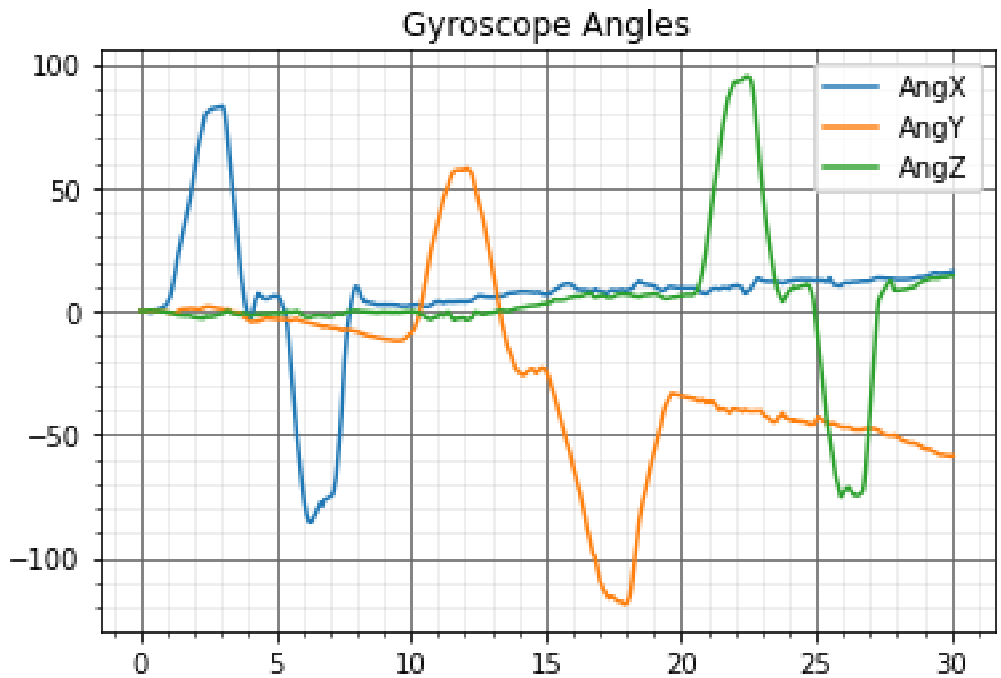
```
plt.plot(data["Time"], data["AccelY"], label="AccelY")
plt.plot(data["Time"], data["AccelZ"], label="AccelZ")

# display the plot
plt.title("Accelerometer")
plt.grid(b=True, which='major', color='#666666', linestyle='-')
plt.minorticks_on()
plt.grid(b=True, which='minor', color='#999999', linestyle='-', alpha=0.2)
plt.legend()
plt.show()

# plot gyroscope
plt.plot(data["Time"], data["GyroX"], label="GyroX")
plt.plot(data["Time"], data["GyroY"], label="GyroY")
plt.plot(data["Time"], data["GyroZ"], label="GyroZ")

# display the plot
plt.title("Gyroscope")
plt.grid(b=True, which='major', color='#666666', linestyle='-')
plt.minorticks_on()
plt.grid(b=True, which='minor', color='#999999', linestyle='-', alpha=0.2)
plt.legend()
plt.show()

# plot magnetometer
plt.plot(mag_data["Time"], mag_data["MagX"], label="MagX")
plt.plot(mag_data["Time"], mag_data["MagY"], label="MagY")
plt.plot(mag_data["Time"], mag_data["MagZ"], label="MagZ")

# display the plot
plt.title("Magnetometer")
plt.grid(b=True, which='major', color='#666666', linestyle='-')
plt.minorticks_on()
plt.grid(b=True, which='minor', color='#999999', linestyle='-', alpha=0.2)
plt.legend()
plt.show()
```


Accelerometer
AccelX
AccelY
AccelZ
10
5
0
−5
−10
0
5
10
15
20
25
30

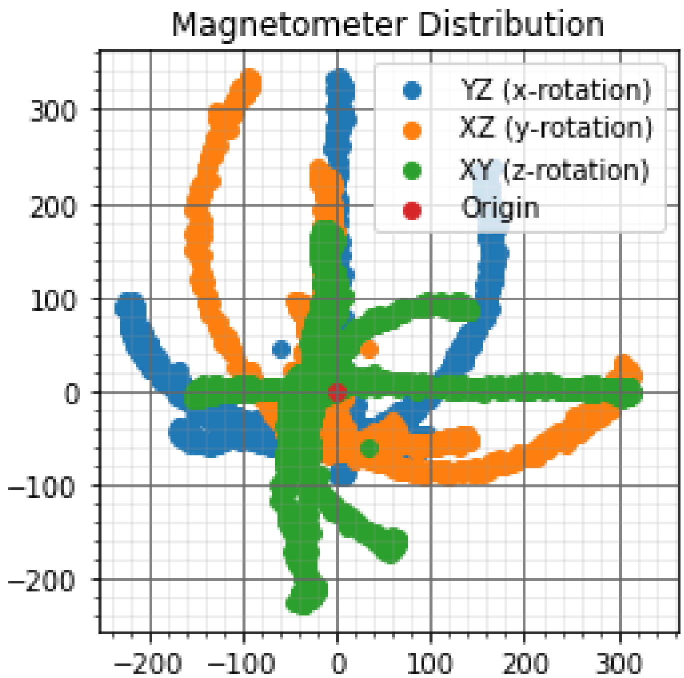

Gyroscope
GyroX
GyroY
GyroZ
200
100
0
−100
0
5
10
15
20
25
30



[30]:
```
from scipy import integrate

# calculate angles from gyroscope
ang_x = integrate.cumtrapz(y=data["GyroX"], x=data["Time"], initial=0)
ang_y = integrate.cumtrapz(y=data["GyroY"], x=data["Time"], initial=0)
ang_z = integrate.cumtrapz(y=data["GyroZ"], x=data["Time"], initial=0)

# plot gyroscope angles
plt.plot(data["Time"], ang_x, label="AngX")
plt.plot(data["Time"], ang_y, label="AngY")
plt.plot(data["Time"], ang_z, label="AngZ")

# display the plot
plt.title("Gyroscope Angles")
plt.grid(b=True, which='major', color='#666666', linestyle='-')
plt.minorticks_on()
plt.grid(b=True, which='minor', color='#999999', linestyle='-', alpha=0.2)
plt.legend()
plt.show()
```

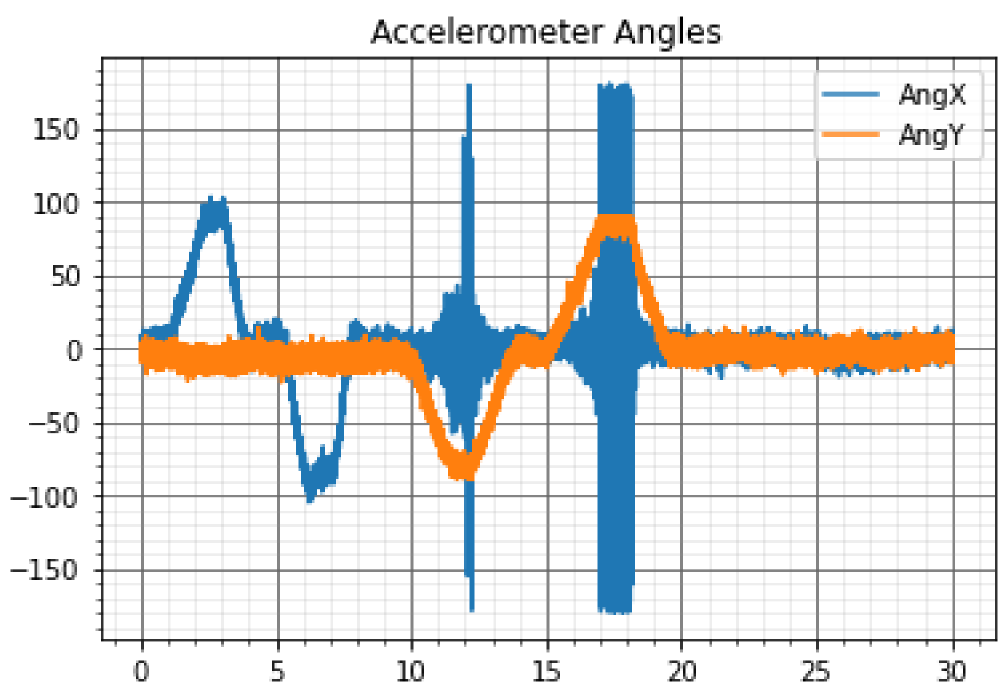


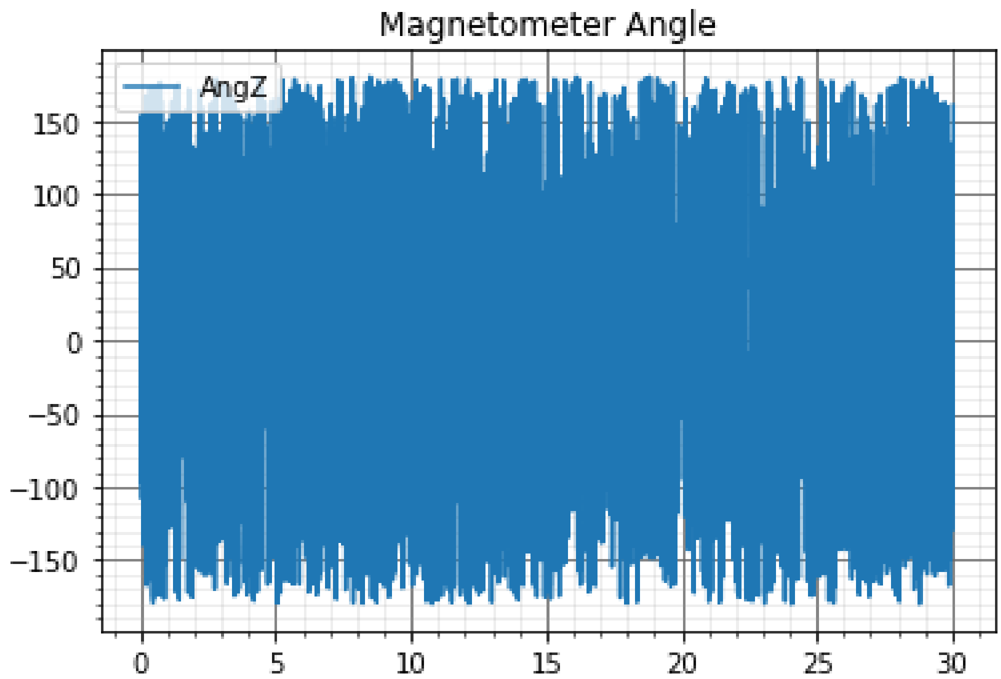
[31]:
```
# hard iron calibration

# plot x-axis rotation (MagY, MagZ)
plt.scatter(data["MagY"], data["MagZ"], label="YZ (x-rotation)")

# plot y-axis rotation (MagX, MagZ)
plt.scatter(data["MagX"], data["MagZ"], label="XZ (y-rotation)")

# plot z-axis rotation (MagX, MagY)
plt.scatter(data["MagX"], data["MagY"], label="XY (z-rotation)")

# plot origin (0,0)
plt.scatter([0], [0], label="Origin")

# display the plot
plt.title("Magnetometer Distribution")
plt.axis("square")
plt.grid(b=True, which='major', color='#666666', linestyle='-')
plt.minorticks_on()
plt.grid(b=True, which='minor', color='#999999', linestyle='-', alpha=0.2)
plt.legend()
plt.show()
```



[32]:
```
# OLD: calculate angles from accelerometer
# acc_ang_x = np.arctan2(-data["AccelY"], -data["AccelZ"]) * RAD_TO_DEG
# acc_ang_y = np.arctan2(-data["AccelX"], np.sqrt(data["AccelY"]**2 +
↪data["AccelZ"]**2)) * RAD_TO_DEG

MU = 0.01

# EXPERIMENTAL ANGLE CALCULATIONS
# TODO: added 2 negatives to acc_ang_x, doesn't work without it: why?
acc_ang_x = np.arctan2(-data["AccelY"], -np.sign(data["AccelZ"]) * np.
↪sqrt(data["AccelZ"]**2 + MU * data["AccelX"]**2)) * RAD_TO_DEG
acc_ang_y = np.arctan2(-data["AccelX"], np.sqrt(data["AccelY"]**2 +
↪data["AccelZ"]**2)) * RAD_TO_DEG

# calculate z-angle (yaw) from accelerometer & magnetometer
M_x = mag_data["MagX"] * np.cos(acc_ang_y) + mag_data["MagZ"] * np.
↪sin(acc_ang_y)
M_y = mag_data["MagX"] * np.sin(acc_ang_x) * np.sin(acc_ang_y) +
↪mag_data["MagY"] * np.cos(acc_ang_x) - mag_data["MagZ"] * np.sin(acc_ang_x)
↪* np.cos(acc_ang_y)

# remove NaNs from mag calculations
M_x = M_x[~np.isnan(M_x)]
M_y = M_y[~np.isnan(M_y)]
```

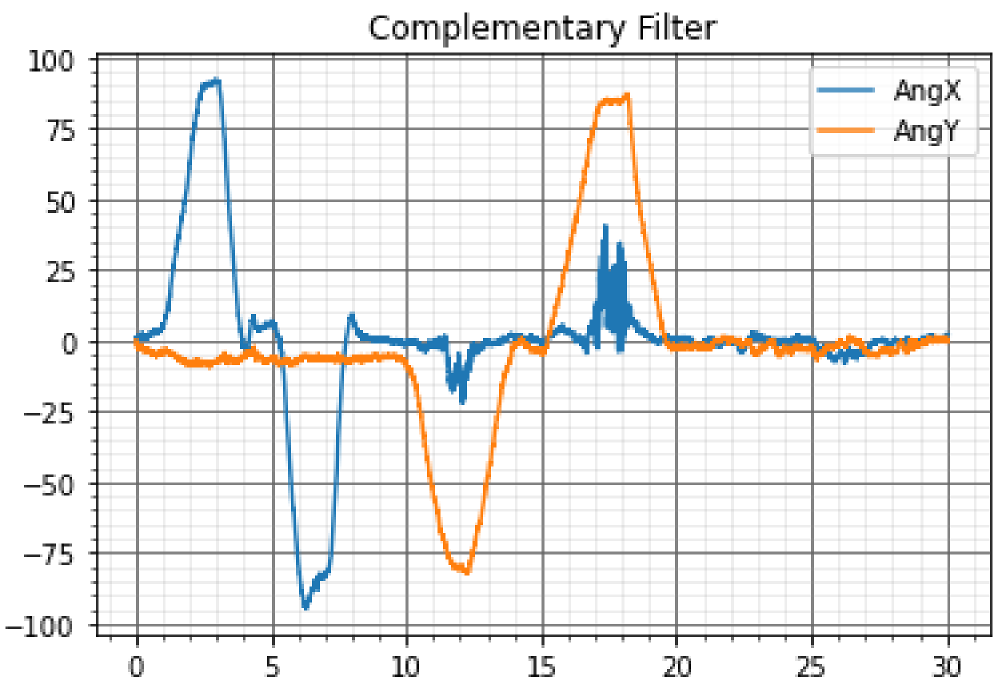
```python
mag_ang_z = np.arctan2(-M_y, M_x) * RAD_TO_DEG

# plot accelerometer angles
plt.plot(data["Time"], acc_ang_x, label="AngX")
plt.plot(data["Time"], acc_ang_y, label="AngY")

# display the plot
plt.title("Accelerometer Angles")
plt.grid(b=True, which='major', color='#666666', linestyle='-')
plt.minorticks_on()
plt.grid(b=True, which='minor', color='#999999', linestyle='-', alpha=0.2)
plt.legend()
plt.show()

# plot mag+accel z-axis angle
plt.plot(mag_data["Time"], mag_ang_z, label="AngZ")

# display the plot
plt.title("Magnetometer Angle")
plt.grid(b=True, which='major', color='#666666', linestyle='-')
plt.minorticks_on()
plt.grid(b=True, which='minor', color='#999999', linestyle='-', alpha=0.2)
plt.legend()
plt.show()
```





[33]:
```
# complementary filter
HP_weight = 0.98
LP_weight = 0.02

# create empty array w/ 3 axes
# set first elements to 0 (will be removed)
cf_ang = [[0],[0],[0]]

# group all axes of calculated angles together
# TODO: add z axis once it matches shape of the others
gyro_ang = np.array([data["GyroX"].to_numpy(),data["GyroY"].to_numpy()])
calc_ang = np.array([acc_ang_x, acc_ang_y])

# pair the calculated arrays for each axis together and loop
# TODO: to add z-axis, just add it to the tuple and as a zip() param
for i, (gyro_arr, calc_arr) in enumerate(zip(gyro_ang, calc_ang)):

    # pair the individual samples together and loop
    for gyro, calc in zip(gyro_arr, calc_arr):
        cf_ang_prev = cf_ang[i][-1]
        cf_ang_samp = HP_weight * (cf_ang_prev + gyro * dt) + LP_weight * calc
        cf_ang[i].append(cf_ang_samp)
```

```
# remove initial 0 value
cf_ang[0].pop(0)
cf_ang[1].pop(0)

plt.plot(data["Time"], cf_ang[0], label="AngX")
plt.plot(data["Time"], cf_ang[1], label="AngY")

# display the plot
plt.title("Complementary Filter")
plt.grid(b=True, which='major', color='#666666', linestyle='-')
plt.minorticks_on()
plt.grid(b=True, which='minor', color='#999999', linestyle='-', alpha=0.2)
plt.legend()
plt.show()
```



[ ]:

```

```