ENPM673: Project 4

Aditi Arun Bhoir

UID: 119197257

Part 1: Calibration

Computer Vision Pipeline:

1. Import the necessary libraries.
2. Read the images using cv2.imread() function.
3. Convert the images to grayscale using cv2.cvtColor() function.
4. Extract key points and descriptors using SIFT algorithm.
5. Draw the key points on original images using cv2.drawKeypoints() function.
6. Match the features between two images using BF matcher by creating bf_matcher object.
7. Sort the matches based on their distance using sorted () function.
8. Select the top 120 matches with lowest distances and draw the matches on images.
9. Extract the coordinates of best matches from the matches objects and stored in a NumPy array best_matches in the format [x1, y1, x2, y2], where (x1, y1) were the coordinates of key points in the first image, and (x2, y2) were the coordinates of key points in the second image.
10. Define a normalizing function that will take any feature from the matched features and scale and translate to make all the points in the same range. The function the normalized points normalised_value and the translation matrix "Translation" as outputs.
11. Define a function to compute the fundamental matrix using svd. Extract the feature points by slicing and normalize them. Initialize matrix A with zeros. Loop through each pair of normalized features and calculate the corresponding row of "A" by filling it with values computed from these feature points.
12. Perform Singular Value Decomposition (SVD) on A using np.linalg.svd () function and get initial Fundamental matrix by taking the last column of Vt.
13. Perform SVD on F to enforce rank2 constraint by setting smallest singular value to 0.
14. Compute the fundamental matrix by multiplying translation2.T, F, and translation1 together.
15. Define the function to perform ransac method to find best fundamental matrix. initializes variables best_num_inliners to 0, best_inliers to an empty list, and best_F_matrix to None to keep track of the best fundamental matrix and its corresponding inliers found during the iterations.
16. Iterate through num_iterations. In each iteration, randomly select 8 feature points from matched_features to form random_points. call the compute_fundamental_matrix

function to compute the fundamental matrix fundamental_matrix using the random_points.

17. Loop through all the feature points in matched_features and compute the error between each feature point and its epipolar line using the computed fundamental_matrix. If the error is below the threshold_distance, the feature point is considered an inlier and its index is appended to the inliers list.
18. Compare the number of inliers found in the current iteration with the best_num_inliners to determine if the current fundamental matrix is better than the previously best one. If so, update best_num_inliners, best_inliers, and best_F_matrix accordingly.
19. Store the inliers belonging to the iteration giving the best fundamental matrix. The num_iterations are selected as 800 and threshold_distance as 0.007.
20. Define the intrinsic matrices for left and right images. The essential matrix is calculated as the multiplication of K.T @ Best_F_matrix @ K.
21. Decompose the E matrix using svd. Set the singular value to 0 and recalculate it using U @ np. diag([1, 1, 0]) @ Vt.T.
22. We will get 2 translations and 2 rotations from this. But one of the translations has negative z coordinate. Therefore, we consider only one translation and two rotations.
23. Define a triangulation function which takes the two image points and corresponding projection matrices and returns the 3D point cloud.
24. Form the projection matrix using the rotation, translation, and intrinsic matrix.
25. For left projection matrix = [cam0 np. zeros]
26. For right projection matrix = [(cam1 @ R) (cam1 @ C)]
27. For the two point clouds find which has maximum points having positive depth value i.e. positive z coordinate.
28. The R belonging to max points having +ve depth will be selected as the final rotation matrix.

Math for the fundamental matrix calculation and Essential Matrix Calculation:

Fundamental Matrix:

Fundamental matrix is a 3*3 matrix we can set up a homogeneous linear system with 9 unknowns.

$$\begin{bmatrix} x'_i & y'_i & 1 \end{bmatrix} \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = 0$$

$$x_i x'_i f_{11} + x_i y'_i f_{21} + x_i f_{31} + y_i x'_i f_{12} + y_i y'_i f_{22} + y_i f_{32} + x'_i f_{13} + y'_i f_{23} + f_{33} = 0$$

$$\begin{bmatrix} x_1x'_1 & x_1y'_1 & x_1 & y_1x'_1 & y_1y'_1 & y_1 & x'_1 & y'_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_mx'_m & x_my'_m & x_m & y_mx'_m & y_my'_m & y_m & x'_m & y'_m & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{21} \\ f_{31} \\ f_{12} \\ f_{22} \\ f_{32} \\ f_{13} \\ f_{23} \\ f_{33} \end{bmatrix} = 0$$

Where (x1, y1) and (x1$'$, y1$'$) are the feature points from left image and right image respectively.

Now we have to solve,

Ax = 0 using SVD

When applying the SVD to A, we obtain the decomposition USVT, where S is a diagonal matrix containing the singular values of A. The last singular value (σ9) is zero if A has rank 8 since we have 8 equations for 9 unknowns. The last column of V gives the true solution if all singular values except the last one is non-zero.

However, when we estimate the fundamental matrix F from correspondences, it should have rank 2 due to the constraint that it must satisfy. But due to noise in the correspondences, the estimated F matrix can have a higher rank, including the possibility that the last singular value is non-zero. To enforce the rank 2 constraint, we set the last singular value to zero using the truncated SVD method.

Epipolar constraint:

It is calculated as follows:

Error = X2.T @ F @ X1

Where x1 and x2 are corresponding feature points in the two views, T denotes transpose, and F is the fundamental matrix.

Essential Matrix:

Essential Matrix is a 3 by 3 matrix and is calculated as follows:

E = K.T @ F @ K

Where K is the camera intrinsic, and F is the fundamental matrix.

When computing the F matrix, the singular values of E may not necessarily be (1,1,0) due to the presence of noise in K. To address this, it is possible to reconstruct E with singular values of (1,1,0).

$$\mathbf{E} = U \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} V^T$$

$$\mathbf{E} = UDV^T \text{ and } W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

1. $C_1 = U(:,3)$ and $R_1 = UWV^T$
2. $C_2 = -U(:,3)$ and $R_2 = UWV^T$
3. $C_3 = U(:,3)$ and $R_3 = UW^TV^T$
4. $C_4 = -U(:,3)$ and $R_4 = UW^TV^T$

It is important to note that the det(R)=1. If det(R)=−1, the camera pose must be corrected *i.e.* C=−C and R=−R.

Results for part 1:

Dataset 1:

Detected features in both the images.

Matched features between both the images

```
Fundamental matrix =
  [[-6.31861831e-11  6.30404416e-08 -7.72869016e-05]
 [-6.18890192e-08 -9.10121143e-10  1.55447451e-03]
 [ 7.74600458e-05 -1.55241750e-03 -1.84790371e-03]]
```

```
Essential matrix =
  [[ 0.02176833  0.05426557 -0.05149629]
 [ 0.043745    0.34580774  0.93684825]
 [-0.31674052 -0.88307299  0.33853476]]
```

Fundamental Matrix and Essential Matrix

```
Final rotaion matrix =
  [[ 0.95660653 -0.2612189   0.12910705]
 [-0.29025101 -0.89326277  0.34327246]
 [ 0.02565727 -0.36585013 -0.93032004]]
```

```
Final translation matrix =
  [0.99696014 0.02861348 0.07246895]
```

Rotation Matrix and Translation Matrix

<u>Dataset 2:</u>

Detected features in both the images.

Matched features between both the images

```
Fundamental matrix =
  [[ 3.97098567e-10  3.72833677e-07 -2.78933035e-04]
 [-3.75224603e-07  3.23321722e-08  2.79711295e-03]
 [ 2.77402268e-04 -2.78399716e-03 -2.50517054e-02]]
```

```
Essential matrix =
  [[-0.13092253  0.20996455  0.07940954]
 [-0.01677003 -0.47445664  0.87965589]
 [ 0.49131093 -0.73251257 -0.3941096 ]]
```

Fundamental Matrix and Essential Matrix

```
Final rotaion matrix =
  [[ 0.84970613  0.52721108  0.00692656]
 [-0.48366757  0.77416229  0.40833617]
 [ 0.20991707 -0.3503159   0.91280534]]
```

```
Final translation matrix =
  [0.96564398 0.02855134 0.25829541]
```

Rotation Matrix and Translation Matrix

Dataset 3:

Detected features in both the images

Matched features between both the images

```
Fundamental matrix =
  [[-4.56546131e-09  9.32201445e-07 -9.71431145e-04]
 [-9.32040273e-07  9.75372680e-09 -9.63322200e-04]
 [ 9.71685123e-04  9.90067869e-04 -3.30217286e-02]]



Essential matrix =
  [[-0.3831907  -0.48971304  0.4660273 ]
 [ 0.82393396 -0.54126735  0.1629409 ]
 [-0.35343321 -0.36918217  0.36954949]]
```

Fundamental Matrix and Essential Matrix

```
Final rotaion matrix =
  [[ 0.51372087 -0.76728683 -0.3838773 ]
 [ 0.52874828  0.63550283 -0.56263791]
 [ 0.67565977  0.08606437  0.73217266]]


Final translation matrix =
  [-0.62941606  0.04015906  0.77603007]
```

Rotation Matrix and Translation Matrix

Part 2: Rectification

Computer Vision Pipeline:

1. Define a draw_epilines function that draws epipolar lines on two input images using the provided lines and corresponding feature points and returns the modified images with epipolar lines and marked feature points.
2. Get the shape (height, width) of dataset_1_image_1 and dataset_1_image_2 and store them in variables h1, w1, h2, w2 respectively.
3. Use the cv. stereoRectifyUncalibrated function to compute homography matrices (H1, H2) using the best feature points (best_points_calc1, best_points_calc2) and the computed fundamental matrix (best_F_matrix).
4. Warp the grayscale images (gray1, gray2) using the homography matrices H1 and H2 to obtain the rectified grayscale images img1_rectified and img2_rectified using the cv.warpPerspective function.
5. Compute the epipolar lines for the feature points in img2_rectified using the cv. computeCorrespondEpilines function with the corresponding view as 2 and the computed fundamental matrix as best_F_matrix. Reshape the output lines to have 3 elements per line and store them in the variable lines.
6. Draw the epipolar lines on img1_rectified using the "draw_epilines" function, passing the rectified images, lines1, and the best feature points (best_points_calc1, best_points_calc2) as arguments. Store the resulting images in img5 and img6.
7. Do the same for image 1. Concatenate both the images and display.

Results for part 2:

Dataset 1:

```
Homography matrix for left image  =
 [[-2.18954394e-03  3.35340521e-05  8.11339634e-02]
 [ 6.52596680e-05 -2.28162646e-03 -6.00908799e-02]
 [ 9.40742616e-08 -6.87060809e-09 -2.36226426e-03]]


Homography matrix for right image  =
 [[-2.18954394e-03  3.35340521e-05  8.11339634e-02]
 [ 6.52596680e-05 -2.28162646e-03 -6.00908799e-02]
 [ 9.40742616e-08 -6.87060809e-09 -2.36226426e-03]]
```

Homography matrices for left and right images

Epilines on both images after rectification

Epilines on both images before rectification

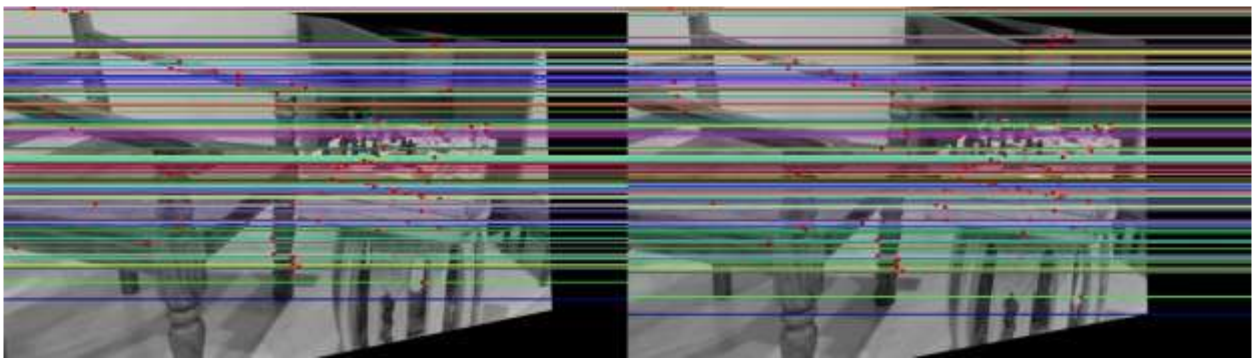
Two lines on the warped images showing the points at the same height.

Dataset 2:

```
Homography matrix for left image  =
 [[ 2.29435905e-03 -6.13666364e-05  1.56845939e-01]
 [-3.20520307e-04  2.79283466e-03  3.42684191e-01]
 [-4.32896148e-07  2.34712389e-08  3.23726199e-03]]


Homography matrix for right image  =
 [[ 2.29435905e-03 -6.13666364e-05  1.56845939e-01]
 [-3.20520307e-04  2.79283466e-03  3.42684191e-01]
 [-4.32896148e-07  2.34712389e-08  3.23726199e-03]]
```

Homography matrices for left and right images



Epilines on both images after rectification

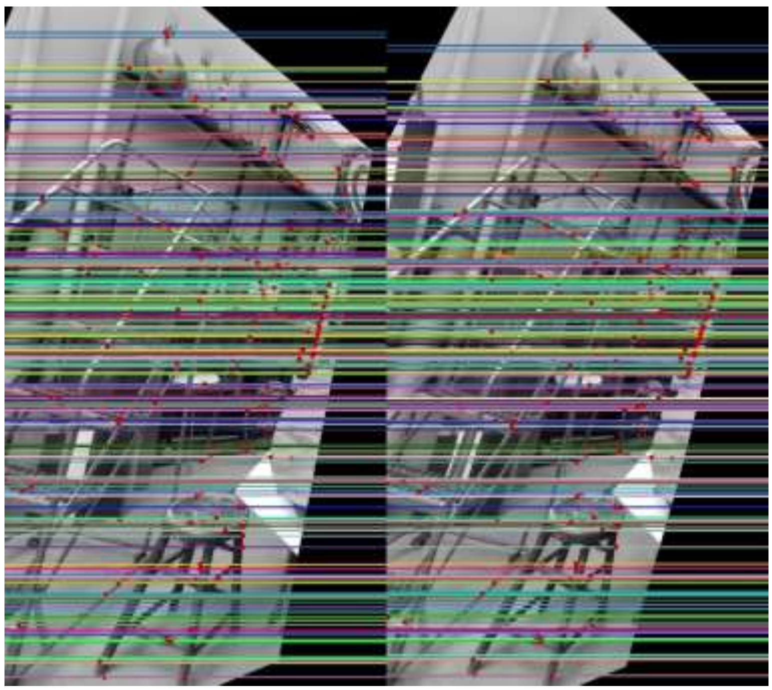
Epilines on both images before rectification

Dataset3:

```
Homography matrix for left image  =
 [[ 1.86709767e-03 -1.31702020e-04 -2.15484583e-01]
 [ 8.98816384e-04  1.34138428e-03 -4.71886475e-01]
 [ 8.64257028e-07 -5.60758630e-08  9.42039090e-04]]


Homography matrix for right image  =
 [[ 1.86709767e-03 -1.31702020e-04 -2.15484583e-01]
 [ 8.98816384e-04  1.34138428e-03 -4.71886475e-01]
 [ 8.64257028e-07 -5.60758630e-08  9.42039090e-04]]
```

Homography matrices for left and right images



Epilines on both images after rectification

Epilines on both images before rectification

Part 3: Correspondence

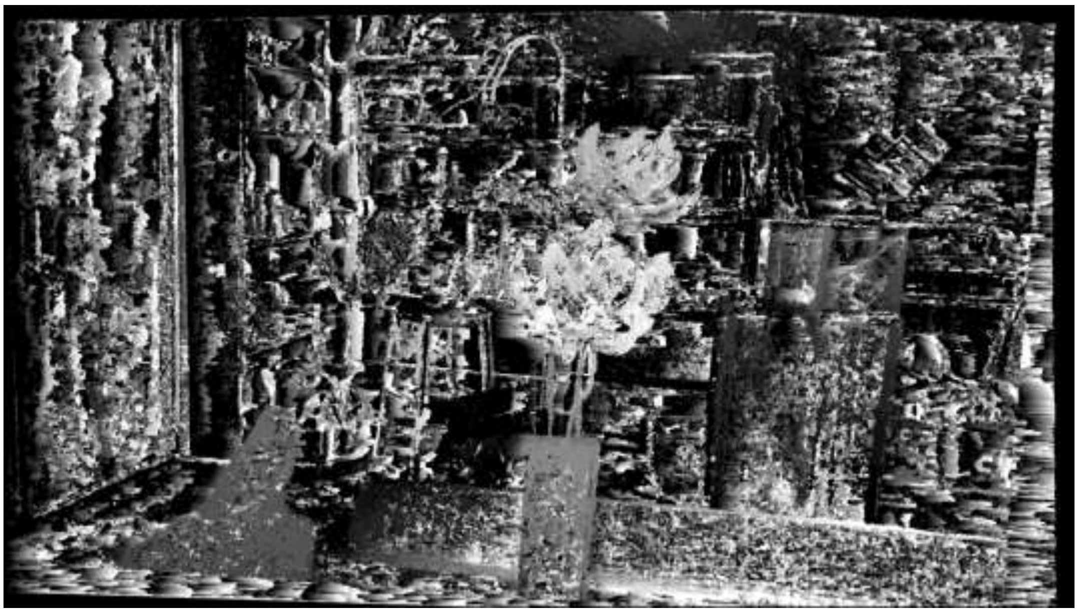
Computer Vision Pipeline:

1. Set the window size for the block matching algorithm to a suitable value, which means that it will compare blocks of 5x5 pixels between the left and right images.
2. Set the disparity range to a suitable value, which means that it will search for matching blocks in a range of up to 64 pixels to the left of each pixel in the left image.
3. Initialize an empty disparity map, which will be filled with the computed disparities between the left and right images.
4. Loop over all the pixels of the left image and skip the borders.
5. For each pixel in the left image extract 5 cross 5 block of pixels, this block will be centered at the corresponding pixel of the right image. Search for the best matching block in the range of disparities specified.
6. Calculate the sum squared differences between the left and right image block for each possible disparity value and the block will be selected giving the lowest SSD. The center of that block will be the corresponding point in the right image.
7. Fill the disparity map with the computed value for each pixel in the left image.

8. Normalize the disparity map to the range (0, 255).
9. Display the grayscale and heatmap conversion image.

Results for part 3:

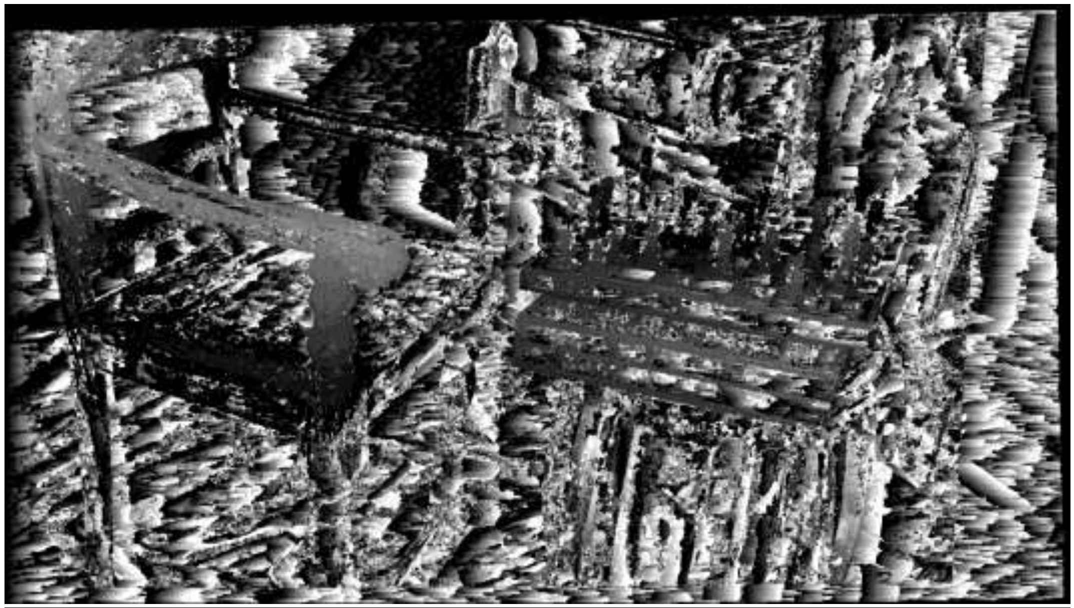
Dataset 1:



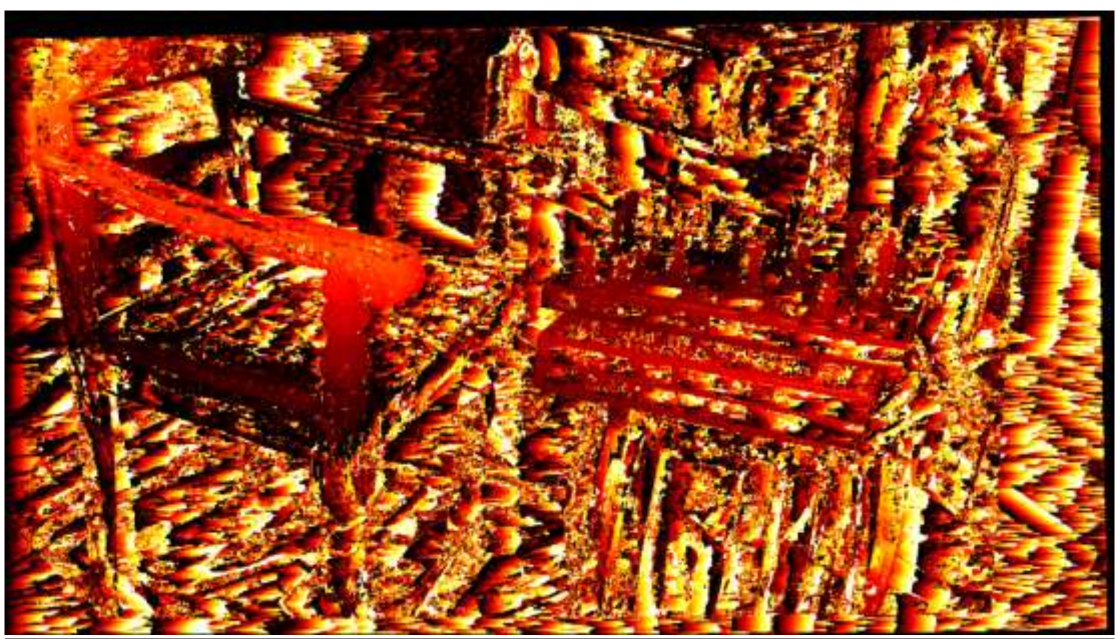
Disparity as grayscale image

Disparity using heatmap conversion.

Dataset 2:



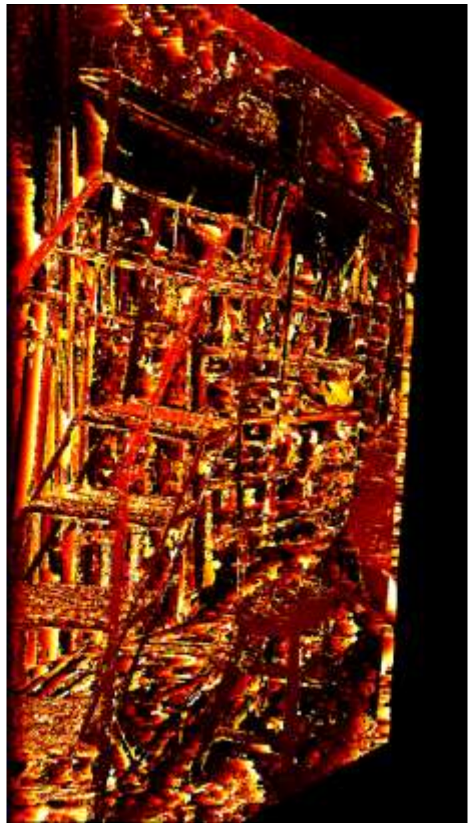
Disparity as grayscale image



Disparity using heatmap conversion.

Dataset3:

Disparity as grayscale image



Disparity using heatmap conversion.

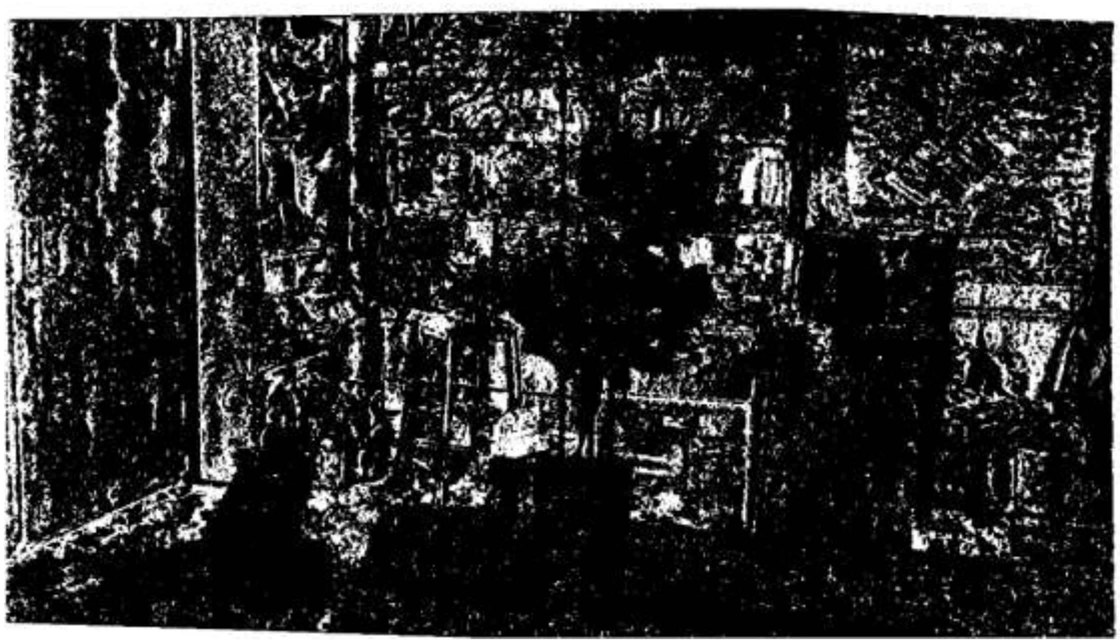
Part 4: Compute depth image

Computer Vision Pipeline:

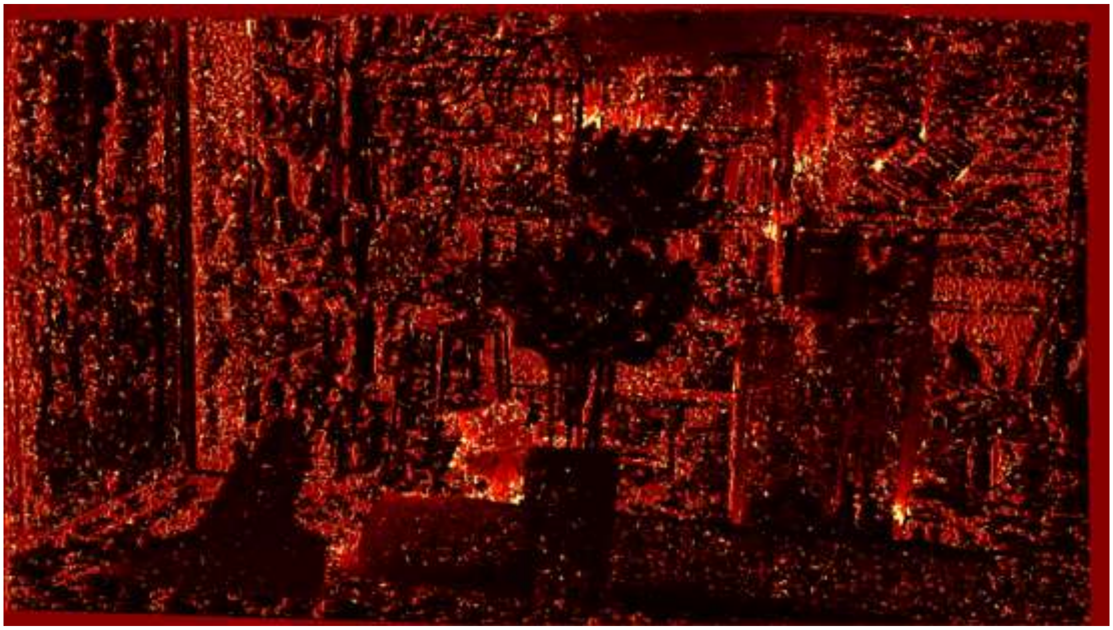
1. Define the baseline distance and convert to meters, define focal length in pixels.
2. Set small value as epsilon to avoid the division by zero errors.
3. Calculate the depth map using depth = (baseline * focal_length)/disparity. This gives the distance of each point in the scene from camera in meters.
4. Normalize the depth map to (0, 255) using min and max values of depth map.
5. Display the grayscale and heatmap conversion image.

Results for part 4:

Dataset 1:

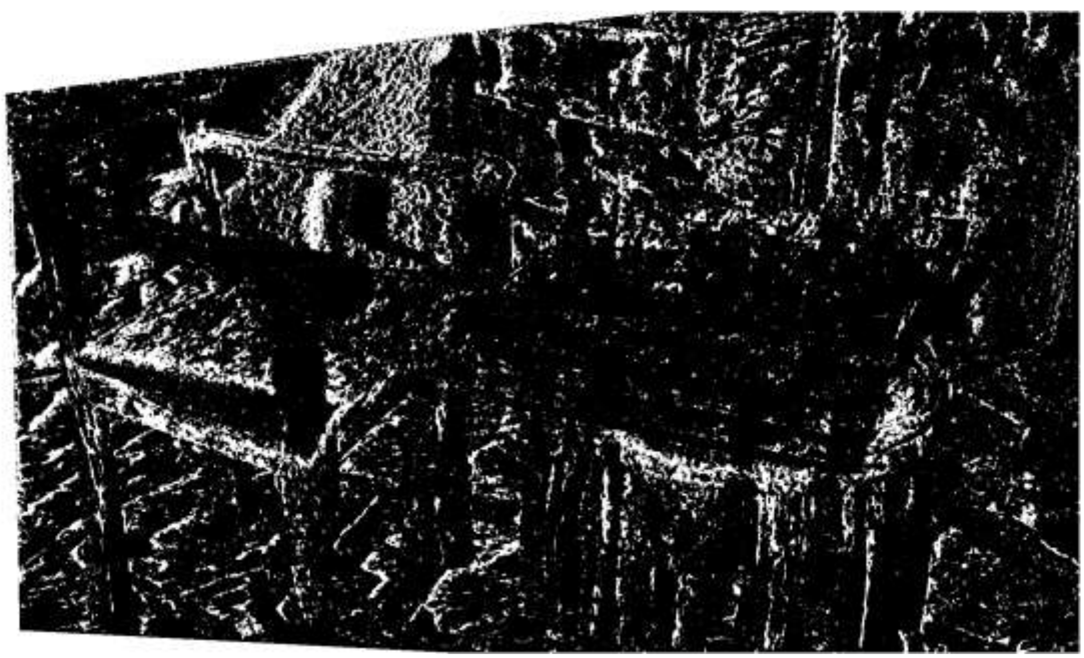


Depth as grayscale image

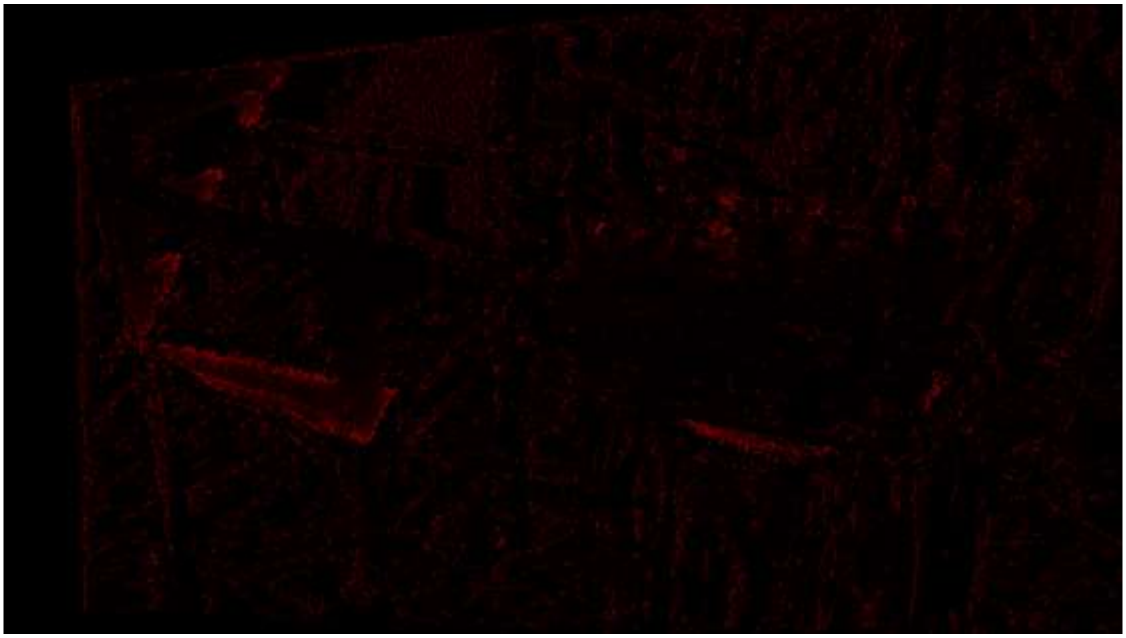


Depth using heatmap conversion.

Dataset 2:



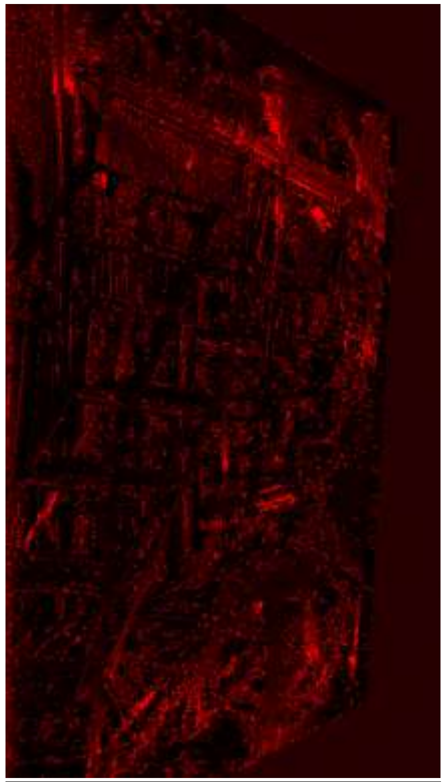
Depth as grayscale image



Depth using heatmap conversion.

Dataset3:

Depth as grayscale image



Depth using heatmap conversion.