#!/usr/bin/env python

# MAE 5010 Autopilot Design airborne man test file.

# 3 Nov 2019, N. Baker and I. Faruque, i.faruque@okstate.edu

from \_\_future\_\_ import print\_function

from pymavlink import mavutil

import multiprocessing, time

import numpy as np

import air

import math

current\_milli\_time = lambda: int(round(time.time() \* 1000))

# define indices of xh for easier access.

x, y, z, vt, alpha, beta, phi, theta, psi, p, q, r = range(12)

# define indices of y for easier access.

ax, ay, az, gyro\_p, gyro\_q, gyro\_r, mag\_x, mag\_y, mag\_z = range(9)

pres\_baro = 9

gps\_posn\_n, gps\_posn\_e, gps\_posn\_d, gps\_vel\_n, gps\_vel\_e, gps\_vel\_d = range(10, 16)

# define indices of servo for easier access.

mode\_flag = 0

rcin\_0, rcin\_1, rcin\_2, rcin\_3, rcin\_4, rcin\_5 = range(1, 7)

servo\_0, servo\_1, servo\_2, servo\_3, servo\_4, servo\_5 = range(7, 13)

throttle, aileron, elevator, rudder, none, flaps = range(7, 13)

# define indices of cmd for easier access.

psi\_c, h\_c = range(2)

def estimator\_loop(y, xh, servo):

# get sensors for read\_sensor function call.

adc, imu, baro, ubl = air.initialize\_sensors()

time.sleep(3)

while True:

new\_gps = air.read\_sensor(y, adc, imu, baro, ubl) # updates values in y

# print(new\_gps)

if (new\_gps):

pass # do estimation with gps here.

else:

pass # do estimation without gps here.

y = np.array(

[0.538385085, -0.035892339, 9.662217659, -0.002443459, 0.133168509, 0.014660753, -5.046, -20.184,

57.246])

A = [[-0.028, 0.233, 0, -9.815, 0, 0, 0, 0, 0, 0, 0, 0]

[-0.978, -8.966, 20.1170, 0, 0, 0, 0, 0, 0, 0, 0, 0]

[0.102, 0.022, -6.102, 0, 0, 0, 0, 0, 0, 0, 0, 0]

[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]

[0, 0, 0, 0, -0.45, 0, -0.986, 0.635, 0, 0, 0, 0]

[0, 0, 0, 0, 57.028, -72.97, 3.279, 0, 0, 0, 0, 0]

[0, 0, 0, 0, 135.737, -0.588, -4.436, 0, 0, 0, 0, 0]

[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]

[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]

[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]

B = [[0, 1, 0, 0]

[-23.448, 0, 0, 0]

[-50.313, -0.104, 0, 0]

[0, 0, 0, 0]

[0, 0, 0, 0.315]

[0, 0, 677.27, 18.099]

[0, 0, -8.875, -99.521]

[0, 0, 0, 0]

[0, 0, 0, 0]

[0, 0, 0, 0]

[0, 0, 0, 0]

[0, 0, 0, 0]]

F = np.expm(A\*0.01)

def heading\_calc(mag\_x, mag\_y, mag\_z, phi, theta):

dt = 3 # declination angle in degrees

M\_B = np.array([mag\_x, mag\_y, mag\_z]) # magnetometer output

M\_i = Rot\_i\_B(M\_B, phi, theta) # put magnetometer in inertial frame

psi = dt + math.atan(M\_i[1] / M\_i[2]) # dt is declination angle

return psi

def Rot\_i\_B(Matrix, phi, theta):

Rot\_Mat = np.array([[math.cos(theta), math.sin(theta) \* math.sin(phi), math.sin(theta) \* math.cos(phi)],

[0, math.cos(phi), -math.sin(phi)],

[-math.sin(theta), math.cos(theta) \* math.sin(phi),

math.cos(theta) \* math.cos(phi)]])

R\_Mat = Matrix \* Rot\_Mat

return R\_Mat

# define initial states from y

ax = y[0]

ay = y[1]

az = y[2]

gyro\_p = y[3]

gyro\_q = y[4]

gyro\_r = y[5]

mag\_x = y[6]

mag\_y = y[7]

mag\_z = y[8]

# definition of noise and biases

AccelVariance = .002 # noise of accelerometer

GyroVariance = 1e-5 # noise of gyro

AttitudeVariance = .3 # attitude noise

AccelBias = np.array([0, 0, 0]) # Bias of Accelerometer... from christian

GyroBias = np.array([0, 0, 0]) # gyro bias

MagBias = np.array([0, 0, 0]) # magnetometer bias

# initial orientation estimate

phi = 0

theta = 0

psi = heading\_calc(mag\_x, mag\_y, mag\_z, phi, theta) # need to create function to do this

euler = np.array([phi, theta, psi])

# define x\_hat

x\_hat = np.array(

[euler, GyroBias, AccelBias]) # 1-3 Euler angles ,,,, #4-6 xyz gyro bias estimates, #7-9 xyz gyro bias

# define R uncertainty in measurement

Variance = np.arrays[AccelVariance,AccelVariance,AccelVariance,GyroVariance,GyroVariance,GyroVariance,AttitudeVariance,AttitudeVariance,AttitudeVariance]

Identity = np.identity(9)

R = Identity \* Variance

# Define Q uncertainty in model

Q = np.identity(9)

EKFl = EKF(initial\_x=x\_hat, initial\_P=P)

test = EXF1.step(F, Q, G, U, y, hx, C, R)

# write estimated values to the xh array.

def controller\_loop(xh, servo, cmd):

while True:

if (servo[mode\_flag] == 1):

pass # rewrite servo\_out values to servo array based on their previous values and xh.

# if (servo[servo\_1]<1.5): servo[servo\_1] = 1.55

# else: servo[servo\_1] = 1.45

# time.sleep(1)

# Controller should assign values in range 1.25 to 1.75 to outputs;

# WARNING, servo damage likely if values outside this range are assigned

# Example: This is a manual passthrough function

servo[throttle] = servo[rcin\_0]

servo[aileron] = servo[rcin\_1]

servo[elevator] = servo[rcin\_2]

servo[rudder] = servo[rcin\_3]

servo[servo\_4] = servo[servo\_4] # no servo; channel used for manual/auto switch

servo[flaps] = servo[rcin\_5]

if \_\_name\_\_ == "\_\_main\_\_":

master = mavutil.mavlink\_connection('/dev/ttyAMA0', baud=57600, source\_system=255)

# initialize arrays for sharing sensor data.

y = multiprocessing.Array('d', np.zeros(26)) # imu, baro, gps, adc

xh = multiprocessing.Array('d', np.zeros(12)) # position, orientation, rates

servo = multiprocessing.Array('d', np.zeros(13)) # mode\_flag, rcin, servo\_out

cmd = multiprocessing.Array('d', np.zeros(2)) # psi\_c, h\_c

# start processes for interpreting sensor data and setting servo pwm.

estimator\_process = multiprocessing.Process(target=estimator\_loop, args=(y, xh, servo))

estimator\_process.daemon = True

estimator\_process.start()

controller\_process = multiprocessing.Process(target=controller\_loop, args=(xh, servo, cmd))

controller\_process.daemon = True

controller\_process.start()

servo\_process = multiprocessing.Process(target=air.servo\_loop, args=(servo,))

servo\_process.daemon = True

servo\_process.start()

time.sleep(5)

# start process for telemetry after other processes have initialized.

telemetry\_process = multiprocessing.Process(target=air.telemetry\_loop, args=(y, xh, servo, master))

telemetry\_process.daemon = True

telemetry\_process.start()

print("\nsending heartbeats to {} at 1hz.".format('/dev/ttyAMA0'))

# loop for sending heartbeats and receiving messages from gcs.

while True:

time\_sent = 0

while True:

# send heartbeat message if one second has passed.

if ((current\_milli\_time() - time\_sent) >= 980):

master.mav.heartbeat\_send(1, 0, 0, 0, 4, 0)

# still haven't figured out how to get mode to show up in mission planner.

# print('heartbeat sent.')

time\_sent = current\_milli\_time()

# Simple waypoint tracker

# DO WAYPOINT TRACKING HERE

# handle incoming commands over telemetry

# try:

# msg = master.recv\_match().to\_dict()

# if (not (msg['mavpackettype'] == 'RADIO' or msg['mavpackettype'] == 'RADIO\_STATUS' or msg['mavpackettype'] == 'HEARTBEAT')):

# print(msg)

# if (msg['mavpackettype'] == 'COMMAND\_LONG'):

# master.mav.command\_ack\_send(msg['command'],4)

# print("acknowledge sent.")

# except:

# pass