**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.

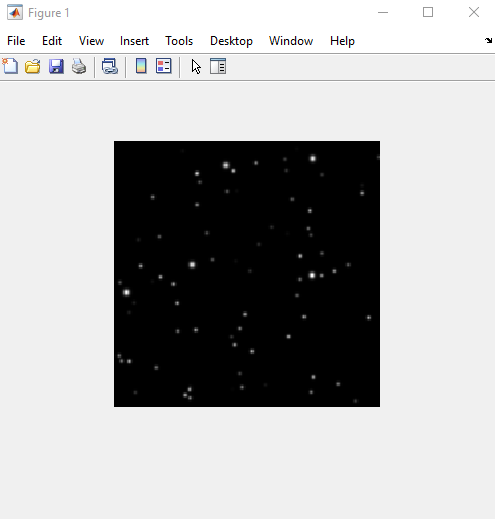


Figure . Raw image

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

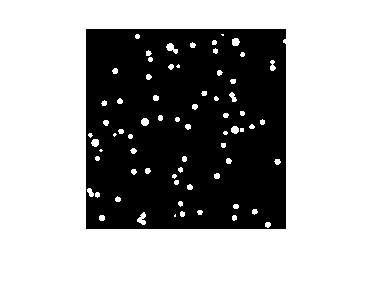


Figure . Identified stars

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:

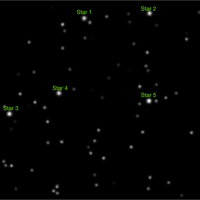


Figure . Image with the stars of interest labeled

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.

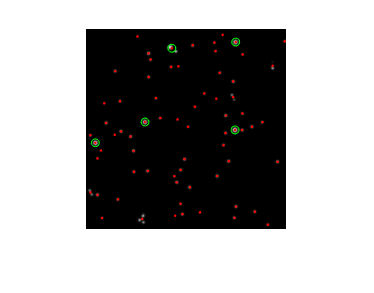


Figure . Centroid locations plotted

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 1

% 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

**Problem II: Attitude Dynamics: Analytical**

**Given:**

**To Find:**

***Part 1)***