**Problem I: Attitude Determination**

**Given:** Image of stars; Inertial-Frame unit vectors

**To Find:** Body-Frame unit vectors; Rotation matrix using various methods; Error;

***Part A) Extracting Body Frame unit vectors***

The first step was to make sure that Matlab was properly able to open the .png file. The following image shows that Matlab succefully opens the image, as shown in Figure 1.

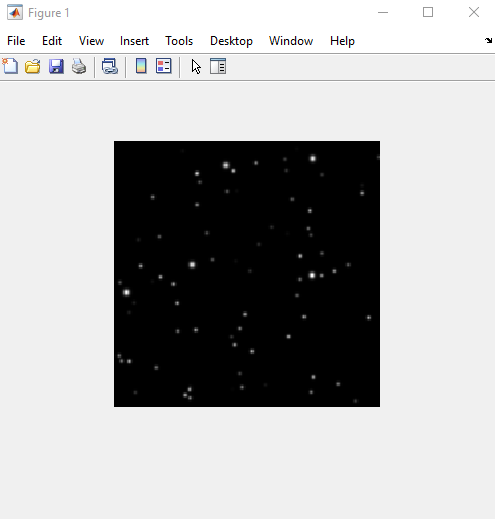


Figure 1. Raw image

The next step was to extract all the stars in the image, regardless of its brightness. The following image shows just that.

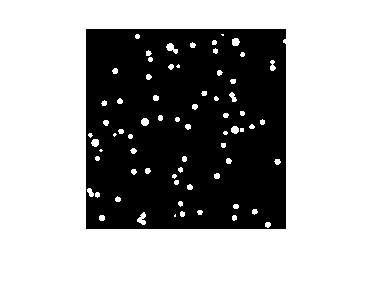


Figure 2. Identified stars

The next step is to find the centroid of each of these stars in the image. Specifically, the x & y pixel coordinates, whose origin is at the top left corner, were found. The following is that data:

However, we do not care about the coordinates of every star. We only want to analyze the ones labeled in this image:

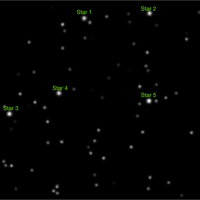


Figure 3. Image with the stars of interest labeled

A black background with white numbers and symbols

Description automatically generatedUsing Matlab’s built-in feature, the coordinates of these stars were approximately found as follows:

A black text on a white background

Description automatically generatedNext, these approximate coordinates were compared to the centroid found earlier to get more exact coordinates. This was done using the following equation:

This yields the following “exact” coordinates of the centroid of each star. These coordinates were overlayed onto the image to verify that we have identified the correct stars and compared to Figure 3.

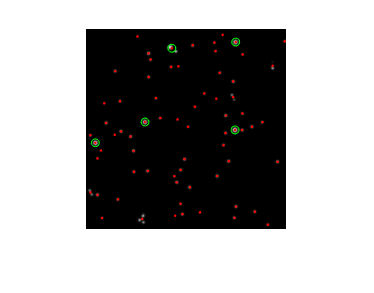


Figure 4. Centroid locations plotted

Table 1. Exact coordinates of the stars of interest

|  |  |  |
| --- | --- | --- |
| Star # | x | y |
| 1 | 86.3143 | 19.8000 |
| 2 | 150.1887 | 13.5660 |
| 3 | 9.7843 | 114.3137 |
| 4 | 59.5000 | 93.5000 |
| 5 | 149.5294 | 101.4314 |

A close up of a number

Description automatically generatedThe next step is to convert these pixel coordinates to actual, physical coordinates in the body frame. This is done using the following equation:

where P is the physical coordinate, W is the width of the image, and FOV is the field of view of the camera (4 degrees in this case). These coordinates were then normalized to become unit vectors. The table below summarizes this data.

Table 2. Body-Frame Coordinates of the stars of interest

|  |  |  |  |
| --- | --- | --- | --- |
| **Star #** |  |  |  |
| 1 | -0.1587 | -0.9298 | 0.332 |
| 2 | 0.4827 | -0.8313 | 0.2754 |
| 3 | -0.9424 | 0.1495 | 0.2991 |
| 4 | -0.8096 | -0.1299 | 0.5724 |
| 5 | 0.8654 | 0.025 | 0.5004 |

***Part B) Inertial Frame Unit Vectors***

The inertial frame unit vectors of these five stars are known and were hard-coded as follows:

Table 3. Inertial-Frame Coordinates of the stars of interest

|  |  |  |  |
| --- | --- | --- | --- |
| **Star #** |  |  |  |
| 1 | -0.9211 | -0.3426 | 0.1851 |
| 2 | -0.4770 | -0.7120 | 0.5154 |
| 3 | -0.4966 | 0.8168 | -0.2937 |
| 4 | -0.7362 | 0.6766 | 0.0104 |
| 5 | 0.3047 | -0.2994 | 0.9042 |

***Part C) Attitude Estimation using TRIAD***

The TRIAD method only uses two measurements to estimate the attitude of the spacecraft. For this, I will choose stars 2 and 3 because they are further apart so precision errors will hopefully not be much of an issue.

Doing the TRIAD method, the DCM that relates the body frame to the inertial frame is as:

Next, the error was calculated using:

Where *b* is the body-frame vectors and *r* is the inertial frame vectors. Taking the magnitude of this error vector yields an error of **0.0158.**

***Part D) Attitude Estimation using OLAE***

The OLAE method uses all the information to estimate the attitude of the spacecraft. From this, the resulting DCM is

and the error is **0.0140**.

***Part E) Attitude Estimation using Davenport q-method***

The Davenport q-method also uses multiple measurements, but is also better adapted for noise in the measurements. Using this metho, the resulting DCM is:

and the error is **0.0140**.

When comparing the error of the three methods, the *TRIAD* method has the worst (largest) error. However, the *OLAE* method has the same error as the *Daveport q-method*. This is expected because the latter two methods benefit from taking additional measurements while the TRIAD only uses two measurements. Evidently, utilizing more data in the calculations leads to more accurate results.

**Problem I (Code)**

% AERSP 450 HW5 - Question I

% Made by Nicholas Luis (PSU ID: 930841391)

% Remove semi-colons to see the output

%% Parts A & B - Basic Data

clc; clear; close all;

% Step 1

grayImg = imread('StarField.png');

imshow(grayImg);

% Step 2

binaryImg = imbinarize(grayImg, 'adaptive'); %Adaptive thresholding

imshow(binaryImg)

% Step 3

stats = regionprops(binaryImg, 'Centroid');

centroids = cat(1, stats.Centroid); %Extract x and y pixel coordinates

% Step 4 : Comparing extracted coords to manually-obtained coords

Star1 = [86,19];

Star2 = [150,14];

Star3 = [10,114];

Star4 = [60,93];

Star5 = [150,101];

starCoords = [Star1; Star2; Star3; Star4; Star5];

Cents = NaN(length(starCoords),2);

for i=1:size(starCoords)

[M,I] = min(((centroids(:,1)-starCoords(i,1)).^2 + (centroids(:,2)-starCoords(i,2)).^2).^(0.5));

Cents(i,:) = centroids(I,:); % Saves new centroid coordinates

end

clear i;

% Step 5: Verifying coordinates

imshow("StarField.png")

hold on

plot(centroids(:,1),centroids(:,2),'.r')

hold on

plot(Cents(:,1),Cents(:,2),'og')

% Step 6: Converting pixel coords into spatial

[L,W] = size(binaryImg); % Gets image dimensions

P = (Cents - W/2) \* tan((4\*pi/180)/2);

P = cat(2,P,ones(5,1));

for j = 1:length(P)

P(j,:) = P(j,:)./norm(P(j,:)); % Normalize vectors to get unit vector

end

clear j;

% (Part B)

StarI1 = [-0.921069884293268 -0.342599924017704 0.185082577005643];

StarI2= [-0.476980639452282 -0.711962047538100 0.515363476056510];

StarI3= [-0.496592767571065 0.816782636490539 -0.293703503424244];

StarI4= [-0.736236774114155 0.676644212846351 0.010393347079969];

StarI5 = [ 0.304668730748568 -0.299446239108458 0.904161995655567];

starCoordsInertial = [StarI1; StarI2; StarI3; StarI4; StarI5];

close all;

%% Part C (TRIAD Method)

% Step 1: Computing DCM

b1 = P(2,:)';

b2 = P(3,:)';

b3 = cross(b1,b2) / norm(cross(b1,b2));

r1 = starCoordsInertial(2,:)';

r2 = starCoordsInertial(3,:)';

r3 = cross(r1,r2) / norm(cross(r1,r2));

B = [b1, b3, cross(b1,b3)];

R = [r1, r3, cross(r1,r3)];

C\_BN\_triad = B\*R';

% Step 2

errorMat\_Triad = NaN(3,length(starCoordsInertial));

for k = 1:length(starCoordsInertial)

errorMat\_Triad(:,k) = P(k,:)' - (C\_BN\_triad\*starCoordsInertial(k,:)');

end

clear k;

errorTriad = norm([errorMat\_Triad(1,:), errorMat\_Triad(2,:), errorMat\_Triad(3,:)]);

%% Part D (OLAE Method)

S\_matrix = NaN(3\*length(P), 3);

D\_vector = NaN(length(P), 3);

for n = 1:length(P)

S = P(n,:)' + starCoordsInertial(n,:)';

D\_vector(n,:) = P(n,:)' - starCoordsInertial(n,:)';

cntr = 1 + 3\*(n-1);

S\_matrix(cntr:3\*n, :) = skewSymmetric(S);

end

clear n; clear cntr;

D\_long = reshape(D\_vector',[],1); % Long vector

q = pinv(S\_matrix)\*D\_long;

% DCM relating body to inertial frame using OLAE method

C\_BN\_olae = ((eye(3) + skewSymmetric(q)) / (eye(3) - skewSymmetric(q)))';

% Calculating error

errorMat\_OLAE = NaN(3,length(starCoordsInertial));

for k = 1:length(starCoordsInertial)

errorMat\_OLAE(:,k) = P(k,:)' - (C\_BN\_olae\*starCoordsInertial(k,:)');

end

clear k;

errorOlae = norm([errorMat\_OLAE(1,:), errorMat\_OLAE(2,:), errorMat\_OLAE(3,:)]);

%% Part E (Davenport q-method)

B\_mat = zeros(3,3);

for i = 1:length(P)

B\_mat = B\_mat + ( P(i,:)' \* starCoordsInertial(i, :) );

end

sigma = trace(B\_mat);

S = B\_mat + B\_mat';

Z = [B\_mat(2,3) - B\_mat(3,2);

B\_mat(3,1) - B\_mat(1,3);

B\_mat(1,2) - B\_mat(2,1); ];

% K matrix

K = [sigma, Z';

Z(1), S(1,:) - [sigma, 0, 0];

Z(2), S(2,:) - [0, sigma, 0];

Z(3), S(3,:) - [0, 0, sigma]; ];

% Eigen-stuff

[eVec, eVal] = eig(K);

[~, I] = max(diag(eVal));

q = eVec(:,I);

% Converting quaternion to DCM

C\_BN\_dave = [ q(1)^2 + q(2)^2 - q(3)^2 - q(4)^2, 2\*(q(2)\*q(3) + q(1)\*q(4)), 2\*(q(2)\*q(4) - q(1)\*q(3)) ;

2\*(q(2)\*q(3) - q(1)\*q(4)), q(1)^2 - q(2)^2 + q(3)^2 - q(4)^2, 2\*(q(3)\*q(4) + q(1)\*q(2)) ;

2\*(q(2)\*q(4) + q(1)\*q(3)), 2\*(q(3)\*q(4) - q(1)\*q(2)), q(1)^2 - q(2)^2 - q(3)^2 + q(4)^2 ;]

% Calculating error

errorMat\_dave = NaN(3,length(starCoordsInertial));

for k = 1:length(starCoordsInertial)

errorMat\_dave(:,k) = P(k,:)' - (C\_BN\_dave\*starCoordsInertial(k,:)');

end

clear k;

errorDave = norm([errorMat\_dave(1,:), errorMat\_dave(2,:), errorMat\_dave(3,:)])

%% Functions

function matrixTilda = skewSymmetric(vec)

% This function inputs a vector and returns a skew symmetrix matrix

matrixTilda = [0, -vec(3), vec(2);

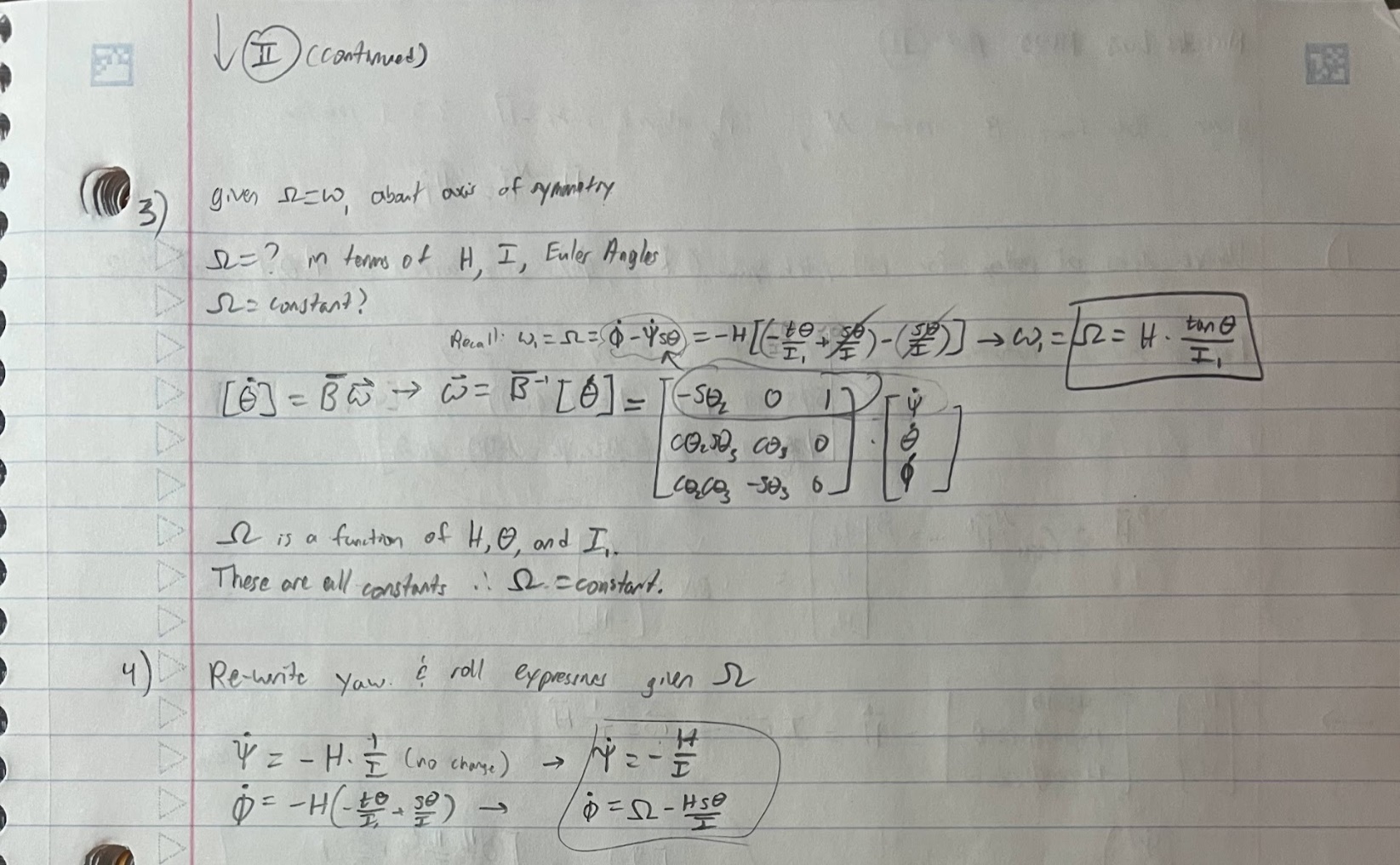
vec(3), 0, -vec(1);

-vec(2), vec(1), 0;];

end

A paper with writing on it

Description automatically generated**Problem II: Attitude Dynamics: Analytical**



**Problem III: Attitude Dynamics: Numerical**

***Part A) Data handling and preliminaries***

1. Given: Measurements of an iPhone  
   Find: Principal moments of inertia assuming rectangular, cuboid shape

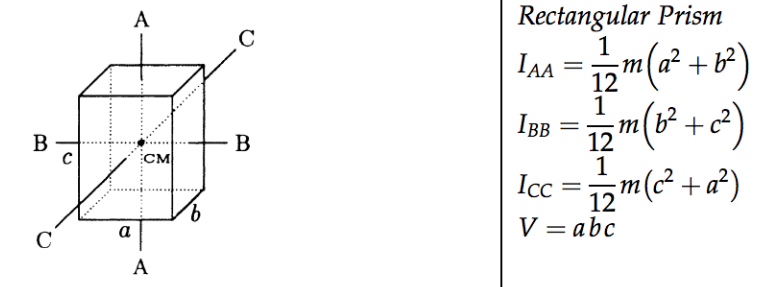
The equations for principal moments of inertia is given by the following:

Figure 5. Moment of inertia for a rectangular prism.

Source: "Mass and Area Moments of Inertia in SOLIDWORKS", wiki.cadcam.com.my/knowledgebase/mass-and-area-moments-of-inertia-in-solidworks/

For consistency, I have set the coordinate *x-y-z* coordinate system in Figure 6 to align with the *A-B-C* system in Figure 5. That means:

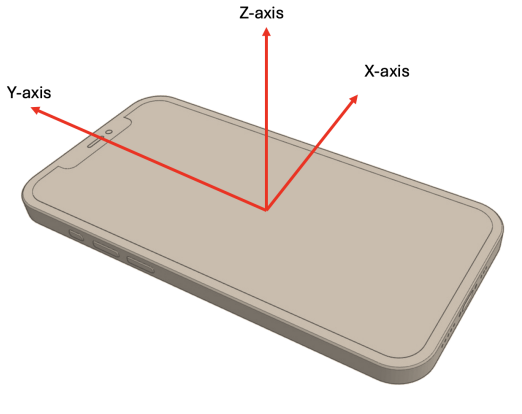


Figure 6. Provided coordinate system of the phone

* a = 71.5 mm
* b = 7.4 mm
* c = 146.7 mm

Given m = 189g, the following are the principal moments of inertia:

1. A simple comparison of the numbers tells us:   
   The y-axis is the minimum moment of inertia axis  
   The x-axis is the intermediate  
   & the z-axis is the maximum
2. Plotting the given angular data yields

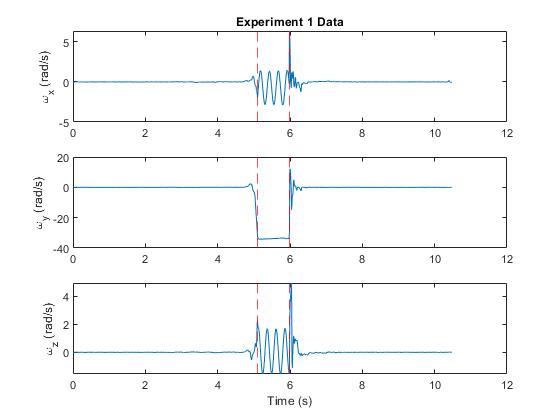


Figure 7. Individual angular velocity components plotted vs time. The red dashed lines indicate the region where the phone was in the air.

Times in which the phone was in the air was identified because, when zoomed in, the angular velocities were sinusoidal. Specifically, **the phone was in the air between 5.10 and 5.98 seconds** for **experiment 1**.

Repeating this process for experiments 2 and 3 yield the following plots:

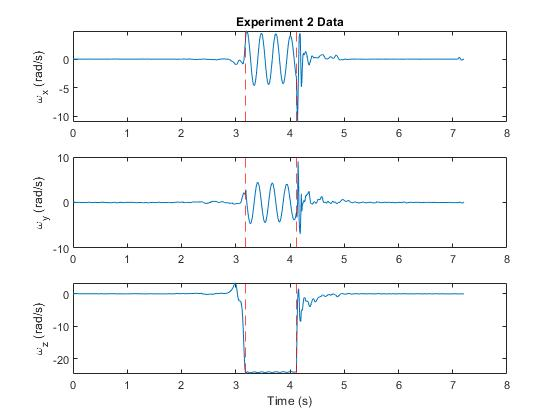


Figure 8. Individual angular velocity components plotted vs time

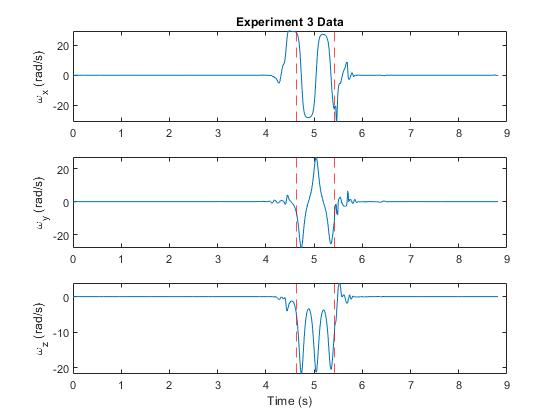


Figure 9. Individual angular velocity components plotted vs time

For **experiment 2, the phone was in the air between and 3.19 and 4.12** seconds.  
For **experiment 3, the phone was in the air between and 4.63 and 5.42** seconds

***Part B) The ‘Fun’ Part***

1. Experiment 1 represents a rotation about the minor axis of inertia, . This is because based on Figure 7, the angular velocity about the y-axis, , was roughly constant. Not only that, but the angular velocity also had a very large magnitude compared to the other subplots.   
     
   Using similar logic: Experiment 2 represents a rotation about the major axis of inertia, . This is because based on Figure 8, the angular velocity about the z-axis, , was roughly constant and very large magnitude.  
     
   Experiment 3 is difficult to determine, but it most likely represents a rotation about the intermediate axis, . Based on Figure 9, the angular momentum about the x-axis, , had a large magnitude. also had a large magnitude, but it was not a sinusoidal shape.
2. Analyzing the energy and angular momentum of the system.

The total energy of the system at any moment in time is given by:

The following is a plot of the total energy of the system as a function of time:

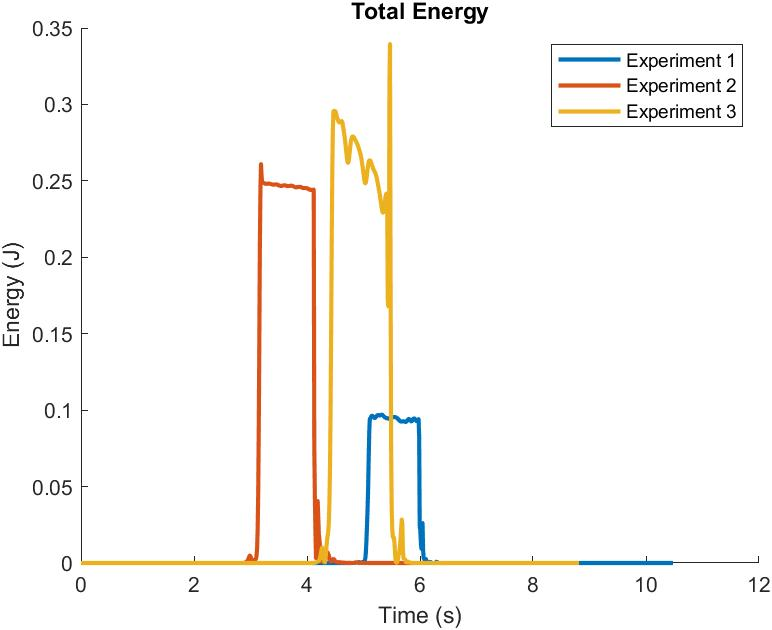


Figure 10. Note that the airtime is when the energy suddenly spikes.

The angular momentum of each component is given by:

The total angular momentum of the system is given by simply taking the magnitude of . The following is a plot of the total angular momentum of the system vs time:

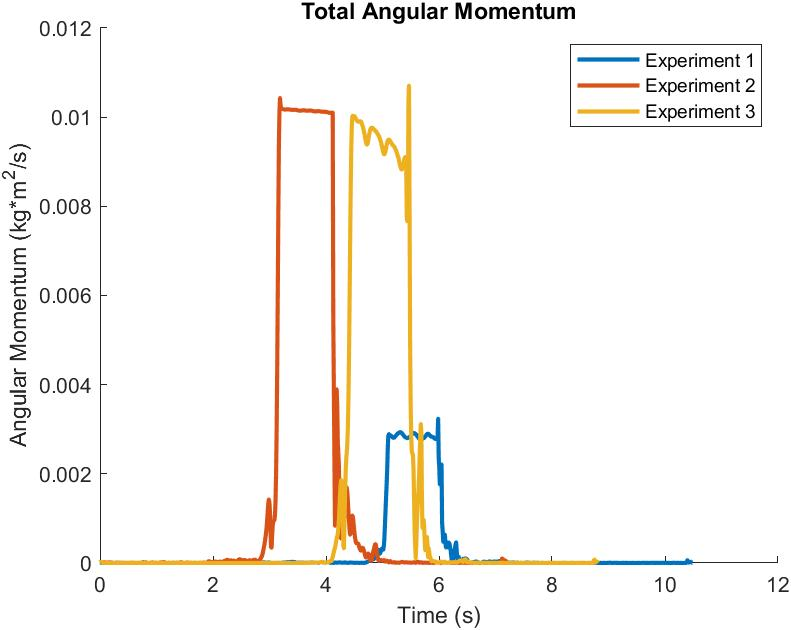
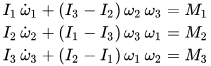


Figure 11. Note that the airtime is when the energy suddenly spikes.

The phone is in the air during the spike that lasts around a second. Even while it is in the air, the energy is not constant. This is especially apparent in Experiment 2, which has a noticeable decrease while it is in the air. Similarly, the angular momentum plots indicate that the angular momentum is not necessarily constant. Thus, **neither energy nor angular momentum are conserved**. In real life, drag from the air causes a torque that slows down the rotation, which appears as a decrease both the angular momentum and energy.

1. Propagating Euler’s Equations of Motion at the beginning of the airtime & comparing it to the experimental results.

The propagation was done using ODE45 in Matlab and the following equations of motion:



Euler’s Equations of Motion. Source: “Euler's equations (rigid body dynamics)”, accessed via Wikipedia.org

Note that these equations are only valid for Torque-Free rotation, which was not the case seen in part 2.

The following are the plots of the propagated data angular velocities compared to the experimental data.

A diagram of a scientific experiment

Description automatically generated with medium confidence

Figure 12 - Minor Axis

A graph of a experiment

Description automatically generated with medium confidence

Figure 13 – Major Axis

A graph of different types of data

Description automatically generated with medium confidence

Figure 14 – Intermediate Axis

When comparing the analytical to experimental results, it is clear that the trends are accurate. However, there is some discrepancies. For example, in Figure 12 (experiment 1), there is a large error between the analytical and the experimental. Figure 13 (experiment 2) shows the opposite of this; The analytical solution starts off very close but drifts over time. The same can be said about Figure 14 (experiment 3). It appears that rotation about the minor axis is the least predictable, but that is to be expected because it is inherently an unstable system; With perturbations (in this case drag), objects will tend to rotate around their major axis.

1. Using the following equation: A math equations with numbers and a plus

   Description automatically generated

Rotation around the minor axis is unstable.

**Problem III (Code)**

% AERSP 450 HW5 - Question III

% Made by Nicholas Luis (PSU ID: 930841391)

% Remove semi-colons to see the output

clear; clc; close all;

%% Part A

% Known properties

h = 146.7; % mm

w = 71.5; % mm

t = 7.4; % mm

m = 189; % g

%Calculate moments of inertia

Ix = (1/12) \* m \* (h^2 + t^2);

Iy = (1/12) \* m \* (w^2 + t^2);

Iz = (1/12) \* m \* (h^2 + w^2);

%load('Experiment1.mat')

% load('Experiment2.mat')

load('Experiment3.mat')

wx = AngularVelocity.X;

wy = AngularVelocity.Y;

wz = AngularVelocity.Z;

T = AngularVelocity.Timestamp;

timeVec = datetime(T, 'InputFormat', 'dd-MMM-yyyy HH:mm:ss.SSS');

t = seconds(timeVec - timeVec(1));

figure(1);

hold on;

subplot(3,1,1);

plot(t, wx);

% title('Experiment 1 Data');

% title('Experiment 2 Data');

title('Experiment 3 Data');

ylabel('\omega\_x (rad/s)');

%xline([5.102, 5.975], '--r'); % For experiment 1

% xline([3.18599, 4.11899], '--r'); % For experiment 2

xline([4.6349, 5.42], '--r'); % For experiment 3

subplot(3,1,2);

plot(t, wy);

ylabel('\omega\_y (rad/s)');

% xline([5.102, 5.975], '--r'); % For experiment 1

% xline([3.18599, 4.11899], '--r'); % For experiment 2

xline([4.6349, 5.42], '--r'); % For experiment 3

subplot(3,1,3);

plot(t, wz);

xlabel('Time (s)');

ylabel('\omega\_z (rad/s)');

% xline([5.102, 5.975], '--r'); % For experiment 1

% xline([3.18599, 4.11899], '--r'); % For experiment 2

xline([4.6349, 5.42], '--r'); % For experiment 3

hold off;

%% Part B

clear; clc; close all;

% VALUES CONVERTED TO METERS AND KILOGRAMS TO GET JOULES DURING ENERGY CALCULATIONS

% Known properties

h = 146.7/1000; % m

w = 71.5/1000; % m

t = 7.4/1000; % m

m = 189/1000; % kg

%Calculate moments of inertia

Ix = (1/12) \* m \* (h^2 + t^2);

Iy = (1/12) \* m \* (w^2 + t^2);

Iz = (1/12) \* m \* (h^2 + w^2);

% Loading Data

load('Experiment1.mat')

wx1 = AngularVelocity.X;

wy1 = AngularVelocity.Y;

wz1 = AngularVelocity.Z;

T1 = AngularVelocity.Timestamp;

timeVec1 = datetime(T1, 'InputFormat', 'dd-MMM-yyyy HH:mm:ss.SSS');

t1 = seconds(timeVec1 - timeVec1(1));

load('Experiment2.mat')

wx2 = AngularVelocity.X;

wy2 = AngularVelocity.Y;

wz2 = AngularVelocity.Z;

T2 = AngularVelocity.Timestamp;

timeVec2 = datetime(T2, 'InputFormat', 'dd-MMM-yyyy HH:mm:ss.SSS');

t2 = seconds(timeVec2 - timeVec2(1));

load('Experiment3.mat')

wx3 = AngularVelocity.X;

wy3 = AngularVelocity.Y;

wz3 = AngularVelocity.Z;

T3 = AngularVelocity.Timestamp;

timeVec3 = datetime(T3, 'InputFormat', 'dd-MMM-yyyy HH:mm:ss.SSS');

t3 = seconds(timeVec3 - timeVec3(1));

% Processing Data

KE1\_hist = NaN(length(t1),1);

W1\_hist = NaN(length(t1),1);

for i = 1:length(t1)

KE1\_hist(i) = (Ix\*wx1(i)^2 + Iy\*wy1(i)^2 + Iz\*wz1(i)^2);

W1\_hist(i) = norm([Ix\*wx1(i), Iy\*wy1(i), Iz\*wz1(i)]);

end

KE2\_hist = NaN(length(t2),1);

W2\_hist = NaN(length(t2),1);

for i = 1:length(t2)

KE2\_hist(i) = (Ix\*wx2(i)^2 + Iy\*wy2(i)^2 + Iz\*wz2(i)^2);

W2\_hist(i) = norm([Ix\*wx2(i), Iy\*wy2(i), Iz\*wz2(i)]);

end

KE3\_hist = NaN(length(t3),1);

W3\_hist = NaN(length(t3),1);

for i = 1:length(t3)

KE3\_hist(i) = (Ix\*wx3(i)^2 + Iy\*wy3(i)^2 + Iz\*wz3(i)^2);

W3\_hist(i) = norm([Ix\*wx3(i), Iy\*wy3(i), Iz\*wz3(i)]);

end

% Plotting

figure(2)

hold on

plot(t1, KE1\_hist, LineWidth=2)

plot(t2, KE2\_hist, LineWidth=2)

plot(t3, KE3\_hist, LineWidth=2)

legend('Experiment 1 ', 'Experiment 2', 'Experiment 3')

title('Total Energy')

xlabel("Time (s)")

ylabel("Energy (J)")

hold off

exportgraphics(gca,"HW5\_Problem3\_Energy.jpg");

figure(3)

hold on

plot(t1, W1\_hist, LineWidth=2)

plot(t2, W2\_hist, LineWidth=2)

plot(t3, W3\_hist, LineWidth=2)

legend('Experiment 1 ', 'Experiment 2', 'Experiment 3')

title('Total Angular Momentum')

xlabel("Time (s)")

ylabel("Angular Momentum (kg\*m^2/s)")

hold off

exportgraphics(gca,"HW5\_Problem3\_Momentum.jpg");

% The following are the regions of times that the phone is in the air (from part A)

times = [[5.102, 5.975];

[3.18599, 4.11899];

[4.6349, 5.42]];

% ---------Experiment 1 Progagation---------

t\_range = times(1,:);

% Initial conditions

index = min(find(t1 >= 5.102));

wx1\_0 = wx1(index);

wy1\_0 = wy1(index);

wz1\_0 = wz1(index);

w1\_0 = [wx1\_0, wy1\_0, wz1\_0]; % Vector of initial conditions for experiment 1

EOMs = @(t, omega) [ ((Iy - Iz) / Ix) \* omega(2) \* omega(3);

((Iz - Ix) / Iy) \* omega(3) \* omega(1);

((Ix - Iy) / Iz) \* omega(1) \* omega(2)];

% Progation using ode45

[t1\_prop, omega1\_prop] = ode45(EOMs, t\_range, w1\_0);

% Plotting

figure(4);

subplot(3,1,1);

plot(t1\_prop, omega1\_prop(:,1), '--r');

hold on;

title('Experiment 1 Propagated Data');

plot(t1(index:index+length(t1\_prop),:), wx1(index:index+length(t1\_prop),:), 'b');

hold off;

legend('Analytical', 'Experimental')

ylabel('\omega\_x (rad/s)');

subplot(3,1,2);

plot(t1\_prop, omega1\_prop(:,2), '--r');

hold on;

plot(t1(index:index+length(t1\_prop),:), wy1(index:index+length(t1\_prop),:), 'b');

hold off;

legend('Analytical', 'Experimental')

ylabel('\omega\_x (rad/s)');

subplot(3,1,3);

plot(t1\_prop, omega1\_prop(:,3), '--r');

hold on;

plot(t1(index:index+length(t1\_prop),:), wz1(index:index+length(t1\_prop),:), 'b');

hold off;

legend('Analytical', 'Experimental')

ylabel('\omega\_x (rad/s)');

xlabel('Time (s)');

% ---------Experiment 2 Progagation---------

t\_range = times(2,:);

% Initial conditions

index = min(find(t1 >= 3.18599));

wx2\_0 = wx2(index);

wy2\_0 = wy2(index);

wz2\_0 = wz2(index);

w2\_0 = [wx2\_0, wy2\_0, wz2\_0]; % Vector of initial conditions for experiment 1

EOMs = @(t, omega) [ ((Iy - Iz) / Ix) \* omega(2) \* omega(3);

((Iz - Ix) / Iy) \* omega(3) \* omega(1);

((Ix - Iy) / Iz) \* omega(1) \* omega(2)];

% Progation using ode45

[t2\_prop, omega2\_prop] = ode45(EOMs, t\_range, w2\_0);

% Plotting

figure(5);

subplot(3,1,1);

plot(t2\_prop, omega2\_prop(:,1), '--r');

hold on;

title('Experiment 2 Propagated Data');

plot(t2(index:index+length(t1\_prop),:), wx2(index:index+length(t1\_prop),:), 'b');

hold off;

legend('Analytical', 'Experimental')

ylabel('\omega\_x (rad/s)');

subplot(3,1,2);

plot(t2\_prop, omega2\_prop(:,2), '--r');

hold on;

plot(t2(index:index+length(t1\_prop),:), wy2(index:index+length(t1\_prop),:), 'b');

hold off;

legend('Analytical', 'Experimental')

ylabel('\omega\_x (rad/s)');

subplot(3,1,3);

plot(t2\_prop, omega2\_prop(:,3), '--r');

hold on;

plot(t2(index:index+length(t1\_prop),:), wz2(index:index+length(t1\_prop),:), 'b');

hold off;

legend('Analytical', 'Experimental')

ylabel('\omega\_x (rad/s)');

xlabel('Time (s)');

% ---------Experiment 3 Progagation---------

t\_range = times(3,:);

% Initial conditions

index = min(find(t1 >= 4.6349));

wx3\_0 = wx3(index);

wy3\_0 = wy3(index);

wz3\_0 = wz3(index);

w3\_0 = [wx3\_0, wy3\_0, wz3\_0]; % Vector of initial conditions for experiment 1

EOMs = @(t, omega) [ ((Iy - Iz) / Ix) \* omega(2) \* omega(3);

((Iz - Ix) / Iy) \* omega(3) \* omega(1);

((Ix - Iy) / Iz) \* omega(1) \* omega(2)];

% Progation using ode45

[t3\_prop, omega3\_prop] = ode45(EOMs, t\_range, w3\_0);

% Plotting

figure(6);

subplot(3,1,1);

plot(t3\_prop, omega3\_prop(:,1), '--r');

hold on;

title('Experiment 3 Propagated Data');

plot(t3(index:index+length(t1\_prop),:), wx3(index:index+length(t1\_prop),:), 'b');

hold off;

legend('Analytical', 'Experimental')

ylabel('\omega\_x (rad/s)');

subplot(3,1,2);

plot(t3\_prop, omega3\_prop(:,3), '--r');

hold on;

plot(t3(index:index+length(t1\_prop),:), wy3(index:index+length(t1\_prop),:), 'b');

hold off;

legend('Analytical', 'Experimental')

ylabel('\omega\_x (rad/s)');

subplot(3,1,3);

plot(t3\_prop, omega3\_prop(:,2), '--r');

hold on;

plot(t3(index:index+length(t1\_prop),:), wz3(index:index+length(t1\_prop),:), 'b');

hold off;

legend('Analytical', 'Experimental')

ylabel('\omega\_x (rad/s)');

xlabel('Time (s)');

%% Final Part

% Plotting ellipsoid

% Compute semi-axes of the ellipsoid from equation above

Figure(13);

hold on;

a =

b =

c =

% Create the ellipsoid

[x, y, z] = ellipsoid(0, 0, 0, a, b, c);

surf(x, y, z, 'FaceColor', [0.5, 0.5, 0.5], 'EdgeColor',...

'y', 'FaceAlpha', 0.5); % Uniform gray