Students:
ELiyas Kidanemariam Abraha [eliyaskidanemariam@gmail.com]
Goitom Abrha Leaku [goitomabrha2000@gmail.com]

Camera Calibration

November 1, 2023

1 Introduction

Camera calibration is the process of determining and fine-tuning the parameters of a camera to understand how it captures images of the real world. This involves estimating the intrinsic and extrinsic camera parameters, which are essential for translating a 3D point in the world into a 2D point in the image.In this lab, the main objective is to explore camera projection geometry and calibrate a simulated camera. This process involves defining intrinsic and extrinsic camera parameters, generating 3D points, projecting them onto the image plane, estimating the camera projection matrix, introducing noise to evaluate robustness, and observing the impact of the number of 3D points on accuracy. The calibration method of Hall is employed, and throughout the steps, intrinsic parameters, camera rotation matrices, and average projection errors are analyzed.

2 Implementation

Part 1: Camera Calibration and Projection

1. **Initial Parameters Setup(Step1)**:first we define the intrinsic parameters and extrinsic parameters with the following values

   - Intrinsic parameters: $au = 557.0943$, $av = 712.9824$, $u0 = 326.3819$, $v0 = 298.6679$
   - Location of the world reference frame in camera coordinates (in mm): $Tx = 100$, $Ty = 0$, $Tz = 1500$
   - World rotation (Euler XYX1 angles): $Phix = 0.8\pi/2$, $Phiy = -1.8\pi/2$, $Phix1 = \pi/5$

2. **Camera Projection Matrix(Step2)**:

   - Then we create the camera intrinsic matrix K and the extrinsic camera transformation cTw using the provided values.To find cTw, we implement rotation matrices from the Euler XYX1 angles
   - The camera projection matrix P is computed by multiplying the intrinsic matrix K with the extrinsic transformation matrix cTw. It is a 3×4 matrix that represents the mapping from 3D world points to 2D world points.

Listing 1: Computing Projection Matrix

```
K = [au , 0, u0 ; 0, av, v0 ; 0, 0, 1]; %intrinsic matrix
    K
Rot_x =[1,0,0;0,cos(Phix),-sin(Phix);0,sin(Phix),cos(Phix
   )];
Rot_y = [cos(Phiy),0,sin(Phiy); 0,1,0; -sin(Phiy), 0, cos
   (-Phiy)];
Rot_x1 = [1,0,0;0,cos(Phix1),-sin(Phix1);0,sin(Phix1),cos
   (Phix1)];
Rot_mat = Rot_x*Rot_y*Rot_x1; % combining the rotation
   matrixes by mult
Rot = [Rot_mat;0,0,0]; % changing to homogeneous
   coordinate

% Tx = 100; Ty = 0; Tz = 1500;
Tran = [100;0;1500;1]; % Translation matrix
cTw = [ Rot , Tran]; % World to Camera Transformation
   Matrices
P= K*[1,0,0,0;0,1,0,0;0,0,1,0]*cTw ;  % Calaculating
   Projection Matrix
```

3. **Define 3D Points(Step 3)**:

   - Then we generate random six 3D points within the range $[-480 : 480; -480 : 480; -480 : 480]$in the 3D space.Those points represent points in the wrld frame.

4. **Projection of 3D Points(Step 4)**:

   - Project the 3D points onto the image plane using the camera projection matrix P preserve subpixel precision is done by calling a function that recieves number of points , 3d-points and the projection matrix.In this step we implement for six 3d points

Listing 2: projection of 3D points into image plane

```
function p2d = calculate_2d_points(num_points,p3d,P)
    % claculate the 2d points in the image plan from the
        camera projection
    % matrices and the 3d points
    p2d_homeg = [];
    for i=1: num_points
        point = P*[p3d(i,:),1]';
        p2d_homeg = [p2d_homeg ; point'];

    end
```

```
        %Change to Cartishian      %
        p2d = p2d_homeg(:, 1:2) ./ p2d_homeg(:, 3);
    end
```

5. **Visualization(step 5)**:

   - Then We Plot the 2D points and to display the image plane and plot 2D points we figure and plot Matlab functions.This visualization helps us to verify the 2D points are correctly projected onto the image plane and gives you an idea of their spatial distribution within the image as shown in figure 1.We make so many tests the points are Confirm the points are well spread in the image plane due to the randomly generated 3d points are spread between [-480,480].

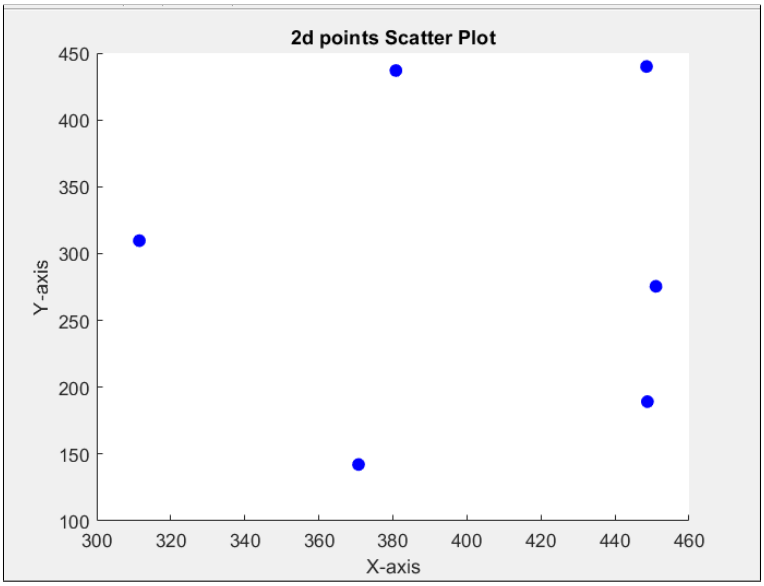


Figure 1: Scattered 2d points

Q1: Will the distribution of points in the image affect the accuracy in the computation? Why? If the 2d points are concentrated in a small area or aligned along a single axis, it can lead to poor calibration accuracy.If the points are too closely packed or aligned along certain axes, it may be challenging for the algorithm to determine the camera's intrinsic and extrinsic parameters accurately.The points should be well spread to provide sufficient information for accurate calibration.

Part 2: Calibration Accuracy and Noise

1. **Camera Projection Matrix Estimation((Step 6))**:

   - Use the 3D points and their projections to estimate the 3x4 camera projection matrix P using the method of Hall.

- The function **get_estimated_projecion_matrix** is designed to estimate the camera projection matrix P using the method of Hall. It initializes matrices Q and B2 to store intermediate values and iterates through a set of 3D-to-2D point correspondences. For each correspondence, it constructs rows in the matrix Q based on the 3D world coordinates and 2D image coordinates, assembling the complete matrices Q and B2. By solving for unknown variables A through a mathematical equation, it derives a column vector representing parameters related to the camera projection matrix. Adding a value of 1 to this vector, it reshapes the parameters and multiply by the end value of the original projection matrix, producing the estimated projection matrix P_Estimated.This function effectively calibrates the camera by estimating the transformation from 3D world coordinates to 2D image coordinates, a crucial step in computer vision and image analysis.

Listing 3: camera projection matrix estimation

```
function P_Estimated = get_estimated_projecion_matrix(p2d
    ,p3d,num_points,P)
    % initializing
    Q=zeros(12,11);
    B2= zeros(12,1);
    % callculate the projection matrices using method of
       hall
    for i=1:num_points;
        Q(2*i-1,:) = [p3d(i,:),1,0,0,0,0,-p2d(i,1)*p3d(i,1),
            -p2d(i,1)*p3d(i,2),-p2d(i,1)*p3d(i,3)];
        Q(2*i,:) = [0,0,0,0,p3d(i,:),1,-p2d(i,2)*p3d(i,1), -
           p2d(i,2)*p3d(i,2),-p2d(i,2)*p3d(i,3) ];
        B(2*i-1,1) = p2d(i,1);
        B(2*i,1) = p2d(i,2);
    end
    A=inv(Q'*Q)*Q'*B;
    % Getting the unkown variables in colomun vector
    A=[A;1];
    % getting the estimated P by reshaping the calculated
       unkown variables and multiplyin by the P(3,4) to
    P_Estimated = reshape(A,4,3)'*P(3,4);
send
```

2. **Comparison with Defined Parameters(Step 7)**:

- After the We get the estimated then we compare it with original projection matrix one defined earlier by calculating their difference.The result is almost zero as shown below

so hey are they same.

$$P_{\text{difference}} = 10^{-8} \begin{bmatrix} -0.0002 & -0.0019 & 0.0031 & 0.3376 \\ -0.0021 & -0.0024 & 0.0008 & -0.0175 \\ -0.0000 & 0.0000 & 0.0000 & 0 \end{bmatrix}$$

- Then we extract the intrinsic parameter matrix K and the camera rotation matrix cRw from estimated P.

3. **Add Gaussian Noise(Step 8)**:

   - We add Gaussian noise to all 2D points, creating discrepancies between $[-1, +1]$ pixels afexting 95% of the points.
   - then we calculate the camera projection matrix estimation with noisy 2D points.We compare the noisy projection matrix with one nosie free projection and by computing thier difference they have slightly difference depending on noise data.

   Q2: How did the intrinsic parameters change? When Gaussian noise is added to the 2D points, the intrinsic parameters, such as the focal lengths (u and v) and the principal point coordinates (u0 and v0), undergo slight variations.The extent of these changes depends on the magnitude of the noise introduced.We introduce a noise between and affecting 95% of the points the change is not much significant. **Q3: Why is the axis skew parameter different from zero?** When Gaussian noise is added to the 2D points and the calibration is performed on these noisy data, it can lead non-zero skew (γ) values. The noise introduces uncertainties in the position of image points, potentially causing the image axes to deviate from orthogonality.

4. **Average Projection Error(Step 9)**:

   - Then we Compute the average projection error as the mean Euclidean distance between 2D points from different.The function in listing 5 quantifies the average error between two sets of 2D points by calculating the squared Euclidean distances and taking their mean. It provides a measure of how different the two sets of points are on average.

Listing 4: calculating the the average projection error

```
function error = calculate_avarege_projection_error(
   p2d_new,p2d)
  % Calculate the element-wise squared differences
  squared_differences = (p2d_new - p2d).^2;
  distance = sum(squared_differences,2);
  error= mean(distance);
end
```

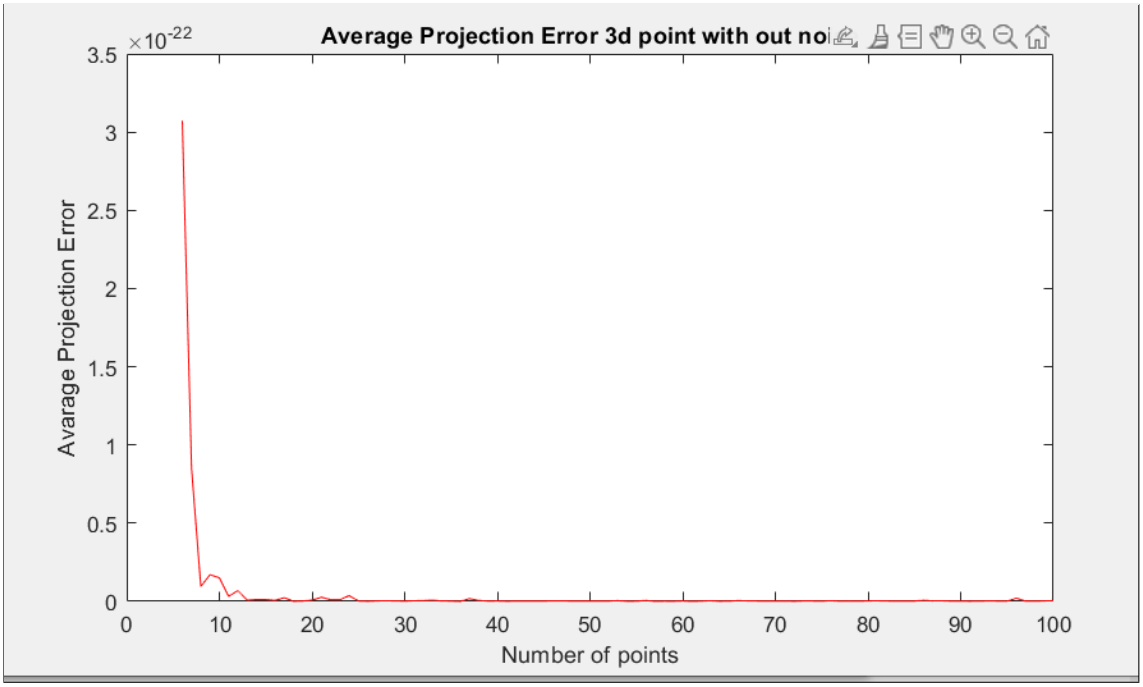
5. **Increasing the Number of Points(Step 10)**:

- We Increase the number of 3D points upto 200 then repeat the previous steps with the increased number of points. we define a loop to estimate and record average projection errors and noise errors for different numbers of points. It then generates two plots, one showing the relationship between the number of points and the average projection error as shown in the figure 2a which is the average projection error is almost zero($3 * 10^{-22}$) initially and the other showing the relationship between the number of points and the noise error as shown in the figure 2b the projection error decreases as the number points increase.These plots provide insights into how the errors change as the number of points varies.We Observe the improved accuracy, as the number of 3d points increase the number of projection error decrease.

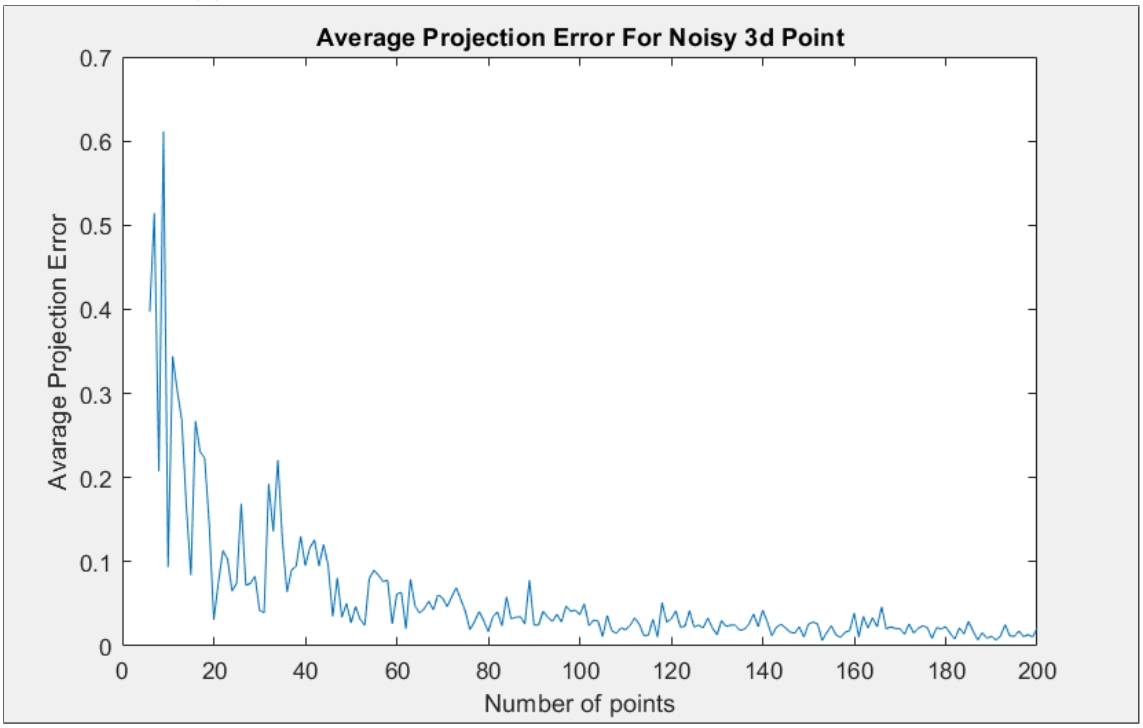
Listing 5: calculating the the average projection error

```
num_points = 6:200; % number of points to estimate
average_projection_error= [];
i=1;
while i<=length(num_points)
    num_point = num_points(i);
    [error1,noise_error]=projection_handler(num_point,P,K);
    average_projection_error=[average_projection_error;
       num_point, error1,noise_error];
    i=i+1;
end

figure;
plot(average_projection_error(:,1),
   average_projection_error(:,2))
xlabel('Number of points')
ylabel('Avarage Projection Error')
plot(average_projection_error(:,1),
   average_projection_error(:,3))
xlabel('Number of points')
ylabel('Avarage Projection Error');
end
```



(a) average projection error with out noise



(b) average projection error with noise

Figure 2: avarage projection error

3 Conclusion

In conclusion, the distribution of 2D points in the image significantly affects the accuracy of the camera calibration computation.If points are concentrated in specific areas, the calibration's accuracy may be compromised.Additionally, the introduction of Gaussian noise to the 2D points led to slight variations in the intrinsic parameters, while the non-zero skew value (γ) resulted from uncertainties introduced by the noise in the image point positions. The computation of the average projection error provided an essential metric for quantifying the disparity between sets of 2D points, offering insights into the accuracy of the estimation process. Finally, the analysis of increasing the number of points further underscored the robustness of the calibration process with a larger dataset, showcasing its improved accuracy and reliability.