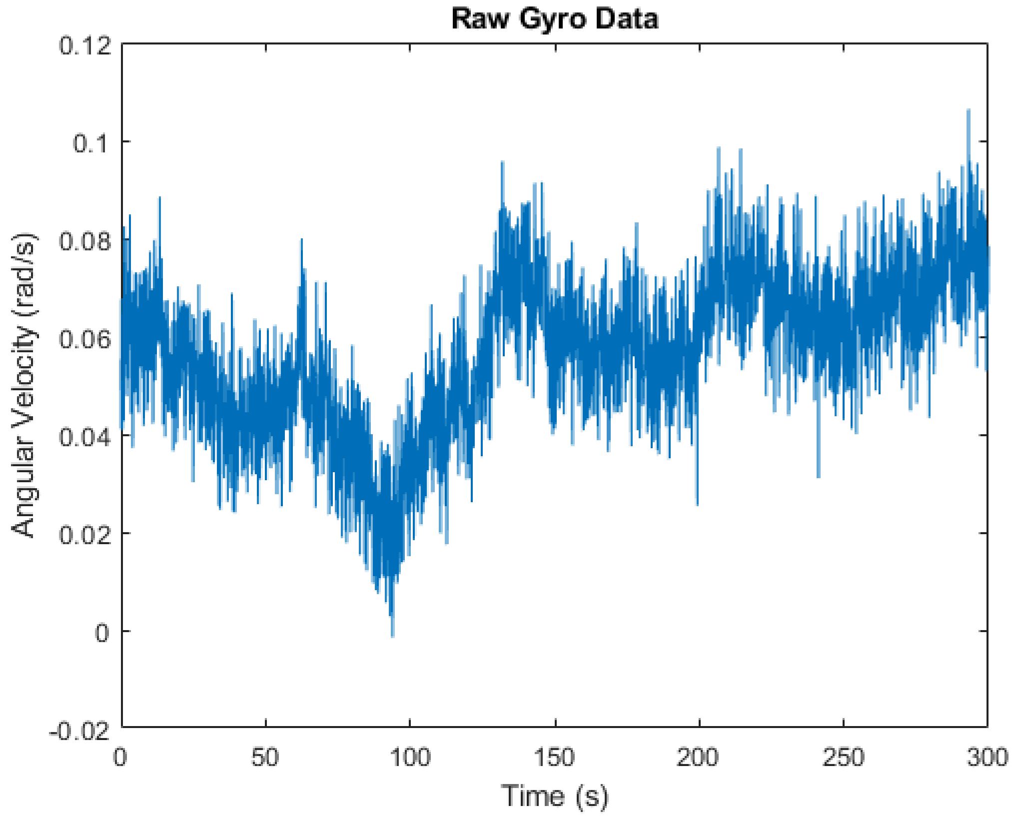
```
clc; clear all; close all;

% HW5

%{
Pt. 1: Gyro Calibration

Determine an error model (bias stability, output noise, turn on bias,
 etc) for
the imu. What limitations would you put on your model?

Notes:
* Gyro is static
* Gyro samples at 10Hz
%}
gyro_data = load('gyro.mat');
gyro_t = gyro_data.t;
gyro_y = gyro_data.imu_signal;

figure(1)
plot(gyro_t, gyro_y);
title("Raw Gyro Data");
xlabel("Time (s)");
ylabel("Angular Velocity (rad/s)");

%{
Methodology: Use the model:
W_meas(t) = W_true(t) + B_N(t) + B_K(t) + B_B(t) where
B_N(t): Noise from angle random walk
B_K(t): Noise from rate random walk
B_B(t): Noise from bias instability

We will use the Allan variance parameters to find these

NOTE: These characterize the dynamic errors of the gyro. We can
 approximate
the static offset bias as the mean of the data taken, but that is not
 a
particularly scientific approach.
%}

static_bias = mean(gyro_y)

Fs = 10;
dt = 1/Fs;
theta = cumsum(gyro_y, 1)*dt;

maxNumM = 100;
L = size(theta, 1);
maxM = 2.^floor(log2(L/2));
m = logspace(log10(1), log10(maxM), maxNumM).';
m = ceil(m); % m must be an integer.
```

```
m = unique(m); % Remove duplicates.

tau = m*dt;

avar = zeros(numel(m), 1);
for i = 1:numel(m)
    mi = m(i);
    avar(i,:) = sum( ...
        (theta(1+2*mi:L) - 2*theta(1+mi:L-mi) + theta(1:L-2*mi)).^2,
 1);
end
avar = avar ./ (2*tau.^2 .* (L - 2*m));
adev = sqrt(avar);

% Finding angle random walk coeff:
% Find the index where the slope of the log-scaled Allan deviation is
 equal
% to the slope specified.
slope = -0.5;
logtau = log10(tau);
logadev = log10(adev);
dlogadev = diff(logadev) ./ diff(logtau);
[~, i] = min(abs(dlogadev - slope));

% Find the y-intercept of the line.
b = logadev(i) - slope*logtau(i);

% Determine the angle random walk coefficient from the line.
logN = slope*log(1) + b;
N = 10^logN


% Finding rate random walk coeff:
% Find the index where the slope of the log-scaled Allan deviation is
 equal
% to the slope specified.
slope = 0.5;
logtau = log10(tau);
logadev = log10(adev);
dlogadev = diff(logadev) ./ diff(logtau);
[~, i] = min(abs(dlogadev - slope));
% Find the y-intercept of the line.
b = logadev(i) - slope*logtau(i);
% Determine the rate random walk coefficient from the line.
logK = slope*log10(3) + b;
K = 10^logK


% Find bias instability coeff:
% Find the index where the slope of the log-scaled Allan deviation is
 equal
% to the slope specified.
slope = 0;
logtau = log10(tau);
```

```
logadev = log10(adev);
dlogadev = diff(logadev) ./ diff(logtau);
[~, i] = min(abs(dlogadev - slope));
% Find the y-intercept of the line.
b = logadev(i) - slope*logtau(i);
% Determine the bias instability coefficient from the line.
scfB = sqrt(2*log(2)/pi);
logB = b - log10(scfB);
B = 10^logB
```

```
static_bias =

    0.0566


N =

    0.0027


K =

    0.0029


B =

    0.0129
```




```
clear all;
%{
Pt. 2: Coarse Levelling Using Earth Rate and Gravity

Compute the Euler angles for the initial platform tilt

Notes:
* 35' latitude
%}

g = 9.80756;
w_ie = 7.2921159e-05;

accel = [-0.1710; -0.2564; 9.7925];
ax = accel(1);
ay = accel(2);
az = accel(3);

gyro = [-0.5245; -0.2879; -0.4168] * 10^-4;
wx = gyro(1);
wy = gyro(2);
wz = gyro(3);

lat = 35;

C_n_b = (1/(g*w_ie*cosd(lat))) * ...
```

```
    [g*wx - ax*w_ie*sind(lat), g*wy - ay*w_ie*sind(lat), g*wz-
az*w_ie*sind(lat);
    az*wy-ay*wz, ay*wz-az*wx, ay*wx-ax*wy;
    -cosd(lat)*ax*w_ie, -cosd(lat)*ay*w_ie, -cosd(lat)*az*w_ie];

euler_ZYX =  rad2deg(rotm2eul(C_n_b));
roll = euler_ZYX(3)
pitch = euler_ZYX(2)
yaw = euler_ZYX(1)
```

```
roll =

  178.5001


pitch =

   -0.9993


yaw =

 -150.0212
```

```
clear all;
%{
Pt. 3: IMU Coarse Alignment

a) What will the readings on the three rate gyros be if you have a
 strapdown
IMU at: N37.6195, W122.3739, 10m altitude? It is perfectly level and
pointed at a true heading of 060.

b) Suppose the bias instability is 0.1deg/hr. How far south do you
 have to
go before the change in gyro readings exceeds this?

c) Repeat (b) with instabilities of 1, 10, 20 deg/hr

%}

g = 9.80756;
w_ie = 7.2921159e-05;

p_g_nb = [37.6195; -122.3739; 10]; % LLA
rpy = deg2rad([0; 0; 60]);

C_n_b = body_to_nav(rpy);
C_b_n = C_n_b'
% A:
W_b_ib = C_b_n * [cosd(p_g_nb(1)); 0; -sind(p_g_nb(1))] * w_ie % rad/s
```

```
%b/c
instabilities = [0.1, 1, 10, 20] * 4.84814e-6; % rad/s
inst_labels = [0.1, 1, 10, 20];
for i = 1:length(instabilities)
    p_g_temp = p_g_nb;
    val = instabilities(i);
    step = deg2rad(0.01);
    distance = 0;
    is_exceeded = false;
    while (~is_exceeded)
        p_g_temp = p_g_temp - [step; 0; 0];
        distance = distance + step;
        W_temp = C_b_n * [cosd(p_g_temp(1)); 0; -sind(p_g_temp(1))] *
 w_ie; % rad/s
        W_delta = W_temp - W_b_ib;
        if norm(W_delta) > val
           is_exceeded = true;
        end
    end
    s = ["Accuracy: ", inst_labels(i), ", Travel: ", distance, "
 degrees south"]
end

clear all;
%{
Pt. 4: INS Position and Velocity Mechanization

(Also Pt. 5? I guess I overkilled 4 and went straight to 5)

Propagate the state forward

Notes:
* column 1 is the time
* column 2:4 are yaw pitch roll
* column 5:7 are x y z accelerometer readings f_b_ib
%}

% v^{n}_{nb}
v0 = [0; 206; 0]; % m/s (NED?)

% p^{g}_{nb}
p0 = [44.8805; -93.2169; 256.3]; % LLA
lat_arr(1) = p0(1);
lon_arr(1) = p0(2);

% Using const gravity at starting position for convenience
g_n_b = [0.03297; -0.01132; -9.82483]; % so use -g_n_b

imu_data = load('imudata.mat');
imu_data = imu_data.imudata;

imu_t = imu_data(:,1);
% Orientations in nav?
imu_yaw = imu_data(:,2);
```

```
imu_pitch = imu_data(:,3);
imu_roll = imu_data(:,4);
% Specific forces in nav?
imu_x = imu_data(:,5);
imu_y = imu_data(:,6);
imu_z = imu_data(:,7);
dt = imu_t(2) - imu_t(1);

state = {};
measurements = {};

for i = 1:length(imu_t)
    m.yaw = imu_yaw(i);
    m.pitch = imu_pitch(i);
    m.roll = imu_roll(i);
    m.x = imu_x(i);
    m.y = imu_y(i);
    m.z = imu_z(i);
    measurements{i} = m;
end

s.p_g_nb = p0;
s.v_n_eb = v0;
s.C_n_ib = zeros(3, 3);
states{1} = s;


for i = 1:length(imu_t)
    % Pull in measurements
    m = measurements{i};
    yaw = m.yaw;
    pitch = m.pitch;
    roll = m.roll;
    fx = m.x;
    fy = m.y;
    fz = m.z;

    rpy = [roll; pitch; yaw];
    xyz = [fx; fy; fz]; % = f_b_ib

    % Pull in previous state
    s = states{i};

    % 1: Attitude Update
    % Rotation matrix from body into nav frame
    C_n_b = body_to_nav(rpy);

    % 2: Specific Force Transformation
    % Specific force in nav frame
    f_n_ib = transform_accel(rpy, xyz); % haven't accounted for
 gravity yet

    % 3: Velocity Update
    % TODO: Account for coriolis
```

```
    prev_v = s.v_n_eb;
    a_n_eb = f_n_ib - g_n_b;
    v_n_eb = prev_v + dt * a_n_eb;

    % 4: Position Update
    % TODO
    prev_lla = s.p_g_nb;
    prev_lat = prev_lla(1);
    prev_lon = prev_lla(2);
    prev_alt = prev_lla(3);

    alt = prev_alt - dt * v_n_eb(3);
    lat = prev_lat + dt * (v_n_eb(1)) / (RN(prev_lat) + prev_alt);
    lon = prev_lon + dt * (v_n_eb(2)) / (cosd(prev_lat) *
 (RE(prev_lat) + prev_alt));


    p_g_nb = [lat; lon; alt];
    lat_arr(i+1) = lat;
    lon_arr(i+1) = lon;

    % Append to state history
    new_state.v_n_eb = v_n_eb;
    new_state.C_n_b = C_n_b;
    new_state.p_g_nb = p_g_nb;
    states{i+1} = new_state;
end

figure(2);
plot(lon_arr, lat_arr);
xlabel("Lon");
ylabel("Lat");
title("Trajectory");
snapnow
```

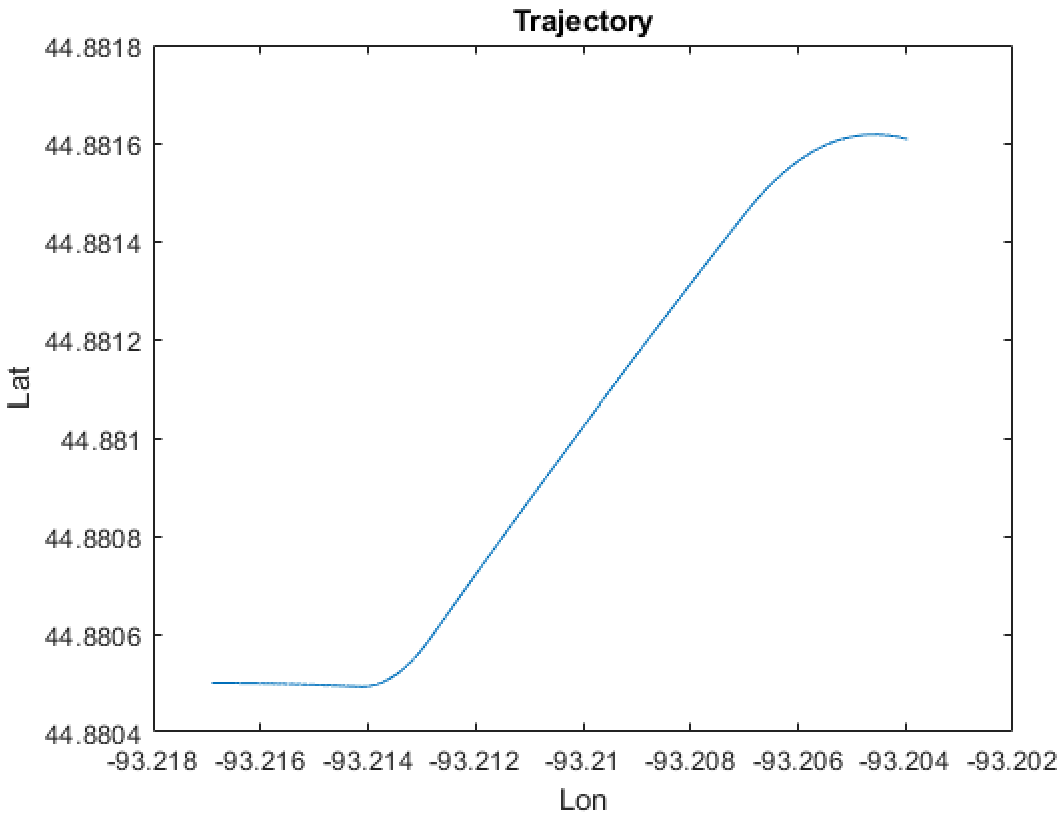
Local Functions

```
function [C_n_b] = body_to_nav(rpy)
    roll = rpy(1);
    pitch = rpy(2);
    yaw = rpy(3);

    Rz = [cos(yaw), -sin(yaw), 0; ...
        sin(yaw), cos(yaw), 0; ...
        0, 0, 1];
    Ry = [cos(pitch), 0, sin(pitch); ...
        0, 1, 0; ...
        -sin(pitch), 0, cos(pitch)];
    Rx = [1, 0, 0; ...
        0, cos(roll), -sin(roll); ...
        0, sin(roll), cos(roll)];

    C_n_b = Rz * Ry * Rx;
```

```
end

function [f_n_ib] = transform_accel(rpy, xyz)
    C_n_b = body_to_nav(rpy);
    f_n_ib = C_n_b * xyz;
end

% Account for ellipsoid
function [r_N] = RN(lat)
    e = 0.0818;
    R_0 = 6378137;
    r_N = ((1 - e^2)*R_0) / ((1 - e^2 * sind(lat) * sind(lat))^(3/2));
end

% Account for ellipsoid
function [r_E] = RE(lat)
    e = 0.0818;
    R_0 = 6378137;
    r_E = R_0 / sqrt(1 - e^2 * sind(lat) * sind(lat));
end
```

```
C_b_n =

    0.5000    0.8660         0
   -0.8660    0.5000         0
         0         0    1.0000


W_b_ib =

   1.0e-04 *

    0.2888
   -0.5002
   -0.4451


s = 

  1×5 string array

    "Accuracy: "    "0.1"    ", Travel: "    "0.38101"    " degrees
 south"


s = 

  1×5 string array

    "Accuracy: "    "1"    ", Travel: "    "3.8101"    " degrees
 south"
```

```
s =

  1×5 string array

    "Accuracy: "    "10"    ", Travel: "    "38.832"    " degrees
south"


s =

  1×5 string array

    "Accuracy: "    "20"    ", Travel: "    "83.3412"    " degrees
south"
```



Published with MATLAB® R2018b