Computer Vision Course

Comp 558

Assignment Number 3

By Asa Borzabadi Farahani

CONTENTS

Question 1: Fundamental Matrix Estimation 3
Question 2: Fundamental Matrix Refinement – RANSAC.....10
Question 3: Rectification 14
Question 4: Reconstruction 20
Question 5: Dense Reconstruction – Disparity 29

QUESTION 1: FUNDAMENTAL MATRIX ESTIMATION

In the eight-point algorithm, we have the pixel coordinate of 8 pairs of points in image one and image two, and our goal is to find the Fundamental matrix. This goal can be achieved by solving a least square minimization problem as below:

$$A\,x = 0 \qquad \begin{bmatrix} x_1^1x_2^1 & y_1^1x_2^1 & x_2^1 & x_1^1y_2^1 & y_1^1y_2^1 & y_2^1 & x_1^1 & y_1^1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\ x_1^ix_2^i & y_1^ix_2^i & x_2^i & x_1^iy_2^i & y_1^iy_2^i & y_2^i & x_1^i & y_1^i & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\ x_1^Nx_2^N & y_1^Nx_2^N & x_2^N & x_1^Ny_2^N & y_1^Ny_2^N & y_2^N & x_1^N & y_1^N & 1 \end{bmatrix} \begin{bmatrix} F_{11} \\ F_{12} \\ F_{13} \\ F_{21} \\ F_{22} \\ F_{23} \\ F_{31} \\ F_{32} \\ F_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

8×9

One needs to follow the algorithm mentioned below:

1. **Select matching pairs and normalize points**
2. **Construct the M x 9 matrix A**
3. **Find the SVD of A**
4. **Entries of F are the elements of column of V corresponding to the least singular value**
5. **Enforce rank 2 constraint on F**
6. **Un-normalize F**

- **Select matching pairs and normalize points**

Here, in the very first step, these **eight pairs of features** in image number one and image number two were chosen as below:

data point pairs in image 1 and image 2

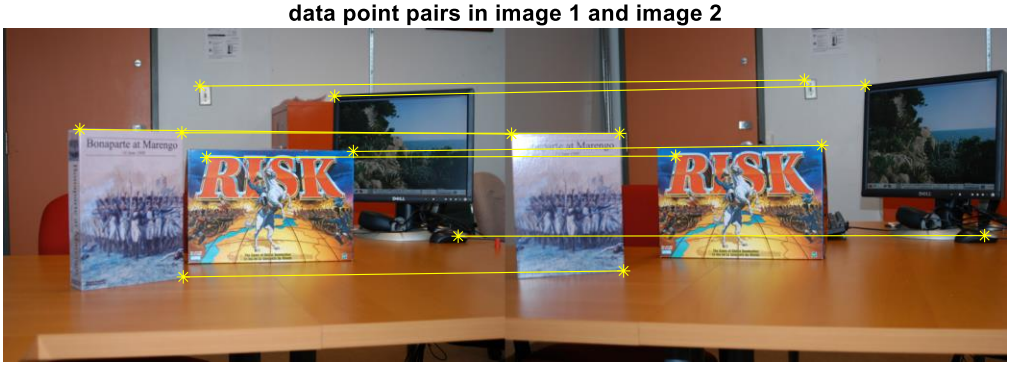


image 1

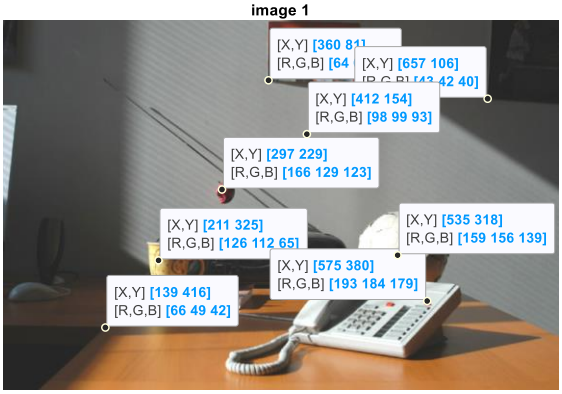
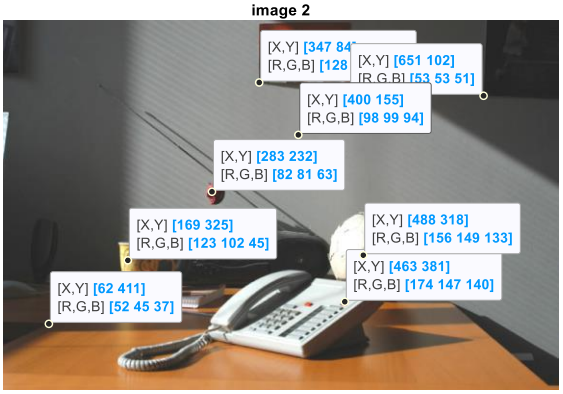
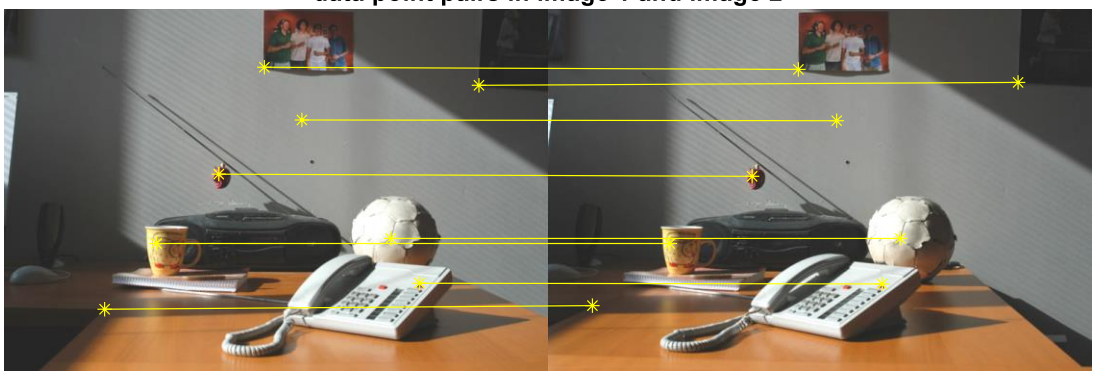


image 2



data point pairs in image 1 and image 2



In the next step, I tried to **normalize** the data that I have chosen; this normalization step makes the mean value of x-pixel position of features in image 1, x-pixel position of features in image 2, y pixel position of features in image 1, and y pixel position of features in image 2 equal to 0 and normalize the standard deviation of these values. For the first given pair of images, the 8 pairs of features are as below:

```
% data of the 8 pairs
xPos_img1 = [115,295,268,305,270,497,525,682];
yPos_img1 = [152,87,157,194,373,102,185,312];
```

```
xPos_img2 = [10,449,172,256,178,540,475,719];
yPos_img2 = [159,78,158,192,364,86,176,311];
```

The normalization process is done using this method:

```
% data normalization
mean_x1 = mean(xPos_img1);
mean_y1 = mean(yPos_img1);
mean_x2 = mean(xPos_img2);
mean_y2 = mean(yPos_img2);

std_x1 = std(features1(:,1));
std_y1 = std(features1(:,2));
std_x2 = std(features2(:,1));
std_y2 = std(features2(:,2));
```

To apply normalization, one needs to multiply x and y value by these matrices to get normalized values:

```
M1 = [1/(std_x1) 0 0; 0 1/(std_y1) 0; 0 0 1] * [1 0 -mean_x1; 0 1 -mean_y1; 0 0 1];
```

$$M1 = \begin{bmatrix} \frac{1}{(\mathrm{std}_{x1})} & 0 & 0 \\ 0 & \frac{1}{(\mathrm{std}_{y1})} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -mean(x1) \\ 0 & 1 & -mean(y1) \\ 0 & 0 & 1 \end{bmatrix}$$

```
M2 = [1/(std_x2) 0 0; 0 1/(std_y2) 0; 0 0 1] * [1 0 -mean_x2; 0 1 -mean_y2; 0 0 1];
```

$$M2 = \begin{bmatrix} \frac{1}{(\mathrm{std}_{x2})} & 0 & 0 \\ 0 & \frac{1}{(\mathrm{std}_{y2})} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -mean(x2) \\ 0 & 1 & -mean(y2) \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} x1\_normalized \\ y1\_normalized \\ 1 \end{bmatrix} = M1 * \begin{bmatrix} x1 \\ y1 \\ 1 \end{bmatrix} \; and \; \begin{bmatrix} x2\_normalized \\ y2\_normalized \\ 1 \end{bmatrix} = M2 * \begin{bmatrix} x2 \\ y2 \\ 1 \end{bmatrix}$$

This is how one can **apply this normalization** in MATLAB:

```
% apply normalization
for i = 1:1:8
    a = M1 * [xPos_img1(i); yPos_img1(i); 1];
    x1_norm(i) = a(1); y1_norm(i) = a(2);
    b = M2 * [xPos_img2(i); yPos_img2(i); 1];
    x2_norm(i) = b(1); y2_norm(i) = b(2);
end
```

Note that the normalization step is a step that can prevent numerical issues in extreme cases.

- **Construct the M x 9 matrix A**

In the next step, **A matrix** was built using these normalized values:

$$\begin{bmatrix} x_1^1x_2^1 & y_1^1x_2^1 & x_2^1 & x_1^1y_2^1 & y_1^1y_2^1 & y_2^1 & x_1^1 & y_1^1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\ x_1^ix_2^i & y_1^ix_2^i & x_2^i & x_1^iy_2^i & y_1^iy_2^i & y_2^i & x_1^i & y_1^i & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\ x_1^Nx_2^N & y_1^Nx_2^N & x_2^N & x_1^Ny_2^N & y_1^Ny_2^N & y_2^N & x_1^N & y_1^N & 1 \end{bmatrix}$$

- **Find the SVD of A**

Now, the **least square problem needs to be solved**.

$$Ax = 0$$
$$minimize\ Ax - subject\ to\ \left||x|\right| = 1.$$

The best solution, in this case, would be the **eigenvalue of A^TA corresponding to the smallest eigenvector**. One can find the eigenvalues and eigenvectors of A by singular value decomposition of A.

```
% apply SVD (on A)
[U,S,V] = svd(A);
```

- **Entries of F are the elements of column of V corresponding to the least singular value**

Now, the smallest eigenvalue will be selected, and the corresponding eigenvector will be found; this eigenvector gives the values of normalized fundamental matrix entries.

```
% find F - the eigenvector that corresponds to the smallest eigenvalue
F_norm_rank3 = [V(1,9),V(2,9),V(3,9);...
 V(4,9), V(5,9),V(6,9);...
 V(7,9), V(8,9),V(9,9)];
```

- **Enforce rank 2 constraint on F**

But the problem is that this normalized fundamental matrix is **not rank deficient**, so one step in the eight-point algorithm is to compensate for this constraint. To have a fundamental matrix of rank 2, one should apply singular value decomposition on this normalized fundamental matrix, **find the smallest singular value, and then set it equal to zero**.

```
% rank(F_rank3) = 3; make it have rank 2
[UF,SF,VF] = svd(F_norm_rank3);
SF(3,3) = 0 ;
```

After making the smallest singular value equal to zero, a rank-deficient normalized fundamental matrix will be found; this new fundamental matrix has rank two instead of having rank 3.

```
F_norm_rank2 = UF*SF*transpose(VF); % This F has rank 2
```

- **Un-normalize F**

After obtaining the normalized fundamental matrix, one must retrieve the fundamental matrix in the original coordinate (not the normalized version). To do so, I should apply this matrix multiplication:

```
% denormalize F
F = transpose(M2) * F_norm_rank2 * M1;
F_final = F/norm(F)
```

By completing this step, the final version of the fundamental matrix will be obtained.

To ensure that the results are promising, the results are also compared with the fundamental matrix that the build-in MATLAB function can generate. There is a slight difference in the magnitude of entries; ***this is acceptable since the coordinate normalization step in the build-in function is different from how I applied normalization.***

Note that “F_final” is the fundamental matrix I found using the described method, and “F_test” is the fundamental matrix generated by the MATLAB build-in function for the eight-point

algorithm. And, for the first pair of images (monitor, risk game, book,…), I obtained this fundamental matrix:

```
F_final =

    0.0000   -0.0000    0.0029
   -0.0000    0.0000    0.0438
    0.0009   -0.0403   -0.9982


F_test =

   -0.0000    0.0000   -0.0029
    0.0000   -0.0000   -0.0438
   -0.0009    0.0404    0.9982
```

```
F_test =
estimateFundamentalMatrix(features1,
features2,'Method','Norm8Point')
```

And, for the second pair of images (telephone, ball, radio,…), I obtained this fundamental matrix:

```
F_final =

   -0.0000   -0.0000    0.0027
    0.0000   -0.0000   -0.0399
   -0.0052    0.0377    0.9985


F_test =

   -0.0000   -0.0000    0.0027
    0.0000   -0.0000   -0.0398
   -0.0052    0.0376    0.9985
```

```
F_test =
estimateFundamentalMatrix(features1,
features2,'Method','Norm8Point')
```

- **Epipoles and epipolar lines (extra information):**

Now that I have the fundamental matrix I can find the right and left null space of the fundamental matrix. These null spaces of the fundamental matrix are epipoles.

$$Finding\ epipoles:$$

$F\ e_1 = 0$ → right null space of fundamental matrix (epipole 1)

$transpose(e_2)\ F = 0$ → left null space of fundamental matrix (epipole 2)

At the first pair of images, the epipoles are located at:

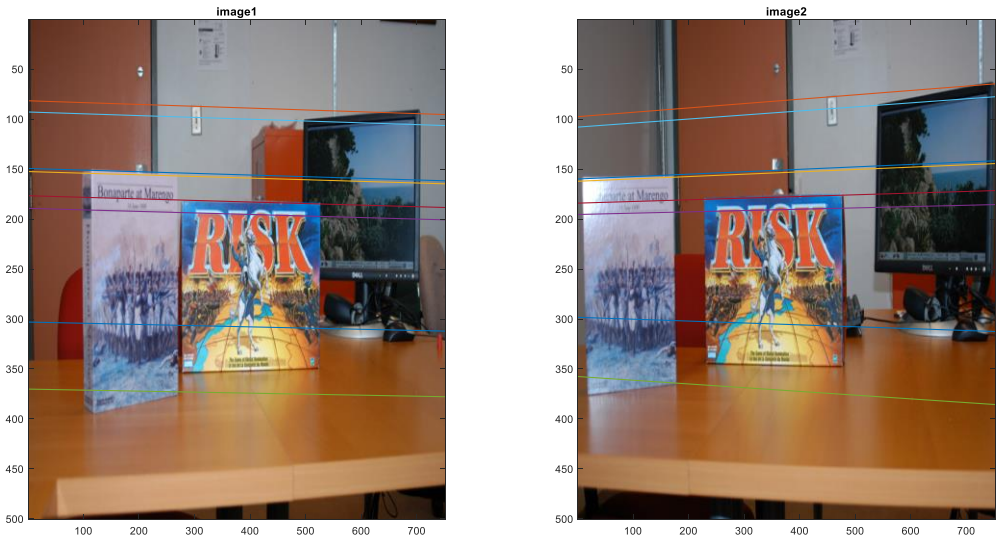
```
e1 =                 e2 =

    1.0e+04 *            1.0e+03 *

     3.5007              -3.2080
     0.0725               0.2378
     0.0001               0.0010
```

These epipoles are out of the physical plane. And now, I can also plot the corresponding epipolar lines; the epipoles intersect outside the image boundary in each image.

$$Epipolar\ lines:$$

$$l_1 = transpose(x_2)\ F$$

$$l_2 = F\ x_1$$



The same thing holds for the second pair of images; the epipoles are outside the physical plane.

QUESTION 2: FUNDAMENTAL MATRIX REFINEMENT – RANSAC

In the previous section, I found eight pairs of features in image one and image two and found the fundamental matrix based on these data; however, such a method is not doable when we are given a new pair of images, and we want to find the fundamental matrix precisely and quickly, In this case, usually a feature matching algorithm (such as SIFT, or SURF features) is performed, and the corresponding features in pairs of images are found. So, **in practical situations, the ground truth is not given, and correspondences between two images are unclear**, so there must be a way to compute interest points and predict matches between stereo image pairs. One should keep in mind that these feature matching algorithms are not error-free and might not be 100% precise. Hence, we need to find **a robust way** of finding the fundamental matrix.

RANSAC is an iterative algorithm that nondeterministically fits a model to a random sample of points taken from the dataset.

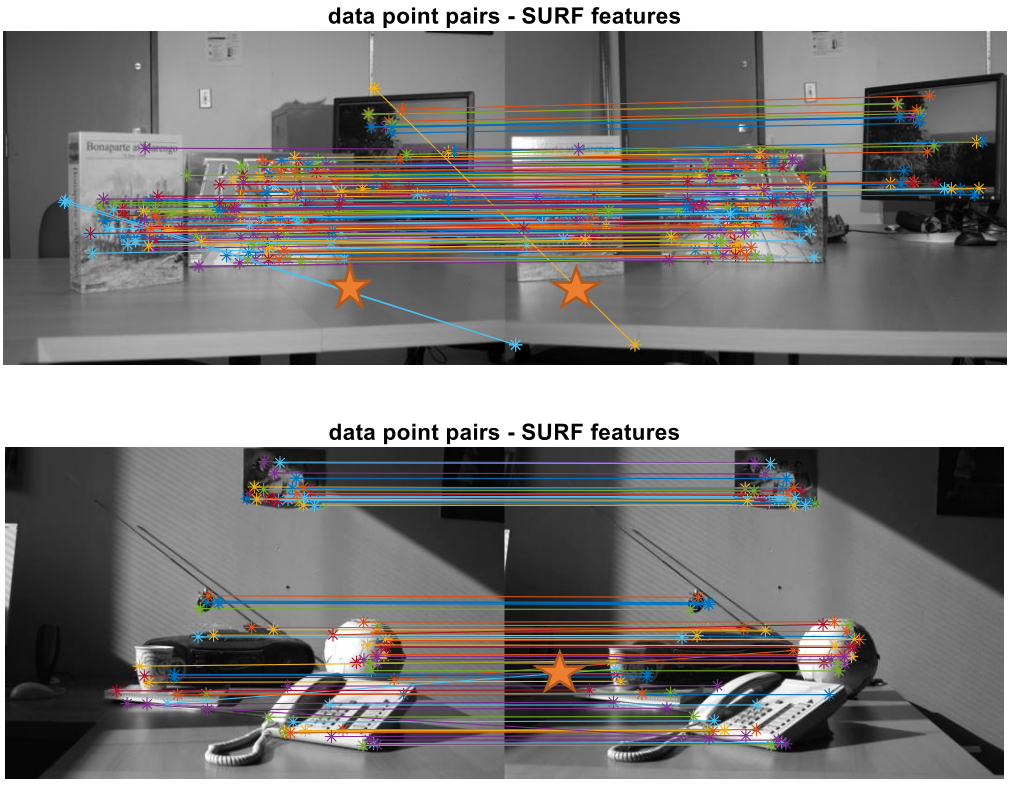


Figure 1 You can see that when feature mapping algorithm is used to find the corresponding features in pairs of images, there might be some errors (some sample errors are magnified by using orange stars beside them); we want to use RANSAC to make sure that these outliers are not affecting our model (the proposed model is the fundamental matrix in this case).

Here, I tried two different algorithms for RANSAC.

1) First version of RANSAC - MATLAB code: "F_ransac.m"

For i = 1:1:max_num_iterations

Select 8 pairs of points and build Fundamental matrix (F)

Error = x2' F x1

For g = 1:1: size of all features

If (error for this g^{th} pair of features <= 0.01 (arbitrary))

Save feature as inlier

End

end

If (number of inliers > max number of inliers until now)

F_ best = find fundamental matrix by using all these inliers.

And save and update the max number of inliers until now

end

end

At the first step, I implemented the algorithm mentioned above. Then I tried to compare my results (" `F_best`") with the results of the build-in MATLAB function, which generates the RANSAC model of the fundamental matrix ("`F_best_matalb`"). Here are the results for the first pair of images.

Note that the fundamental matrix that I found is rank two.

```
F_best =

  3×3 single matrix

   -0.0000   -0.0000    0.0007
   -0.0000    0.0000    0.0351
    0.0025   -0.0309   -0.9989

rank of F obtained from ransac is =
 2.000000

F_best_matalb =

    0.0000    0.0000   -0.0019
    0.0000    0.0000   -0.0358
   -0.0012    0.0311    0.9989
```

And now, here are the results for the second pair of images.

```
F_best =

  3×3 single matrix

   -0.0000    0.0000   -0.0006
    0.0000   -0.0000   -0.0682
   -0.0016    0.0656    0.9955

rank of F obtained from ransac is =
 2.000000

F_best_matalb =

    0.0000    0.0000   -0.0016
    0.0000   -0.0000   -0.0675
   -0.0007    0.0647    0.9956
```

So, you can see that there is a high consistency here.

2) *Second version of RANSAC* - MATLAB code: "F_ransac1.m"

For i = 1:1: max_num_iterations (it can be 2000 iterations)

Select 8 pairs of points and build F

Error = x2' F x1

For g = 1:1: size of all features (180 in this case)

If (error for a pair of features <= 0.01)

Save feature as inliers

End

end

If (number of inliers> Cmax(Cmax could be equal to 30))

Build a model for fundamental matrix called F_temp

If (error until now >= error of this model F_temp by considering F_temp)

F_ best = F_temp

Update error until now

end

end

end

As I did for the first version of my code, here again, I implemented this algorithm, and then I tried to compare my results with the build-in MATLAB function:

For the first version of image pairs:

```
F_best =

  3×3 single matrix

    0.0000    0.0001   -0.0184
   -0.0001   -0.0000    0.0350
    0.0182   -0.0301   -0.9986

rank of F obtained from ransac is
 2.000000

F_best_matalb =

    0.0000    0.0000   -0.0013
    0.0000    0.0000   -0.0349
   -0.0019    0.0304    0.9989
```

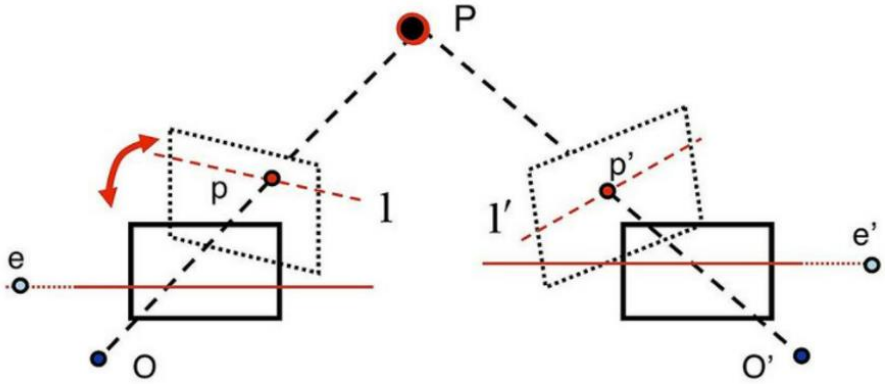
For the second version of image pairs:

```
F_best =

  3×3 single matrix

    0.0000    0.0000   -0.0014
    0.0000   -0.0000   -0.0679
   -0.0009    0.0652    0.9956

rank of F obtained from ransac is =
 2.000000

F_best_matalb =

   -0.0000    0.0000   -0.0009
    0.0000   -0.0000   -0.0656
   -0.0013    0.0624    0.9959

``
```

- ***I would rather be working with the first version of RANSAC; it seems to give me more stable results over multiple runs of my code.***

QUESTION 3: RECTIFICATION

Image rectification is a transformation process used to project images onto a common image plane, and there are many strategies for transforming images to the common plane. This study tries to rectify an image by sending its epipole to infinity. If we do so, the epipolar lines in a pair of images would be horizontal, and so the search problem for finding the corresponding features would be solved efficiently (**correspondence problem**). So, in other words, a pair of images is thought to be rectified when its epipolar lines coincide. One of the essential advantages of rectification is that the search for correspondences is done along the rectified images' rows or columns.



Model used for image rectification example.

A typical way to send epipoles to infinity is to apply a homographic transformation that will send the epipole of the original image to a point in infinity ([w,0,0] in 2D homogenous coordinate).

$$H_1 = \begin{bmatrix} 1 & 0 & 0 \\ -\frac{y_{e1}}{x_{e1}} & 1 & 0 \\ -\frac{1}{x_{e1}} & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ -\frac{y_{e1}}{x_{e1}} & 1 & 0 \\ -\frac{1}{x_{e1}} & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{e1} \\ y_{e1} \\ 1 \end{bmatrix} = \begin{bmatrix} x_{e1} \\ 0 \\ 0 \end{bmatrix} \rightarrow send\ epipole\ 1\ to\ infinity$$

$$H_2 = \begin{bmatrix} 1 & 0 & 0 \\ -\frac{y_{e2}}{x_{e2}} & 1 & 0 \\ -\frac{1}{x_{e2}} & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ -\frac{y_{e2}}{x_{e2}} & 1 & 0 \\ -\frac{1}{x_{e2}} & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{e2} \\ y_{e2} \\ 1 \end{bmatrix} = \begin{bmatrix} x_{e2} \\ 0 \\ 0 \end{bmatrix} \rightarrow send\ epipole\ 2\ to\ infinity$$

I applied this homographic transformation to all pixels in the image and determined the resulting image's bounds and dimensions. Then, I used the inverse homography matrix to map the coordinates of the rectified image to its corresponding pixel in the original image and used warp to interpolate the pixel values. I have included the results that I got on the following pages:

If you look at the rectified images in the following pages, you may note that the first set of images may not give us perfect results, which is expectable since the first pair of images are taken using cameras that have great amount of translation which makes things a little complicated. However, you can see that for the second pair of images the results are promising.

Figure 2 Rectified images - the first set of images

Figure 3 Both rectified images - the first set of images

Figure 4 Rectified images - the second set of images

Figure 5 Both rectified images - the second set of images

QUESTION 4: RECONSTRUCTION

The very first step in this problem is to find the **essential matrix**; as we have the fundamental matrix, it would not be hard to find the corresponding essential matrix; to do so, it is important to know about the relationship between the essential matrix and the fundamental matrix which could be stated as below:

$$E = transpose(K)\ F\ K$$

$$where\ K\ is\ the\ intrinsic\ matix$$

$$K = \begin{bmatrix} fm_x & 0 & -p_x \\ 0 & fm_y & -p_y \\ 0 & 0 & 1 \end{bmatrix} where\ f\ is\ the\ focal\ lenght$$

We know about focal length, which is f = 31mm for stereo pair one, and for stereo pair two, the focal length f is 32mm, we know about Px and Py, which are the position of the central pixels of images, and mx and my are also given to us ($m_x = m_y = \frac{1}{31.2*10^{-3}}$). So, essentially, we know about this K (K contains the intrinsic parameters of the cameras), so we can have the essential matrix. What are the rotation matrix and the translation matrix that has formed this essential matrix? To answer this question, at first, I would like to mention that I looked at these two references to find an appropriate way to decompose essential matrix to get to the correct version of the translation and the rotation matrix.

1.Hartley, R.I., R. Gupta, and T. Chang. Stereo from uncalibrated cameras. in CVPR. 1992.

2. R. Hartley and A. Zisserman. Multiple View Geometry in Computer Vision. Cambridge University Press, ISBN: 0521540518. 2004."

There are important statements in these references that elucidate how one should decompose essential matrix in a meaningful way; some crucial messages are attached.

Result 9.17. *A 3×3 matrix is an essential matrix if and only if two of its singular values are equal, and the third is zero.*

Basically, this statement suggests that I need to modify our essential matrix so that it could meet this constraint. This is how I managed to meet this constraint.

```
[U,S,V]= svd(E);
e = (S(1,1) + S(2,2)) / 2;
S(1,1) = e;
S(2,2) = e;
```

```
S(3,3) = 0;
E = U * S * V';
```

Now that the essential matrix has been modified, such an instruction can be followed to find the rotation and the translation matrices:

Proof. This is easily deduced from the decomposition of $\mathtt{E}$ as $[\mathbf{t}]_\times \mathtt{R} = \mathtt{SR}$, where $\mathtt{S}$ is skew-symmetric. We will use the matrices

$$\mathtt{W} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \mathtt{Z} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}. \tag{9.13}$$

It may be verified that $\mathtt{W}$ is orthogonal and $\mathtt{Z}$ is skew-symmetric. From Result 4.1-($p581$), which gives a block decomposition of a general skew-symmetric matrix, the 3×3 skew-symmetric matrix $\mathtt{S}$ may be written as $\mathtt{S} = k\mathtt{U}\mathtt{Z}\mathtt{U}^\mathsf{T}$ where $\mathtt{U}$ is orthogonal. Noting that, up to sign, $\mathtt{Z} = \mathrm{diag}(1,1,0)\mathtt{W}$, then up to scale, $\mathtt{S} = \mathtt{U}\,\mathrm{diag}(1,1,0)\mathtt{W}\mathtt{U}^\mathsf{T}$, and $\mathtt{E} = \mathtt{S}\mathtt{R} = \mathtt{U}\,\mathrm{diag}(1,1,0)(\mathtt{W}\mathtt{U}^\mathsf{T}\mathtt{R})$. This is a singular value decomposition of $\mathtt{E}$ with two equal singular values, as required. Conversely, a matrix with two equal singular values may be factored as $\mathtt{S}\mathtt{R}$ in this way. □

Since $\mathtt{E} = \mathtt{U}\,\mathrm{diag}(1,1,0)\mathtt{V}^\mathsf{T}$, it may seem that $\mathtt{E}$ has six degrees of freedom and not five, since both $\mathtt{U}$ and $\mathtt{V}$ have three degrees of freedom. However, because the two singular values are equal, the SVD is not unique – in fact there is a one-parameter family of SVDs for $\mathtt{E}$. Indeed, an alternative SVD is given by $\mathtt{E} = (\mathtt{U}\,\mathrm{diag}(\mathtt{R}_{2\times2}, 1))\,\mathrm{diag}(1,1,0)(\mathrm{diag}(\mathtt{R}_{2\times2}^\mathsf{T}, 1))\mathtt{V}^\mathsf{T}$ for any 2×2 rotation matrix $\mathtt{R}$.

Result 9.18. *Suppose that the SVD of* $\mathtt{E}$ *is* $\mathtt{U}\,\mathrm{diag}(1,1,0)\mathtt{V}'$. *Using the notation of (9.13), there are (ignoring signs) two possible factorizations* $\mathtt{E} = \mathtt{SR}$ *as follows:*

$$\mathtt{S} = \mathtt{U}\mathtt{Z}\mathtt{U}^\mathsf{T} \quad \mathtt{R} = \mathtt{U}\mathtt{W}\mathtt{V}^\mathsf{T} \quad \textit{or} \quad \mathtt{U}\mathtt{W}^\mathsf{T}\mathtt{V}^\mathsf{T}. \tag{9.14}$$

The factorization (9.14) determines the $\mathbf{t}$ part of the camera matrix $\mathtt{P}'$, up to scale, from $\mathtt{S} = [\mathbf{t}]_\times$. However, the Frobenius norm of $\mathtt{S} = \mathtt{U}\mathtt{Z}\mathtt{U}^\mathsf{T}$ is $\sqrt{2}$, which means that if $\mathtt{S} = [\mathbf{t}]_\times$ *including scale* then $\|\mathbf{t}\| = 1$, which is a convenient normalization for the baseline of the two camera matrices. Since $\mathtt{S}\mathbf{t} = \mathbf{0}$, it follows that $\mathbf{t} = \mathtt{U}\,(0, 0, 1)^\mathsf{T} = \mathbf{u}_3$, the last column of $\mathtt{U}$. However, the sign of $\mathtt{E}$, and consequently $\mathbf{t}$, cannot be determined. Thus, corresponding to a given essential matrix, there are four possible choices of the camera matrix $\mathtt{P}'$, based on the two possible choices of $\mathtt{R}$ and two possible signs of $\mathtt{t}$. To summarize:

Result 9.19. *For a given essential matrix* $\mathtt{E} = \mathtt{U}\,\mathrm{diag}(1, 1, 0)\mathtt{V}^\mathsf{T}$, *and first camera matrix* $\mathtt{P} = [\mathtt{I} \mid \mathtt{0}]$, *there are four possible choices for the second camera matrix* $\mathtt{P}'$, *namely*

$$\mathtt{P}' = [\mathtt{U}\mathtt{W}\mathtt{V}^\mathsf{T} \mid +\mathbf{u}_3] \text{ or } [\mathtt{U}\mathtt{W}\mathtt{V}^\mathsf{T} \mid -\mathbf{u}_3] \text{ or } [\mathtt{U}\mathtt{W}^\mathsf{T}\mathtt{V}^\mathsf{T} \mid +\mathbf{u}_3] \text{ or } [\mathtt{U}\mathtt{W}^\mathsf{T}\mathtt{V}^\mathsf{T} \mid -\mathbf{u}_3].$$

So, you can see that one can follow the algorithm mentioned above to find the rotation matrix and the translation matrix; however, these matrices are not unique, and one can have four solutions, where it is shown that a **reconstructed point X will be in front of both cameras in one of these four solutions only**! **Thus, testing with a single point to determine if it is in front of both cameras is sufficient to decide between the four different solutions for the camera matrix P**. This is the part of the code that helped me get to these four possible P matrices. Here, I just followed the algorithm used in the references to obtain rotation and translation matrices.

```
[U, ~, V] = svd(E);
W = [0 -1 0; 1 0 0; 0 0 1];
Z = [0 1 0; -1 0 0; 0 0 0];

% Two possible rotation matrices
R1 = U * W * V';
R2 = U * W' * V';
% Force rotations to have the needed properties
if det(R1) < 0
    R1 = -R1;
end
if det(R2) < 0
    R2 = -R2;
end
```

```
% Two possible translation vectors
S1 =  U(:,3);
S2 = -U(:,3);

% Which P matrix is the best version?
P(:,:,1) =  K*[R1, S1];
P(:,:,2) =  K*[R1, S2];
P(:,:,3) =  K*[R2, S1];
P(:,:,4) =  K*[R2, S2];
```

Note that we suppose that the first camera's P matrix is P1 = K*horzcat([1,0,0;0,1,0;0,0,1], [0;0;0]); we suppose that the first camera's coordinate system is the world's coordinate system. Hence the rotation matrix is Identity, and it has no translation. We found in the previous stage the P matrix related to the second camera (P2 = P).

Note that we know that the **null space of P is the camera position**, so basically, we have the camera positions, and this is enough to clarify which of the four P matrices is the solution.

```
% Find camera position
NP1 = null(P1);
cameraCenter1 = NP1(1:3)/NP1(4)

NP2 = null(P2(:,:,k));
cameraCenter2 = NP2(1:3)/NP2(4)
```

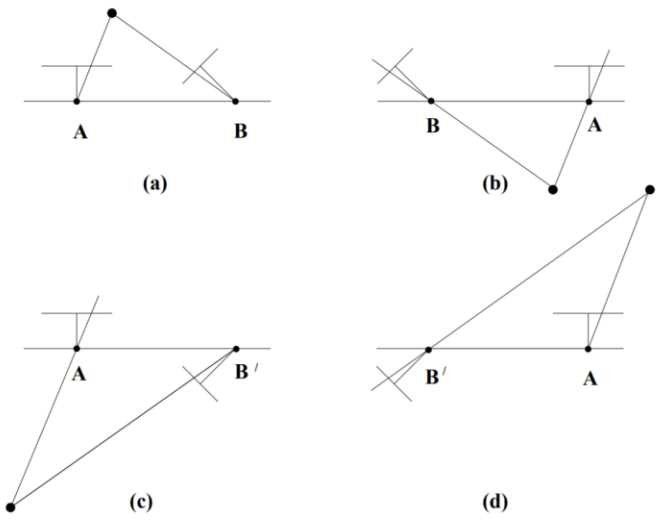


Figure 6 The four possible solutions for calibrated reconstruction from E. Note, only in (a) is the reconstructed point in front of both cameras

The final step to reach the goal of this study is to use **triangulation** to recover the depth of objects seen in the scene. In this regard, I used such a formulation to recover depth.

Concatenate the 2D points from both images

$$\begin{bmatrix} y\boldsymbol{p}_3^\top - \boldsymbol{p}_2^\top \\ \boldsymbol{p}_1^\top - x\boldsymbol{p}_3^\top \\ y'\boldsymbol{p}_3'^\top - \boldsymbol{p}_2'^\top \\ \boldsymbol{p}_1'^\top - x'\boldsymbol{p}_3'^\top \end{bmatrix} \boldsymbol{X} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

In the following pages, I will share some of the results.

- **First Image Pair:**

Let us select some pair of features in the images and try to reconstruct their depth using the algorithm mentioned above:

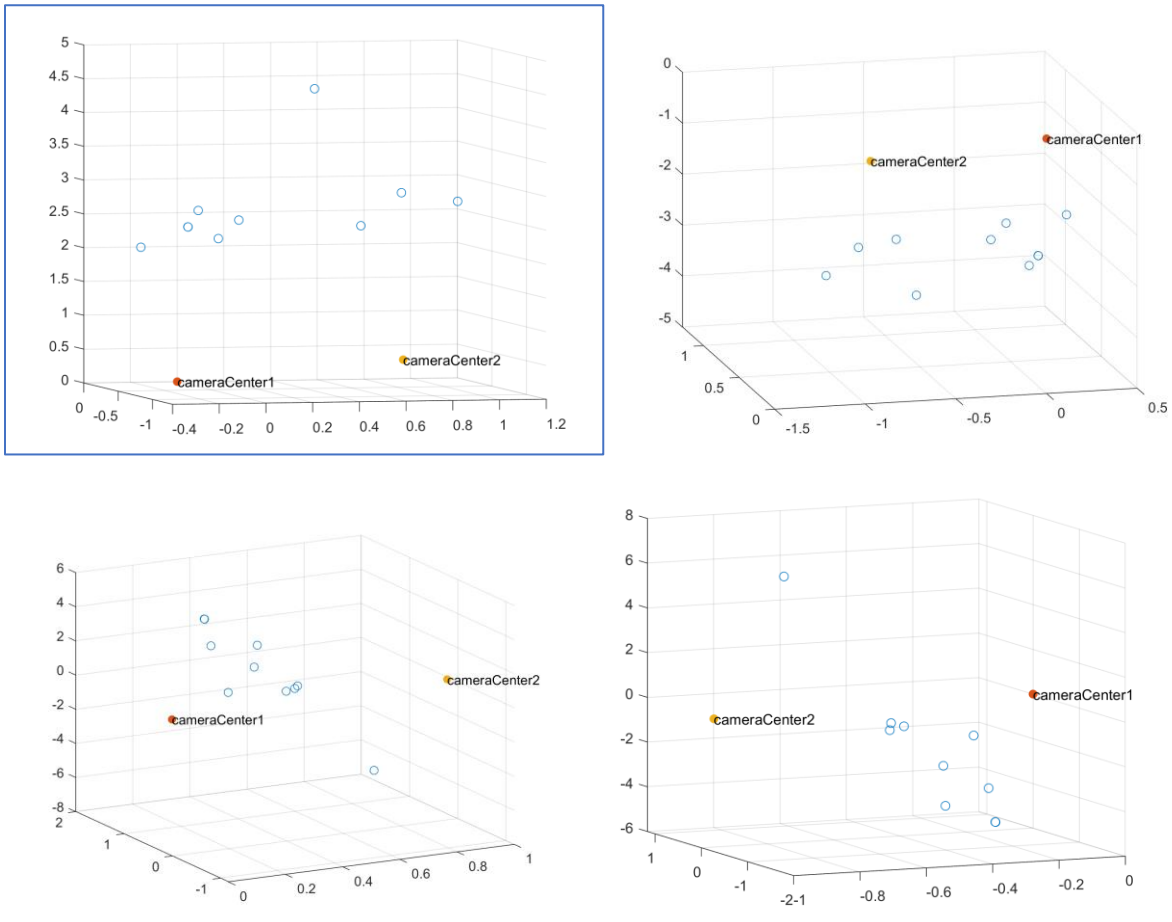


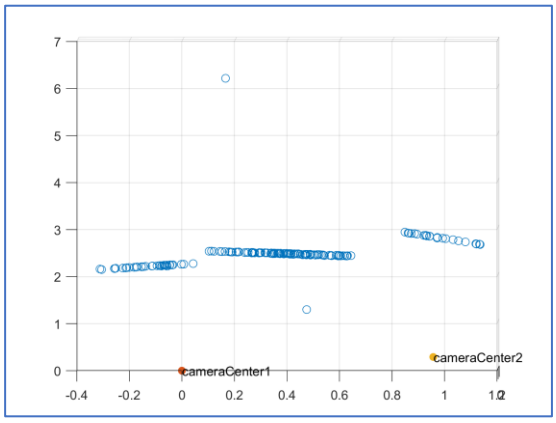
Figure 7 These are the camera positions and depth of 9 pairs of features I have chosen.

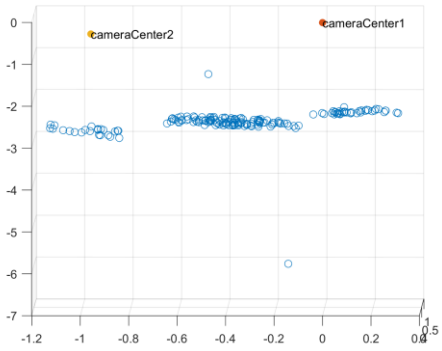
It seems like the left-up panel is acceptable, so I just showed the value of depth on the corresponding image pixel under analysis.

You can see that the results are reasonable.

Also, I used the features that the SURF detector gave us and tried to predict depth for those. You **can see the depth pattern of the two books and the monitor** here. Obviously,

Let us start with selecting the best P:





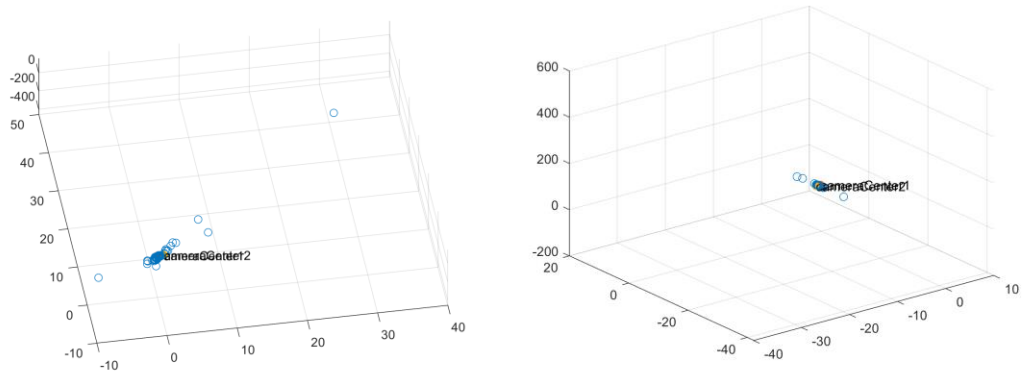


Figure 8 These are the camera positions and depth of all SURF pairs of features.

Here, it seems like the first P, which corresponds to the left-up model, is working well.

- **Second Image Pair:**

Let us select some features in the images and reconstruct their depth using the algorithm mentioned earlier.

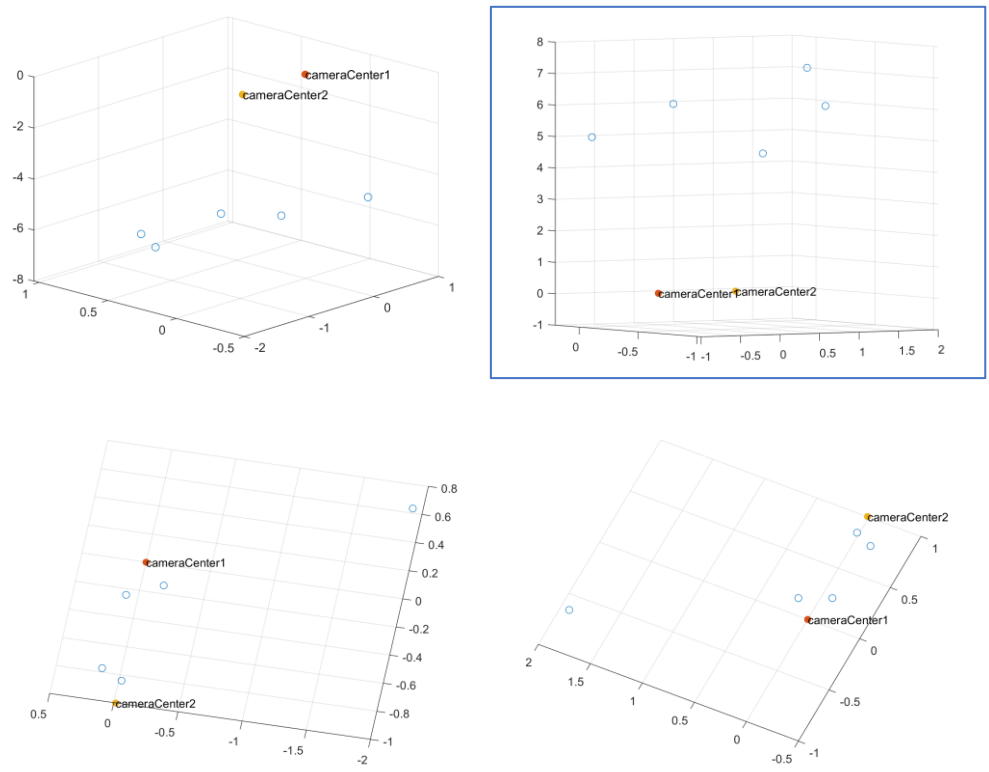
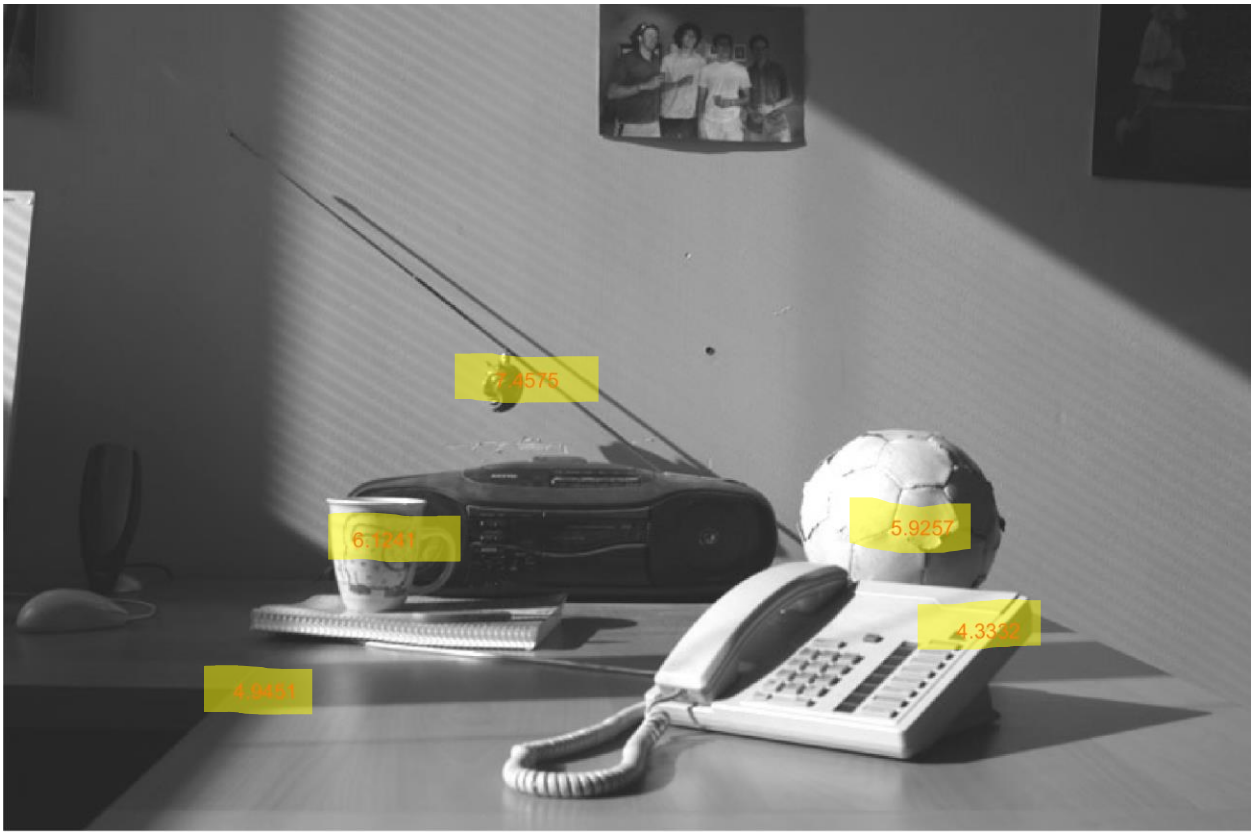


Figure 9 These are the camera positions and depth of 5 pairs of features I have chosen.

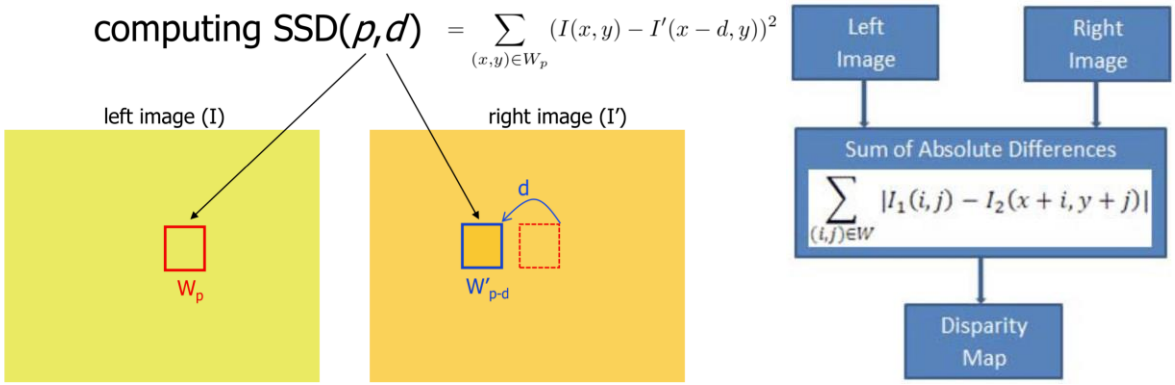
It seems like the right-up panel is acceptable, so I just showed the value of depth on the corresponding image pixel under analysis.



It is evident that the results save the order of objects in the sense of depth.

QUESTION 5: DENSE RECONSTRUCTION – DISPARITY

We expect the closer objects (smaller depths) to correspond to more enormous disparities. This part of the assignment aims to find the disparity map and its corresponding depth map. In this part, we are working with the rectified images, and hence we know that shifts along horizontal scan lines describe the correspondences. The algorithm to find these correspondences and accordingly the disparity is illustrated using these following images:



So, one can estimate the disparity easily by solving a minimization problem (in this problem, I used the sum of absolute differences). You define a fixed size window in image one and a window in image 2, you move the window located in image two by d, and then you try to find the d that minimizes the error between values of the window in the first image and the values within the window in the second image when the error is minimized this d value would be the optimal value of disparity.

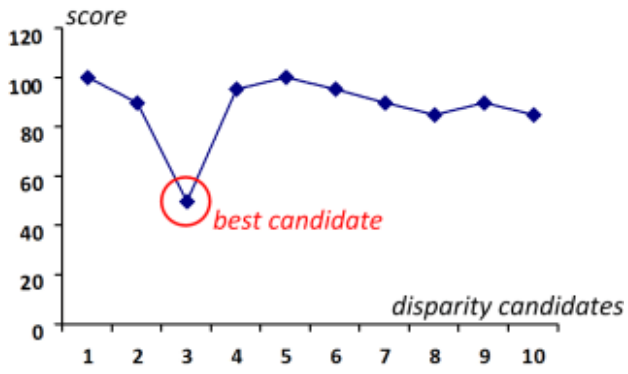
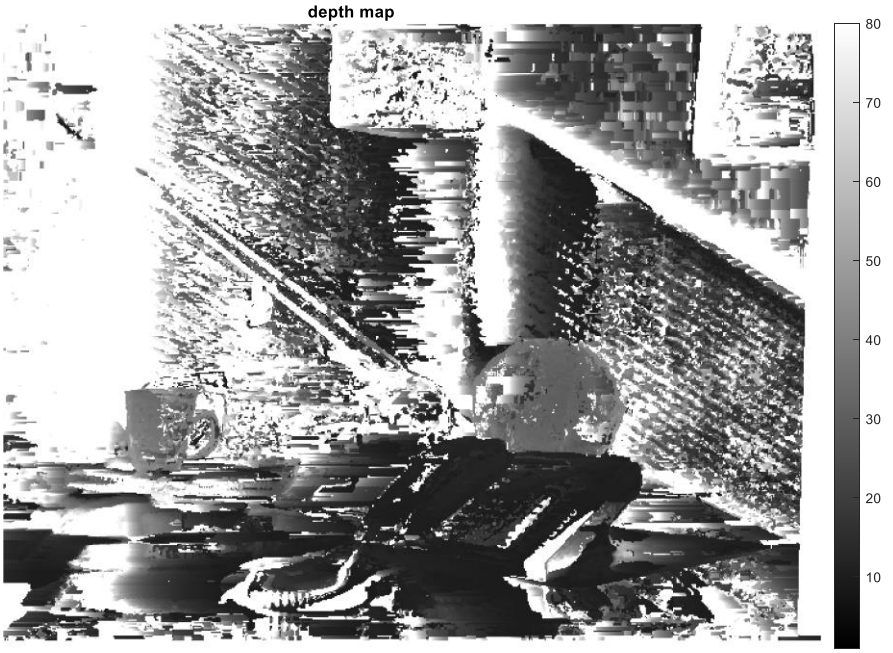


Figure 10 To build the disparity map, one needs to find d that minimizes the SAD (sum of absolute differences).

Here are the results for the second pair of images:



You can clearly see that in the disparity map, the objects near the camera have higher disparity, and objects far from the camera have a small amount of disparity. This is entirely in line with what we would expect.

The biggest problem seems to happen when a large region has the same intensity for all pixels in it. In this case, intensity-based error minimization might pick tiny differences and assign somewhat arbitrary disparities to the region.

I tried to find the disparity map or the depth map for the first set of pairs using the same parameters.

The results for the second pair of images do not seem as well as the first pair; I kind of expected the results since, at the very first step, this first pair consists of images that were taken from various viewpoints, there was lots of translation, and this fact even made the rectified images not to be perfect which means that we might not accept excellent results for this section.