Lab Task: Camera Projection Matrix (P Matrix) Estimation

Objective: The objective of this lab is to estimate the camera projection matrix (P Matrix) using images and corresponding 3D world coordinates. You will experiment by increasing the number of points and use your own images taken from your mobile camera to perform the estimation. You will also explore the visualization of the computed camera parameters and understand the effect of using different numbers of control points.

Instructions:

Part 1: P Matrix Estimation Using Provided Code

1. Use the provided code to estimate the P matrix using the given image and point data.
2. Carefully read through and run the code, making sure you understand each step.
3. Save the following results and visualizations in a PDF for submission:
   - The computed camera matrix (P), intrinsic matrix (K), and rotation matrix (R).
   - The 3D plot shows the camera center, the world points, and the principal axis.
   - The image with projected 3D points and vanishing lines.

Part 2: Using Your Own Image from your camera for P Matrix Estimation

1. Take a new image using your phone or another camera. Make sure to capture a scene with clearly identifiable points, ideally in a structured environment like a room or building corner.
2. Replace the image path in the code with the path to your new image.
3. Select a total of 24 points on the image that can be accurately mapped to 3D world coordinates. Enter these coordinates as prompted.
4. Run the code to compute the P matrix for your image.
5. Save the following in your PDF submission:
   - The image with the selected points plotted and labeled.
   - The computed camera matrix (P), intrinsic matrix (K), and rotation matrix (R).
   - The 3D visualization of the points and camera center.
   - The image with projected 3D points, including back-projected points and points at infinity.

Part 3: Experiment and Reflect

1. Experience with different numbers of control points, such as using 12, 24, and 40 points for P estimation.
2. Reflect on the following questions:
   - How does increasing the number of points affect the accuracy and stability of the P matrix estimation?

    - Is there a noticeable difference in the accuracy of the back-projection when using fewer points versus more points?
    - What challenges did you encounter when manually selecting points and entering 3D world coordinates?
    - Include your reflections in your PDF report.

Submission Requirements:

- A single PDF document containing:
    - Screenshots of the outputs for each part.
    - Reflections on the impact of increasing the number of control points.
    - Answers to the reflection questions.

Evaluation Criteria:

- Demonstration of understanding through reflections and analysis.
- Clarity and completeness of your PDF submission.

Optional Challenge:

- Take two different images of the same scene from different angles. Estimate the P matrix for both images and compare the results. Are the intrinsic matrix (K) values consistent between the two images? How does the rotation matrix (R) differ? Include any observations in your report.

```
close all
clear all
clc
% Prompt user to load data or select points manually
choice = menu('Do you want to load points from the workspace_variables.mat file?', 'Yes', 'No');

if choice == 1
    % Load points from .mat file
    load('workspace_variables.mat');

    % Ensure the necessary variables exist in the loaded file
    if exist('image_points', 'var') && exist('world_points', 'var')
        disp('Points successfully loaded from workspace_variables.mat');
    else
        error('The loaded file does not contain the required variables "image_points" and "world_points".');
    end
else
    % Proceed with manual selection if user chooses "No"

    % Load and display the image
    img = imread('20241104_110446.jpg');  % Replace with your image file path
    imshow(img);  % Display the image
    hold on;  % Keep the image displayed while adding markers

    % Initialize matrices to store 2D and 3D homogeneous coordinates
    num_points = 36;  % Set number of points
    image_points = zeros(num_points, 3);  % For 2D image points in homogeneous coordinates
    world_points = zeros(num_points, 4);  % For 3D world points in homogeneous coordinates

    disp('Click on the image to select 96 points and enter their 3D world coordinates.');

    for i = 1:num_points
        % Use ginput to get one image point at a time
        [x, y] = ginput(1);

        % Store the 2D image coordinates in homogeneous form
        image_points(i, :) = [x, y, 1];

        % Plot the selected point on the image
        plot(x, y, 'r+', 'MarkerSize', 8, 'LineWidth', 1.5);

        % Display the current point number for reference
        text(x, y, sprintf('  %d', i), 'Color', 'yellow', 'FontSize', 10, 'FontWeight', 'bold');

        % GUI prompt for 3D world coordinates
        prompt = sprintf('Enter the 3D world coordinates for point %d (X, Y, Z):', i);
        title = '3D World Coordinate Input';
        dims = [1];  % Set single row dimension for the input dialog
        default_input = {'0', '0', '0'};  % Default values for input fields
        answer = inputdlg({'X:', 'Y:', 'Z:'}, title, dims, default_input);

        % Convert input to numeric values and store in homogeneous form
        if ~isempty(answer)
            world_coords = str2double(answer);  % Convert cell array of strings to numeric array
            world_points(i, :) = [world_coords', 1];  % Store in homogeneous form [X Y Z 1]
        else
            error('3D coordinate input was canceled.');  % Handle case if input is canceled
        end
    end

    % Release the hold on the image display
    hold off;
end

% Construct the matrix A for solving P using SVD
A = [];
for i = 1:num_points
    X = world_points(i, :);      % Homogeneous 3D coordinates [X Y Z 1]
    x = image_points(i, 1);      % x-coordinate of the 2D image point
    y = image_points(i, 2);      % y-coordinate of the 2D image point

    % Build the two rows for each point
    row1 = [zeros(1, 4), -X, y * X];
    row2 = [X, zeros(1, 4), -x * X];

    % Append the rows to A
    A = [A; row1; row2];
end

% Compute the camera projection matrix P using SVD
[~, ~, V] = svd(A);
P = reshape(V(:, end), 4, 3)';  % Reshape the last column of V into a 3x4 matrix

% Compute the camera center as the null space of P
[~, ~, V_P] = svd(P);
camera_center_h = V_P(:, end);  % Last column of V
camera_center = camera_center_h(1:3) / camera_center_h(4);  % Convert from homogeneous

% Decompose P into K and R using QR decomposition on the first 3x3 part
M = P(:, 1:3);  % The 3x3 submatrix of P
[Q, R] = qr(inv(M));  % Use QR decomposition
K = inv(R);           % Intrinsic matrix
K = K / K(3,3);       % Normalize K so that K(3,3) is 1
R = inv(Q);           % Rotation matrix

% Display computed matrices
disp('Computed camera matrix P:');
disp(P);
disp('Intrinsic matrix K:');
disp(K);
disp('Rotation matrix R:');
disp(R);
disp('Camera center (in world coordinates):');
disp(camera_center);

% Visualization in 3D
figure 1;
scatter3(world_points(:,1), world_points(:,2), world_points(:,3), 'bo'); % Plot 3D points
hold on;
scatter3(camera_center(1), camera_center(2), camera_center(3), 'ro', 'filled');  % Plot camera center
quiver3(camera_center(1), camera_center(2), camera_center(3), -R(3,1), -R(3,2), -R(3,3), 10, 'r');  % Plot principal axis
```

```
%title('3D Points, Camera Center, and Principal Axis');
xlabel('X');
ylabel('Y');
zlabel('Z');
legend('3D Points', 'Camera Center', 'Principal Axis');
grid on;
hold off;

% Back-project the 3D points onto the 2D image using the camera matrix P
projected_points = P * world_points';

% Convert from homogeneous coordinates to 2D by normalizing
projected_points = projected_points ./ projected_points(3, :);

% Display the original image and overlay the back-projected points

% Assuming P is already computed and available

% Define the points at infinity along the axes
D_x = [1; 0; 0; 0];  % Point at infinity along X-axis
D_y = [0; 1; 0; 0];  % Point at infinity along Y-axis
D_z = [0; 0; 1; 0];  % Point at infinity along Z-axis
D_o = [0; 0; 0; 1];  % World origin

% Project these points using the camera projection matrix P
image_point_infinity_x = P * D_x;
image_point_infinity_y = P * D_y;
image_point_infinity_z = P * D_z;
image_point_origin = P * D_o;

% Normalize the projected points to get 2D coordinates
image_point_infinity_x = image_point_infinity_x ./ image_point_infinity_x(3);
image_point_infinity_y = image_point_infinity_y ./ image_point_infinity_y(3);
image_point_infinity_z = image_point_infinity_z ./ image_point_infinity_z(3);
image_point_origin = image_point_origin ./ image_point_origin(3);


% Display the original image and overlay the points at infinity
figure(2);
imshow(img);
hold on;

% Plot the actual image points in red
scatter(image_points(:, 1), image_points(:, 2), 'ro', 'filled', 'DisplayName', 'Actual Image Points');

% Plot back-projected points in green
scatter(projected_points(1, :), projected_points(2, :), 'gx', 'filled', 'DisplayName', 'Back-Projected 3D Points');

% Plot points at infinity with different markers and colors
scatter(image_point_infinity_x(1), image_point_infinity_x(2), 'bx', 'LineWidth', 2, 'DisplayName', 'Point at Infinity (X-axis)');
scatter(image_point_infinity_y(1), image_point_infinity_y(2), 'gx', 'LineWidth', 2, 'DisplayName', 'Point at Infinity (Y-axis)');
scatter(image_point_infinity_z(1), image_point_infinity_z(2), 'mx', 'LineWidth', 2, 'DisplayName', 'Point at Infinity (Z-axis)');
scatter(image_point_origin(1), image_point_origin(2), 'ko', 'filled', 'DisplayName', 'World Origin');

% Draw vanishing lines from world origin to points at infinity
```

```
plot([image_point_origin(1), image_point_infinity_x(1)], ...
     [image_point_origin(2), image_point_infinity_x(2)], ...
     'b--', 'LineWidth', 1.5, 'DisplayName', 'Vanishing Line (X-axis)');

plot([image_point_origin(1), image_point_infinity_y(1)], ...
     [image_point_origin(2), image_point_infinity_y(2)], ...
     'g--', 'LineWidth', 1.5, 'DisplayName', 'Vanishing Line (Y-axis)');

% Draw a line connecting points at infinity in X and Y directions
plot([image_point_infinity_x(1), image_point_infinity_y(1)], ...
     [image_point_infinity_x(2), image_point_infinity_y(2)], ...
     'm--', 'LineWidth', 1.5, 'DisplayName', 'Line Between Infinity Points');

% Add title

% Create legend
legend show;  % Show legend to display all categories

hold off;

```