Bobby Vielma CSCE 462

Alexia Perez

Luke Milhorn

Lab 4 Report

1. The accelerometer of the MPU6050 allows for 1kHz. We found this to be plenty for our implementation and could at times be so sensitive to count two or more steps per step if our thresholds or filtering was sloppy. The reason we find it to be fast enough is because even if one is running they will not be able to run faster than the MPU can poll.
2. We did in fact apply filters to the sensor. One of the simple ways we found to do so was by keeping a small sleep() function at the end of every loop. This obviously will cut out some of the data, but not enough to impede the accuracy of the algorithm. We also kept a resting average (when the person isn’t moving) to then judge other readings off of. We also had a tolerance level that was a percentage of this resting average. If a reading went above or below the tolerance threshold we knew a step had probably been taken. We also applied a rolling average of the last three accelerometer Z values that smoothed our data out and used that average Z value in our calculations rather than the raw data.
3. Our step counting algorithm uses the accelerometer Z-axis to tell whether a step has been taken or not. It first collects a resting average from the person when they aren’t moving. This is to set a baseline to compare other Z values off of to check if a step has taken place. We also use a rolling average of Z values (based on the last three) in place of the raw data from the sensor to reduce noise, get smoother readings, and increase accuracy.

We then get a percentage of our resting average and set that as our tolerance level. To help us from counting multiple steps at once, we also have a boolean value named “possible step” that we set to 0 and only set to 1 when we think a step has been initiated.

We first compare our rolling average to see if it is greater than our resting plus our tolerance (rollingAverage > restingAverage + tolerance). If our “possible step” is at 0 then we know a step has likely been initiated and so we set “possible step” to true (possibleStep = 1).

Since acceleration decreases whenever a step has been taken we now check if our rolling average is less than our resting average minus the tolerance (rollingAverage < restingAverage + tolerance). If our “possible step” is 1 then we know we must have come to the end of our step and so then increment our step counter, set possible step back to 0, and print out the step number along with the Z-axis acceleration value.

**Code**

import smbus #import SMBus module of I2C

from time import sleep, perf\_counter

import matplotlib.pyplot as plt

#some MPU6050 Registers and their Address

PWR\_MGMT\_1 = 0x6B

SMPLRT\_DIV = 0x19

CONFIG = 0x1A

GYRO\_CONFIG = 0x1B

INT\_ENABLE = 0x38

ACCEL\_XOUT\_H = 0x3B

ACCEL\_YOUT\_H = 0x3D

ACCEL\_ZOUT\_H = 0x3F

GYRO\_XOUT\_H = 0x43

GYRO\_YOUT\_H = 0x45

GYRO\_ZOUT\_H = 0x47

def MPU\_Init():

#write to sample rate register

bus.write\_byte\_data(Device\_Address, SMPLRT\_DIV, 7)

#Write to power management register

bus.write\_byte\_data(Device\_Address, PWR\_MGMT\_1, 1)

#Write to Configuration register

bus.write\_byte\_data(Device\_Address, CONFIG, 0)

#Write to Gyro configuration register

bus.write\_byte\_data(Device\_Address, GYRO\_CONFIG, 24)

#Write to interrupt enable register

bus.write\_byte\_data(Device\_Address, INT\_ENABLE, 1)

def read\_raw\_data(addr):

#Accelero and Gyro value are 16-bit

high = bus.read\_byte\_data(Device\_Address, addr)

low = bus.read\_byte\_data(Device\_Address, addr+1)

#concatenate higher and lower value

value = ((high << 8) | low)

#to get signed value from mpu6050

if(value > 32768):

value = value - 65536

return value

bus = smbus.SMBus(1) # or bus = smbus.SMBus(0) for older version boards

Device\_Address = 0x68 # MPU6050 device address

MPU\_Init()

stepCount = 0

possibleStep = 0

array = []

arrayFiltered = []

past = perf\_counter()

restingTotal = 0.0

num = 0

i=-1

while stepCount < 50:

#Read Accelerometer raw value

acc\_z = read\_raw\_data(ACCEL\_ZOUT\_H)

#Full scale range +/- 250 degree/C as per sensitivity scale factor

Az = acc\_z/16384.0

array.append(Az)

i = i+1

if (i < 2):

continue

arrayFiltered.append((array[i] + array[i-1] + array[i-2]) / 3.0)

Az = arrayFiltered[i-2]

if (i == 2):

restingTotal = restingTotal + Az

num = num + 1

restingAverage = restingTotal/num

tol = 0.03 \* restingAverage

if(Az > restingAverage + tol and possibleStep == 0 and num > 20):

possibleStep += 1

elif(Az < restingAverage - tol and possibleStep == 1): #and perf\_counter() - past > 0.7):

stepCount += 1

possibleStep = 0

print("Steps=%d" %stepCount, "\tAcc\_Z: %.2f" %Az)

past = perf\_counter()

else:

restingTotal += Az

num += 1

sleep(0.05)