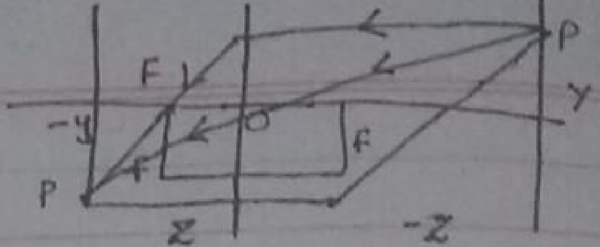
ROBOTICS HOMEWORK 4



Question 1

Solution

Given

$Z = 12$ inches

$y = 40$ Pixels

$z = ?$

By using the len's formula

$$\frac{1}{f} = \frac{1}{z} + \frac{1}{Z}$$

$\because$ 1inch = 100 pixel $= Y$

By using similar triangle formula

$$\rightarrow \frac{Y}{f} = \frac{y}{Z-f}$$

$$\rightarrow \frac{100}{f} = \frac{40}{Z-f}$$

$$\rightarrow f = \frac{5}{7} Z$$

Now Put the value of f in len's formula

$$\rightarrow \frac{1}{f} = \frac{1}{Z} + \frac{1}{Z}$$

$$\rightarrow \frac{1}{\frac{5}{7}Z} = \frac{1}{z} + \frac{1}{Z}$$

$$\rightarrow \frac{7}{5Z} - \frac{1}{Z} = \frac{1}{12}$$

$$\rightarrow Z = 4.8 \text{ inches}$$

Answer

Question # 3

Solution

$$^{1}T_{2} = \begin{bmatrix} 1 & 0 & 0 & B \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Now

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & B \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{bmatrix}$$

$x_1 = x_2 + B$

$y_1 = y_2$

$z_1 = z_2$

$\rightarrow \quad v_1 = f_1 \dfrac{x_1}{z_1} \qquad v_2 = f_2 \dfrac{x_2}{z_2}$

$\rightarrow \quad v_1 = \dfrac{f_1 (x_2 + B)}{z_1} \qquad x_2 = \dfrac{v_2 z_2}{f_2}$

Now subtitute x_2 in "v_1" formula for Depth of field

$$v_1 = f_1 \left(\frac{v_2 z_2}{f_2} + B \right) \Big/ z_1$$

$$z_1 v_1 = f_1 \left(\frac{v_2 z_1 + f_1 B f_2}{f_2} \right)$$

$$z_1 v_1 f_2 = f_1 v_2 z_1 + f_1 B f_2$$

$\because \quad z = z_1 = z_2$

$\rightarrow \quad z\, v_1 f_2 - f_1 v_2 z = f_1 B f_2$

$\rightarrow \quad z = f_1 B f_2 / v_1 f_2 - f_1 v_2$

Answer

Q2 Part a

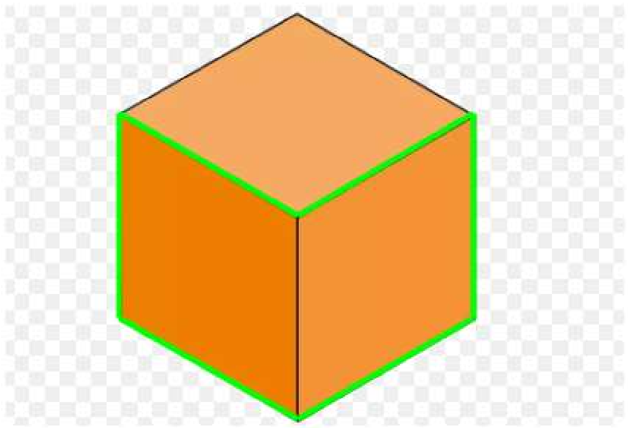
```
% Read the image "hw4cube.png" and store it in the variable "image"
image = imread("hw4cube.png");

% Define the 3D coordinates of the cube vertices in the world coordinate system
% by specifying the x, y, and z coordinates of each vertex
% and store them in the variables a, b, c, d, e, and f.
% The vertices are named in alphabetical order.
a = [78, 74];
b = [200, 144];
c = [320, 74];
d = [320, 214];
e = [200, 283];
f = [78, 214];

% Define the 2D coordinates of the six corners of the cube in the image
% by specifying their x and y coordinates and store them in the variables u and v.
% The corners are named in alphabetical order.
u = [78, 200, 320, 320, 200, 78];
v = [74, 144, 74, 214, 283, 214];

% Define the 3D coordinates of the cube vertices in the world coordinate system
% by specifying the x, y, and z coordinates of each vertex
% and store them in the variable world_coor as a 3 x 6 matrix.
world_coor = [0, 0, 2, 2, 0, 0;
0, 2, 2, 2, 2, 0;
2, 2, 2, 0, 0, 0];

% Define the matrix A that relates the 2D coordinates of the corners of the cube in the image
% to the 3D coordinates of the vertices of the cube in the world coordinate system.
% Then, use the null function to find the null space of A, which gives the solution for x
% such that A*x = 0.
mat = zeros(12,12);

i = 1;

for j = 1:2:11
mat(j,:) = [world_coor(1,i), world_coor(2,i), world_coor(3,i), 1, 0, 0, 0, 0, -u(i)*world_coor(1,i), ...
    -u(i)*world_coor(2,i), -u(i)*world_coor(3,i), -u(i)];
mat(j+1,:) = [0, 0, 0, 0, world_coor(1,i), world_coor(2,i), world_coor(3,i), 1, -v(i)*world_coor(1,i), ...
    -v(i)*world_coor(2,i), -v(i)*world_coor(3,i), -v(i)];
i = i + 1;
end

null_space = null(mat);
x = null_space(:,1);

% Reshape the solution x into a 3 x 4 matrix M, which represents the 3 x 4 projection matrix
% that maps 3D coordinates in the world coordinate system to 2D coordinates in the image plane.
M = reshape(x, [4, 3])'
```

```
M = 3×4

   -0.2297   -0.2391    0.0000   -0.3022
    0.1376   -0.1376    0.2712   -0.8291
    0.0000   -0.0000    0.0000   -0.0039
```

Q2 Part b

```
% Compute the 2D coordinates of the cube vertices by applying the projection matrix M
% to the 3D coordinates of the vertices in the world coordinate system.
% Then, convert the homogeneous coordinates to Euclidean coordinates by dividing the first two
% columns of the resulting matrix by the third column.
P = world_coor';
P_ = [P, ones(size(P,1),1)];
homo_image = P_ * M';

P_final = homo_image(:,1:2) ./ homo_image(:,3);

%drawing the edges of the cube
figure;
```

```
imshow(image);
hold on;

edges = [1 2; 2 3; 3 4; 4 5; 5 6; 6 1]; % Define edges of the cube as pairs of vertex indices
num_edges = size(edges, 1); % Get the number of edges

% Draw each edge in the image
for i = 1:num_edges
    v1_idx = edges(i, 1); % Get index of first vertex of edge
    v2_idx = edges(i, 2); % Get index of second vertex of edge
     % Draw line between the two vertices
    line([P_final(v1_idx, 1) P_final(v2_idx, 1)], [P_final(v1_idx, 2) P_final(v2_idx, 2)], 'Color', 'green', 'LineWidth', 3);
end
```



Q4

Explanation

This code works according to the triangulation method, which works by first constructing the skew-symmetric matrices(skew1 and skew2) for each 2D point in

pts1 and pts2, respectively. These matrices are used to construct the linear system of equations A that relates the 3D point P to its corresponding 2D points in

the two images. Specifically, A is defined as [skew1M1; skew2M2], where M1 and M2 are the camera projection matrices. The SVD is used to solve a linear

system and find the null space of matrix A. The last column of the V matrix is used to get a 3D point P, which is then converted into Cartesian coordinates by

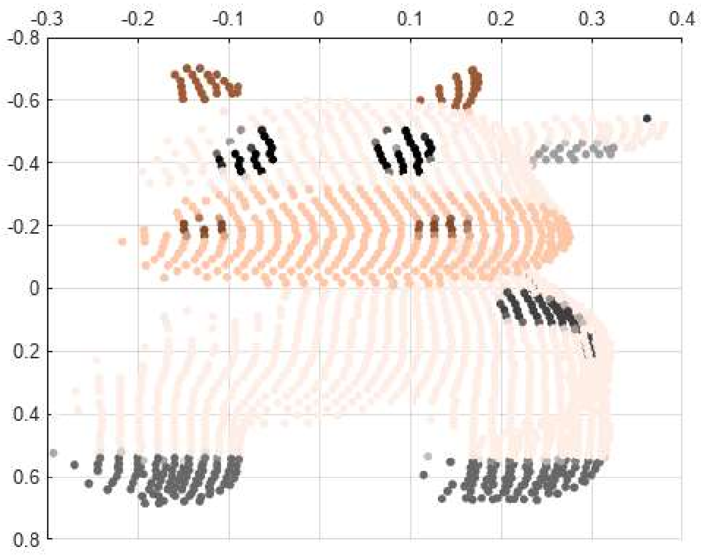
dividing by its homogeneous coordinate. By averaging the colours of the 3D point's corresponding 2D points in the first image, we determine the colour of the

3D point. Using scatter3, we then plot the point cloud.

Code

```
load("triangulation.mat")
num_pts = length(pts2);
point_color = zeros(num_pts, 3); % initialize color array
points_3d = zeros(num_pts, 3); % initialize 3D points array
im1 = imread("moo1.jpg");

% Triangulate 3D points and their corresponding colors
for i = 1:num_pts % loop through all matched point pairs
% Get corresponding points in image 1 and image 2
point1 = pts1(i,:);
point2 = pts2(i,:);
% Construct skew-symmetric matrices
skew1 = double([0 -1 point1(2);         1 0 -point1(1);         -point1(2) point1(1) 0]);
skew2 = double([0 -1 point2(2);         1 0 -point2(1);         -point2(2) point2(1) 0]);

% Construct the linear system to solve for 3D point
```

```
A = [skew1 * M1; skew2 * M2];

% Solve the linear system using SVD
[~,~,V] = svd(A);
P = V(:,end); % get null space of A
P = P ./ P(4); % divide by homogeneous coordinate
point_3d = P(1:3)'; % store 3D point

% Get color of the 3D point by taking the average color of its corresponding 2D points

[x] =im1(pts1(i,2), pts1(i,1),: );
%point_colors(i,:) = double(x, [], 1)'/255;
point_color(i,:) = double(reshape(x, [],1)')/255;

% Store 3D point and its color
points_3d(i,:) = point_3d;
end

% Plot the 3D points
figure;
scatter3(points_3d(:,1), points_3d(:,2), points_3d(:,3), 20, point_color, "filled");
view([-0.044921875,-88.091796875]);

view([0.355 -90.000])
```



Q5 Part a

```
img = imread('objects.png'); % read in image and store in variable 'img'

grayImg = rgb2gray(img); % convert RGB image to grayscale and store in variable 'grayImg'

threshold = 0.1; % set threshold level to 0.1

bw = imbinarize(grayImg, threshold); % create binary image using threshold level and store in variable 'bw'

cc = bwconncomp(bw); % find connected components in binary image and store in variable 'cc'

num_objects = cc.NumObjects; % retrieve number of objects from 'cc' structure and store in variable 'num_objects'

disp(['There are ', num2str(num_objects), ' objects in the image.']); % print number of objects to command window
```

```
There are 13 objects in the image.
```

Q5 Part b

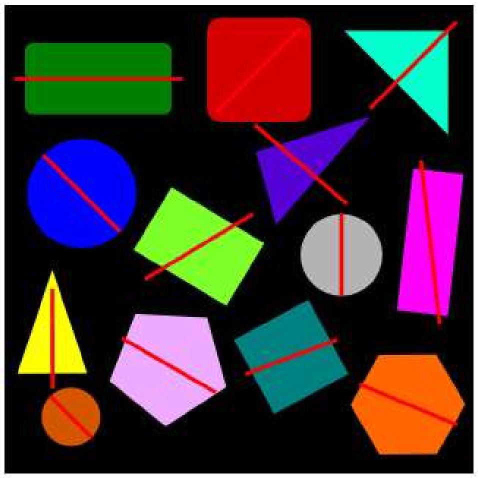
```
img = imread('objects.png'); % read in image and store in variable 'img'

grayImg = rgb2gray(img); % convert RGB image to grayscale and store in variable 'grayImg'

threshold = 0.1; % set threshold level to 0.1
```

```
bw = imbinarize(grayImg, threshold); % create binary image using threshold level and store in variable 'bw'
labelImg = bwlabel(bw);

% Get region properties for each object
props = regionprops(labelImg, 'BoundingBox');

% Extract bounding box coordinates for each object
numObjects = length(props);
for i = 1:numObjects
    boxCoords = props(i).BoundingBox;
    disp(['Object ', num2str(i), ' at position (', num2str(boxCoords(1)), ',', num2str(boxCoords(2)), ')']);
end
```

```
Object 1 at position (9.5,181.5)
Object 2 at position (14.5,26.5)
Object 3 at position (16.5,92.5)
Object 4 at position (25.5,261.5)
Object 5 at position (71.5,211.5)
Object 6 at position (88.5,124.5)
Object 7 at position (138.5,9.5)
Object 8 at position (157.5,201.5)
Object 9 at position (172.5,76.5)
Object 10 at position (202.5,142.5)
Object 11 at position (232.5,17.5)
Object 12 at position (236.5,238.5)
Object 13 at position (268.5,112.5)
```

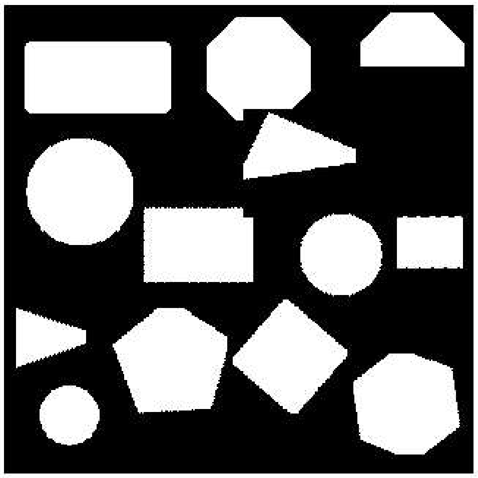
Q5 Part c

```
% Load image
img = imread('objects.png'); % read in image and store in variable 'img'

% Convert image to grayscale
grayImg = rgb2gray(img);

% Specify threshold level
thresholdLevel = 0.1;

% Threshold image to create binary image
binImg = imbinarize(grayImg, thresholdLevel);

% Label connected components in binary image
labelImg = bwlabel(binImg);

% Get region properties for each object
props = regionprops(labelImg, 'Centroid', 'MajorAxisLength', 'MinorAxisLength', 'Orientation');

% Display image
figure;
imshow(img);
hold on;

% Draw line along elongated side of each object
numObjects = length(props);
for i = 1:numObjects
    center = props(i).Centroid;
    majorAxisLength = props(i).MajorAxisLength;
    minorAxisLength = props(i).MinorAxisLength;
    orientation = -props(i).Orientation;
    x1 = center(1) + 0.5*majorAxisLength*cosd(orientation);
    y1 = center(2) - 0.5*majorAxisLength*sind(orientation);
    x2 = center(1) - 0.5*majorAxisLength*cosd(orientation);
    y2 = center(2) + 0.5*majorAxisLength*sind(orientation);
    line([x1,x2], [y1,y2], 'LineWidth', 2, 'Color', 'r');
end

% Turn off hold
hold off;
```



Q5 Part d

```
% Load image
img = imread('objects.png'); % read in image and store in variable 'img'

% Convert image to grayscale
grayImg = rgb2gray(img);

% Specify threshold level
thresholdLevel = 0.1;

% Threshold image to create binary image
binImg = imbinarize(grayImg, thresholdLevel);

% Label connected components in binary image
labelImg = bwlabel(binImg);

% Get region properties for each object
props = regionprops(labelImg, 'Centroid', 'Orientation', 'Image');

% Create new image with black background
newImg = zeros(size(img,1), size(img,2));

% Rotate each object and place it in new image at same location as centroid
numObjects = length(props);
for i = 1:numObjects
    centroid = props(i).Centroid;
    orientation = -props(i).Orientation;
    rotatedImg = imrotate(props(i).Image, orientation, 'bicubic', 'crop');
    [h, w] = size(rotatedImg);
    x1 = round(centroid(1) - w/2);
    x2 = x1 + w - 1;
    y1 = round(centroid(2) - h/2);
    y2 = y1 + h - 1;
    if x1 >= 1 && x2 <= size(newImg,2) && y1 >= 1 && y2 <= size(newImg,1)
        newImg(y1:y2, x1:x2) = rotatedImg;
    end
end

% Display new image
figure;
imshow(newImg);
```



Q5 Part e

1. Edge detection: Edge detection algorithms such as Canny, Sobel, or Prewitt can be used to detect the edges of the object. Once the edges are detected, the shape of the object can be determined by analyzing the edge data. For example, the number of corners or the curvature of the edges can be used to identify the shape of the object.
2. Contour detection: Contour detection algorithms can be used to detect the contours of the object. Once the contours are detected, the shape of the object can be determined by analyzing the contour data. For example, the number of corners or the curvature of the contour can be used to identify the shape of the object.
3. Shape descriptors: Shape descriptors such as Fourier descriptors, Hu moments, and Zernike moments can be used to extract features that describe the shape of the object. These features can be used to compare and classify different shapes.
4. Template matching: If the shape of the object is known beforehand, a template of the shape can be created and used to match and identify the object in the image. This can be done by correlating the template with the image or by using methods such as the Hough transform.
5. Machine learning: Machine learning techniques such as deep learning can be used to train a model to recognize and classify different shapes in the image. The model can be trained using labeled data and can be used to detect the shape of objects in new images.

Q6

Rida: This homework took me approximately 6 to 7 hours to complete over a span of 3 days including understanding the questions, trying themselves out and also asking for help where needed. My part in this homework was to complete questions 2 and 4. Initially the questions

looked a bit difficult but after understanding and getting the idea behind it, they were easier to implement. There were a lot of new commands

I learned like the SVD, reshape and scatter3. I also learned how to get a 3D point cloud from 2D images and how can we get the camera projection matrix from a given picture.

Hussain: My part was to complete question 1, 3 and 5 in the assignment. It took me about 5 hours to complete my part. I have learned that in

computer vision, segmentation is the process of dividing an image into different regions based on their visual characteristics such as color,

texture, or brightness. This can be done using various techniques such as thresholding, edge detection, region growing, clustering, and more.

After segmenting the image, region property extraction is performed to extract various properties of each region such as area, perimeter, centroid, bounding box, orientation, moments, texture, and color. These properties provide useful information about the objects in the image

and their spatial relationships. Overall, I have gained an understanding of the importance of segmentation and region property extraction in

computer vision, and how they can be used to extract useful information from images for a variety of applications such as object detection,

recognition, and tracking.