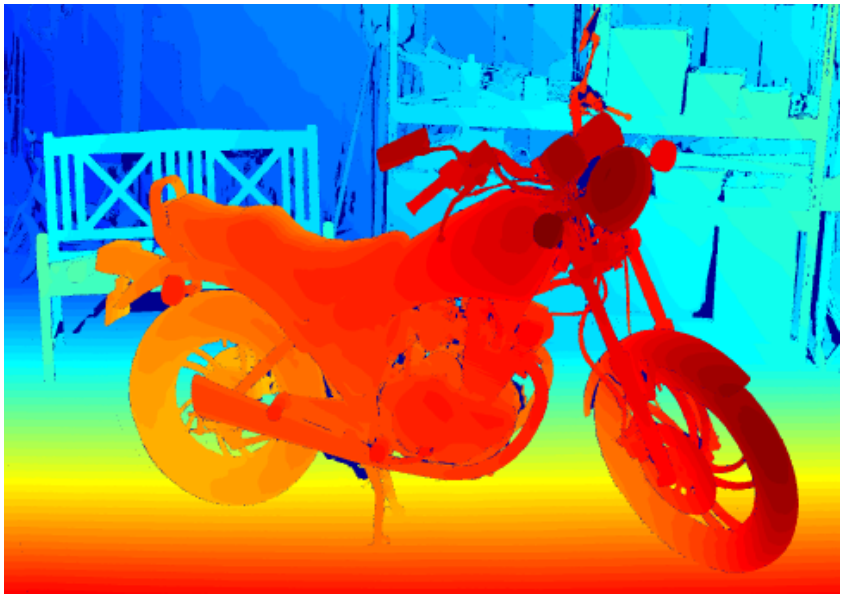
Computer Vision Challenge

SS2019



Group member (G36):

Bowen Li 03709969

Chensheng Chen 03712507

Tao Tang 03685664

Helin Cao 03714812

Contents

1. Introduction

Stereo vision is the process of extracting 3D information of a scene from images. It has wide applications in entertainment, advanced driver assistance systems and visual navigation, where the distances of objects are of high interest.

The depth information can be derived from a pair of images or multiple images. Disparity is defined as the distance between corresponding points when the two images are superimposed. And the disparities of all pixels in the image form the disparity map. The values in this disparity map are inversely proportional to the scene depth at the corresponding pixel location.

2. Euclidean motion computation

2.1 R, T computation

This task can be done in the following steps:

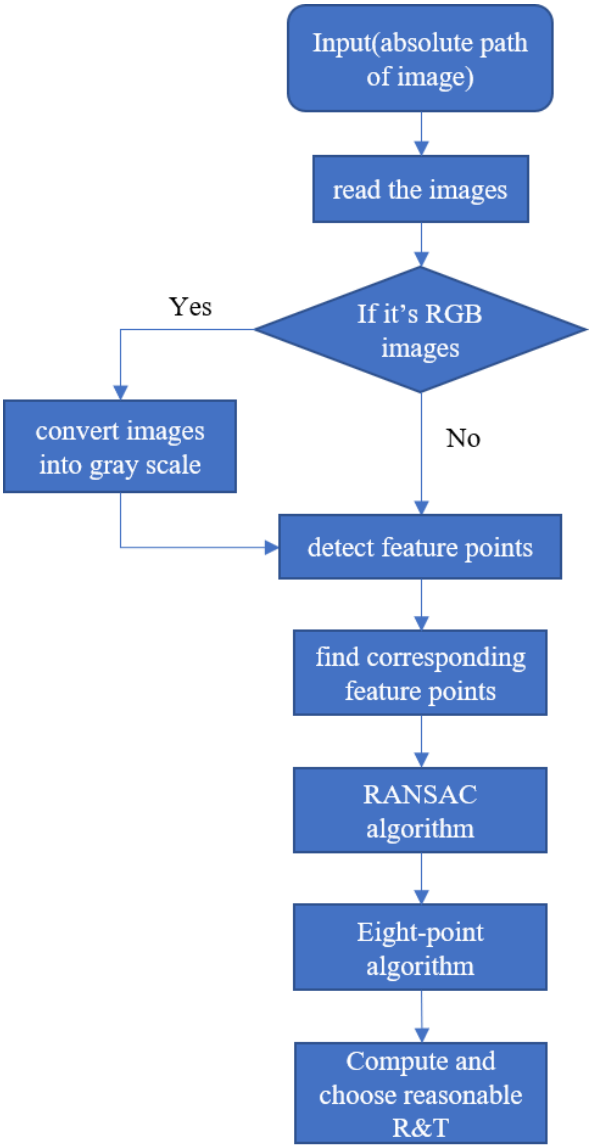


Figure 1.1 R, T computation

1. Convert the original image into gray scale if it is an RGB image.
2. Detect feature points in image using Harris Detector.
3. Find correspondence between different feature points using Normalized Cross Correlation.
4. Use RANSAC algorithm to find eight most suitable point.
5. Compute the essential matrix E using eight-point algorithm, those eight points come from step d).
6. Compute all possible combinations of rotation matrix R and translation vector T and choose the reasonable pair of R and T.

7. As requested in the project, T must be shown in the unit of meter. The variable baseline is given in 'calib.txt', whose unit is mm, so we can use the following formula to convert T:

$$\mathrm{T} \leftarrow -\frac{T \times baseline}{1000} \tag{2.1}$$

2.2 Important algorithm

The most important algorithms are Eight-point algorithm and RANSAC algorithm. They will be interpreted in details as follows:

2.2.1 Eight-point algorithm

Eight-point algorithm is a method which can reconstruct the camera motion from a pair of images which are two views of the scene. For a calibrated camera, we know the epipolar constraint:

$$x_2{}^T \hat{T} R\, x_1 = 0 \tag{2.2}$$

E is called essential matrix, which is defined as

$$E = \hat{T} R \in \mathbb{R}^{3\times 3} \tag{2.3}$$

E can be rewritten as a scalar product in the elements of E and coordinates of corresponding points. We define that

$$E^s = (e_{11}, e_{21}, e_{31}, e_{12}, e_{22}, e_{32}, e_{13}, e_{23}, e_{33})^T \in \mathbb{R}^9 \tag{2.4}$$

is the element vector of E, and

$$a \equiv \boldsymbol{x}_1 \otimes \boldsymbol{x}_2 \tag{2.5}$$

is the Kronecker product of $\mathbf{x}_1$ and $\mathbf{x}_2$, which can be defined as

$$a = (x_1x_2, x_1y_2, x_1z_2, y_1x_2, y_1y_2, y_1z_2, z_1x_2, z_1y_2, z_1z_2)^T \in \mathbb{R}^9 \tag{2.6}$$

And the epipolar constrains can be rewritten as

$$\boldsymbol{x}_2{}^T E \boldsymbol{x}_1 = a^T E^s = 0 \tag{2.7}$$

For n corresponding points, we can write

$$\chi E^s = 0, with\ \chi = (a_1, a_2, \dots, a_n) \tag{2.8}$$

That means the elements of vector E is actually the null space of matrix χ. For such homogeneous linear equations, there are only two possible situations of solution, the unique zero solution or infinity number of solutions. Actually, we need a unique non-zero solution up to a scaling vector. In this case, the rank of the matrix χ must be 8, which means we need at least eight independent points.

And the eight-point algorithm can be generalized in following steps:

1. Compute an approximation of the essential matrix.
   Construct the matrix χ and find the vector E^s which minimizes $||\chi E^s||$ as the ninth column of V_χ in the SVD $\chi = U_\chi \Sigma_\chi V_\chi^T$. Unstack E^s into 3×3-matrix E.
2. Project onto essential space. Since in the reconstruction, E is only defined up to a scalar, we project E onto the normalized essential space by replacing the singular values σ1, σ2, σ3 with 1, 1, 0.
3. Recover the displacement from the essential matrix. There are four possible solutions for rotation and translation, choose the reasonable value.

2.2.2 RANSAC algorithm

RANSAC, RANdom SAmple Consensus, is a useful mathematics method, which is mainly used in Computer Vision. The main idea of RANSAC is to estimate the parameter of model from a set of observed data points which contains the outliers by iteration.

We must consider the parameter when we use RANSAC. Actually, we need to choose a suitable value of parameter ω, which means the percentage of number of points belonging to consensus set in total dataset. And assuming that n is the number of points which are chosen, p is the probability of success after executing the algorithm for k times, we can get the equation:

$$p = 1 - (1 - \omega^n)^k \tag{2.9}$$

Then we know that if we want to improve the probability of success, we should execute the iteration more than one time.

The RANSAC can be generalized in a few steps:

1. Assume that some of points in dataset belong to consensus set.
2. Calculate the consensus model.
3. Try other points which are not chosen and check if they belong to consensus set.
4. Save the number of points which belong to consensus set.
5. Repeat a) to d) several times.
6. Choose the model which has the maximum number of points.

Notice: As we know, the camera moves horizontally, so R should be (or at least close to) identity matrix and T should be $\begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix}$. But because of the random sampling of RANSAC algorithm, the results are not deterministic, which means we could get different values if we re-run the program.

2.3 Function Description

```
gray_image = rgb_to_gray(input_image)
```

Input: image matrix (gray or color).

Output: gray image.

Description: convert RGB image to gray scale. If the image is already in gray scale, then return it directly.

```
merkmale = harris_detektor(input_image, varargin)
```

Input: gray scaled image.

Output: detected corner points.

Description:

1. Use Sobel filter to compute the gradient of image.
2. Obtain the Harris matrix by computing the Harris value of each pixel.
3. Eliminate the pixels whose value is smaller than the threshold ‘tau’.
4. Use Cake matrix to filter the ‘crowded points’ in specific area, only the strongest pixel remains.
5. Return these corner points as output.

```
Korrespondenzen =
punkt_korrespondenzen(I1,I2,Mpt1,Mpt2,varargin)
```

Input: two images and their detected corner points.

Output: corresponding corner points in image1 and image2.

Description: for each corner point in image1, find the corresponding corner point in image2, which has the least NCC value. Return these point pairs as output.

```
[Korrespondenzen_robust] = F_ransac(Korrespondenzen,
varargin)
```

Input: founded corresponding point pairs.

Output: robust point pairs.

Description: use Ransac algorithm to choose robust corresponding point pairs. By this means, we can reduce the influence of outliers.

```
[EF] = achtpunktalgorithmus(Korrespondenzen, K1, K2)
```

Input: corresponding point pairs, calibration matrix

Output: essential matrix E.

Description: use 8-point-algorithm to compute essential matrix E. Notice that the coordinates are calibrated with the help of calibration matrix at the beginning.

`[T1, R1, T2, R2]=TR_aus_E(E)`

Input: essential matrix E

Output: rotation matrix R1, R2 and translation vector T1, T2.

Description: with positive Rz and negative Rz, we can derive 2 candidates of R and T.

`[T, R] = rekonstruktion(T1, T2, R1, R2, Korrespondenzen, K1, K2)`

Input: T1, T2, R1, R2 and calibration matrix.

Output: reasonable combination of R and T.

Description: with derived T1, T2, R1, R2, we have four different combinations of Euclidean motion. To determine which combination is most reasonable, we compute the depth information λ of these points in 3D-space. We choose the combination, which has the maximum number of positive λ.

3. Disparity Computation

3.1 SAD (Sum of absolute Differences)

SAD[1] is a simple algorithm in image matching. We can measure the similarity between two pixel blocks by computing the sum of L1 norm of every pixel in these two blocks.

$$SAD = \sum_{x=0}^{b-1} \sum_{y=0}^{h-1} |B_2(x,y) - B_1(x,y)| \tag{3.1}$$

The steps of this algorithm are as followings;

1. Create a window, the window size is given by variable `window_size` in the program. It can be seen like a kernel window.

2. Fix the window in right image, select all pixels which are covered by the window.
3. Fix the same size window in the corresponding position in the left image, select all pixels covered by this window.
4. Compute the sum of absolute intensity differences between two windows, use L1 norm.
5. Slide the left image window from left to right, repeat step3 and step4. If the sliding window reaches the maximum searching area (given by variable `ndisp` in calib.txt) or the edge of image, stop the sliding.
6. Find the window in left image which produces minimum SAD value. The pixel distance d between two window centers can be treated as the disparity.

3.2 Improvement of SAD

1. The running time using SAD to process big image, for example 'motorcycle' in this project, is rather long, almost 50 minutes, since it has to compute every single pixel disparity. Now, we use Block Matching to speed up SAD. Once we have computed the pixel disparity, assign it all pixels which are covered by the window. In other words, we compute disparity value of 9 pixels (if window length is 1) at each time. By this means, the running time can be decreased significantly. It is also a tradeoff between running time and accuracy. There will be more noisy blocks in disparity map, but we can somehow fix it by using Gaussian filter at the end.
2. By using the SAD algorithm above we can get right image disparity map. But in this project we want to have left image disparity. So we can use `fliplr(right_image)` and `fliplr(left_image)` to preprocess the two image. Flip them again before the output and we can get left image disparity.
3. We have also designed the window size, which is adaptive to the width of the image.

3.3 Function Description

```
[D, R, T] = disparity_map(scene_path)
```

Input: relative path of image folder.

Output: Euclidean motion and disparity matrix.

Description: first of all, we want to extract all information we need for computation of disparity map and Euclidean motion. We can get calibration matrix, baseline and maximum disparity in 'calib.txt' document. With calibration matrix of both cameras, we can compute rotation and translation matrix (refer to 2.1). At least, call `batchmatching_inv` to compute the whole disparity map. Return R, T, and normalized D as output of this function.

```
disparity = batchmatching_inv(left_im, right_im, d_max)
```

Input: left image, output image, maximum disparity.

Output: disparity map of left image.

Description: use improved SAD algorithm to compute the disparity map. At the end of function, we have also used Gaussian filter to make the disparity map more smooth.

```
image_norm = normalization(image)
```

Input: disparity map or ground truth.

Output: normalized disparity map or ground truth.

Description: map the value of disparity map or ground truth to the interval [0, 255]. So they can be used to compute PSNR. The normalization formula is:

$$M(normalized) = \frac{M - min}{max - min} \times 255 \tag{3.2}$$

where M is unnormalized image matrix, min and max denotes the minimum and maximum value of M respectively.

3.4 Gaussian filter

There will be some miscalculated disparity value in our images. Here, the gaussian filter should be used to decrease the effect of noise. Since we can't use Toolbox in this project, we have to write our own gaussian filter.

First of all, the gaussian filter windows should be set. The steps for generating gaussian filter are shown as followings:

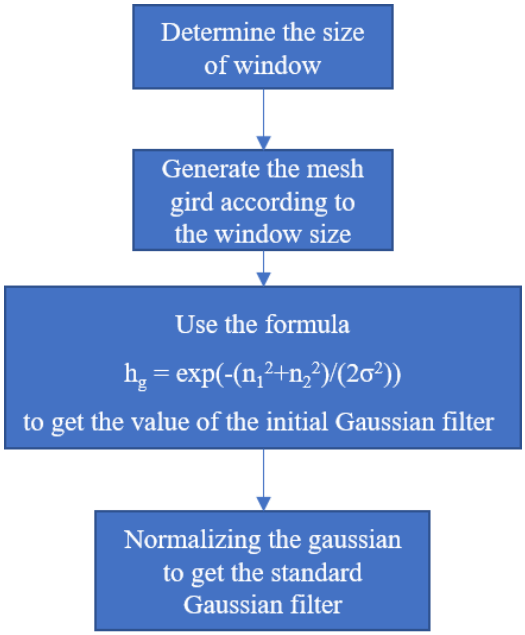


Figure 3.1 Gaussian filter

The function `gaussian_filter` is equal to the function `fspecial` (with gaussian as input parameter). To deal with edge effect, the edge of the original image should be extended.

The function `two_d_filter` is equal to the `imfilter`. The steps for generating `two_d_filter` is as following:

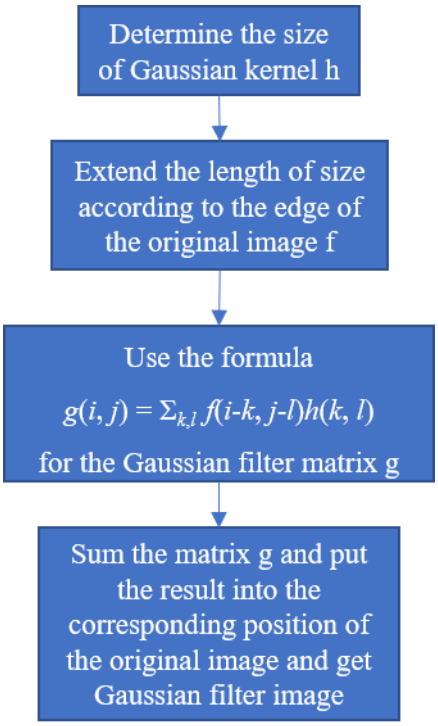


Figure 3.2 2D filter

4. Verification

4.1 PSNR

PSNR is often used to measure the similarity between two images. It is defined via the mean squared error (MSE). Given the disparity map matrix D and ground truth matrix G, where D is output of function `batchmatching_inv` and G is extracted from 'disp0.pfm' by using the function `readpfm`, MSE is defined as:

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}\big(D(i,j) - G(i,j)\big)^2 \tag{3.2}$$

The PSNR (in dB) is defined as:

$$PSNR = 10 \times log_{10}\left(\frac{{MAX_I}^2}{MSE}\right) \tag{3.3}$$

Since we have already normalized D and G, the MAX_I is 255. The formula of normalization can be referred to (3.2).

4.2 Function Description

```
p = verify_dmap(D, G)
```

Input: normalized disparity map D and normalized ground truth G.

Output: PSNR (in dB).

5. GUI

The GUI consists of 8 text functions. Six of these stipulate the names of the edition functions. The names of the text functions are, “Preset”, “Left image”, “Right image”, “R”, “T”, “P”. Another two of these text functions are used to show the display of the matching time. These are shown in Figure 5.1.

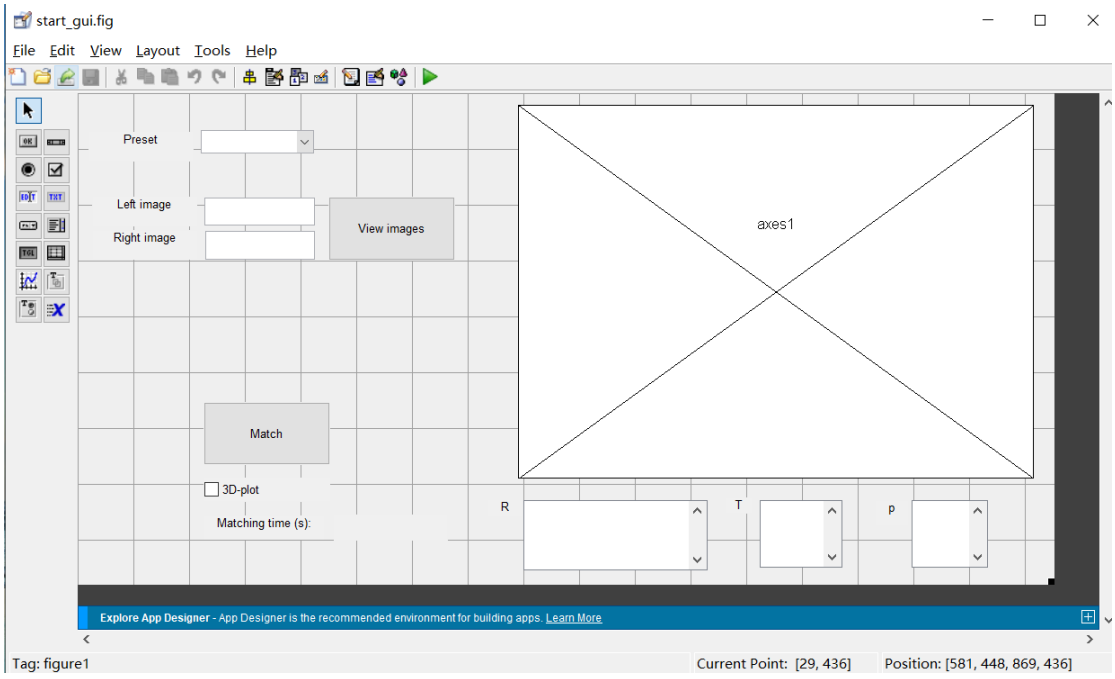


Figure 5.1 Interface of GUI

A popup menu function is used to show the folder of the images. When the drop-down button of the popup menu function is pressed, the user can select the folder by themselves. The callback function of popup menu is used to get the absolute path of the

selected image. The path will be sent to the callback function of push button “View images”.

Five edition functions are created. Two of them are used to show the path of the selected images. Another three edition functions are used to show the rotation matrix, translation vector and the value of psnr. The result is shown in Figure 5.2.

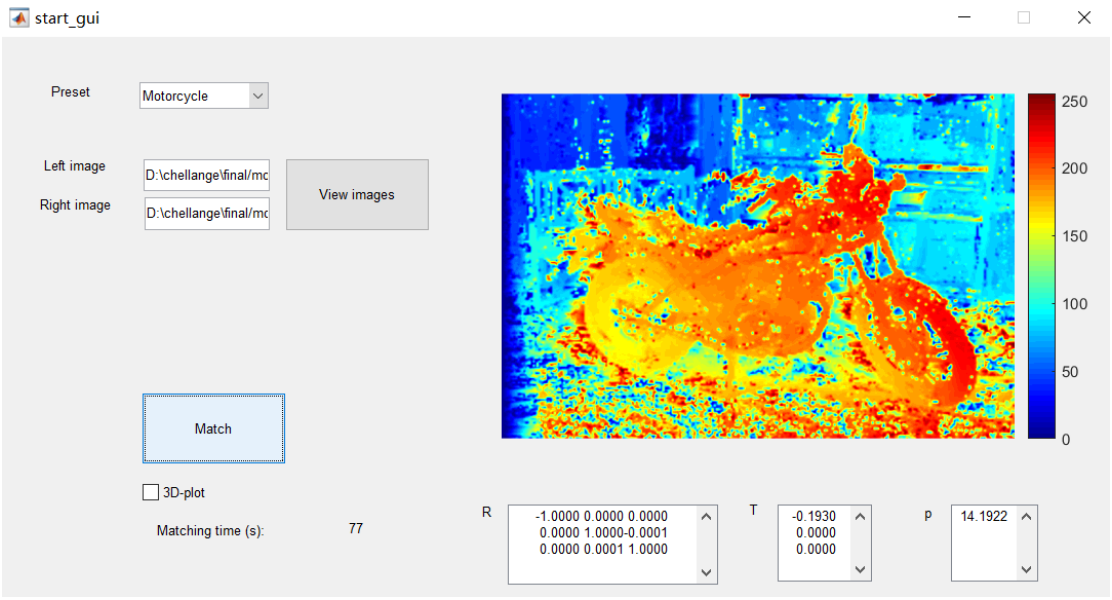


Figure 5.2 Results display

There are also two push button functions and a axes function. The callback function of push button is used to deal with the image. One button is called ‘Match’. When the button is pressed, the disparity map will be computed and at the same time the disparity map will be shown in the specified axes region. The value of ‘R’, ‘T’ and ‘P’ is also displayed in the corresponding edition functions. The result is also shown in Figure 5.2. Another button is called ‘View images’, whose callback function is used to load the selected images and show them. The result is shown in Figure 5.3.

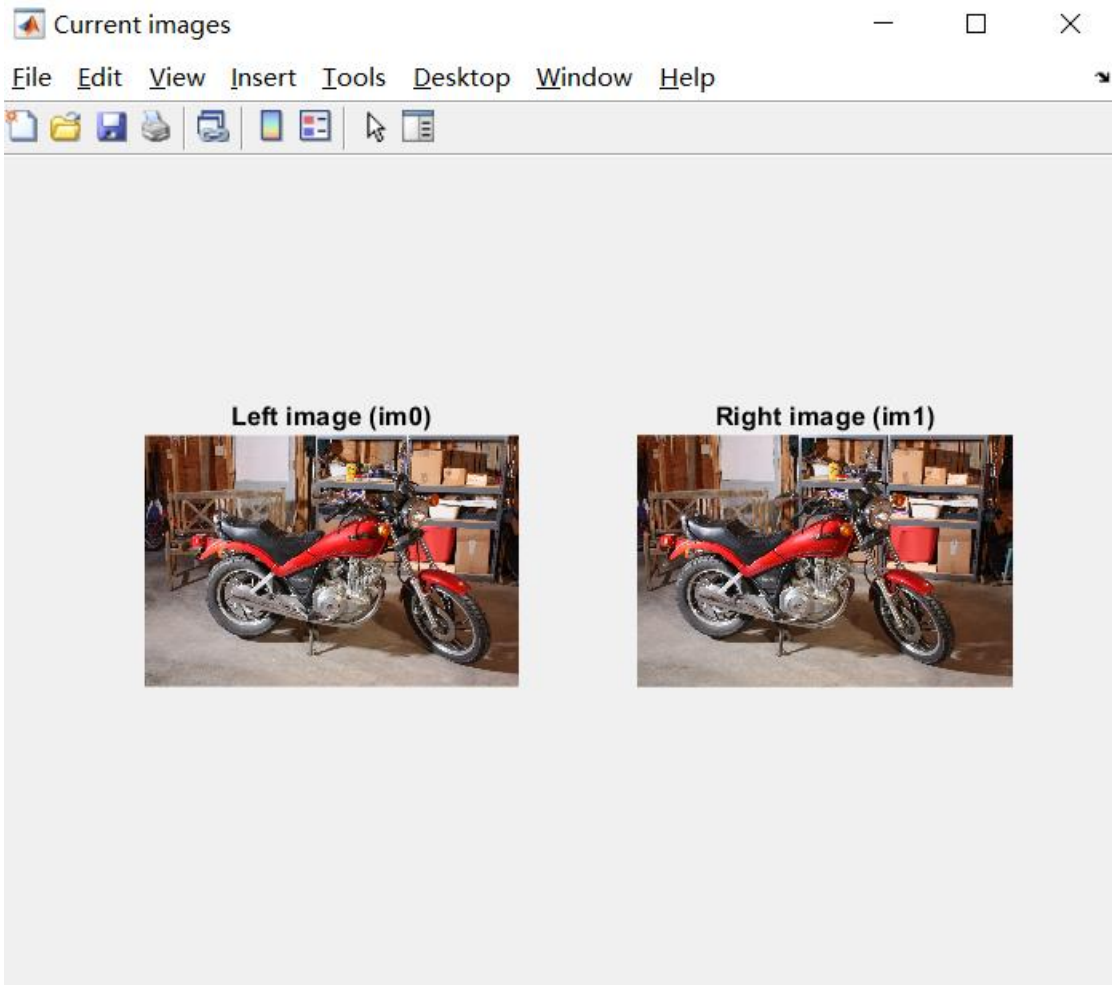


Figure 5.3 Image display

A check box function is used to combine 3D-Plot. If the term of 3D-plot is selected, in the callback function we will get the returned value 1. Then when the button 'Match' is pressed, 3D-reconstructed map will be also computed and displayed.

6. Unittest

The test.m file contains a test class inherited from matlab.unittest.TestCase. The test class consists of a property block and a methods block. And three independent tests are listed in the methods block:

- Check_toolboxes
- Check_variables

- Check_psnr

1) In `check_toolboxes` test, a list of toolboxes used in this file will be returned, after calling a build-in function named `matlab.codetools.requiredFilesAndProducts()`. By checking if the length of the returned list is greater than 1, we will know if any toolboxes are applied.

2) In `check_variables` test, methods from the Verifiable class, like `verifyNotEmpty()` and `verifyGreaterThan()` are applied to check if all required variables are not empty and fulfill the request.
3) In the end, `check_psnr` test is written to check if the PSNR calculated by homemade `verify_disparity()` equals to the one computed by the built-in `psnr()` function within a given tolerance. This can be realized by using constraints in conjunction with the qualification methods.

The tests integrated in the test class can be executed all together by running the following commands:

```
testCase = test;
res = run(testCase)
```

Or each test can be run individually by running

```
testCase = test;
res = run(testCase, 'check_psnr')
```

If everything fulfills the requirements, we will get the following result, which indicates that all three tests are passed.

```
Done test
__________

ans =

  1×3 TestResult array with properties:

    Name
    Passed
    Failed
    Incomplete
    Duration
    Details

Totals:
   3 Passed, 0 Failed, 0 Incomplete.
   10.1374 seconds testing time.
```

Otherwise, an error report in detail is delivered:

```
================================================================================
Verification failed in test/check_variables.
    ---------------------
    Framework Diagnostic:
    ---------------------
    verifyGreaterThan failed.
    --> Each element must be greater than the minimum value.

.
Done test
__________

Failure Summary:

     Name                  Failed  Incomplete  Reason(s)
    ================================================================
     test/check_variables    X                 Failed by verification.

ans =

  1×3 TestResult array with properties:

    Name
    Passed
    Failed
    Incomplete
    Duration
    Details

Totals:
   2 Passed, 1 Failed (rerun), 0 Incomplete.
   10.7814 seconds testing time.
```

7. 3D Plot

To complete the additional task 3D plot, we use the ‘Computer Vision System’ toolbox. The input of `plot_3D` function is the absolute path of two images. After a series of calculation, the function will plot the dense point cloud of images. The function can be described as followings:

1. Extract sparse set of corresponding feature points.
2. Compute the fundamental matrix and the motion of the camera.
   (1 and 2 step can be solved by the same way of eight point algorithm)
3. Match a dense set of points between the two images. Re-detect the point using ‘detectMinEigenFeature’ with a reduced ‘MinQuality’ to get more points. Then track the dense points into the second image. Estimate the 3-D locations corresponding to the matched points using the `triangulate` function. Place the origin at the optical center of the camera corresponding to the first image.
4. Determine the 3-D locations of the matched points using `triangulate`.
5. Plot the dense point cloud using `pointCloud`, after plotting we find that there are a few outliers which seriously influence the quality of the 3D image, so we use `pcdenoise`, so that some of the outliers are deleted.

The results of 3D plots are as followings:

Figure 7.1 Motorcycle

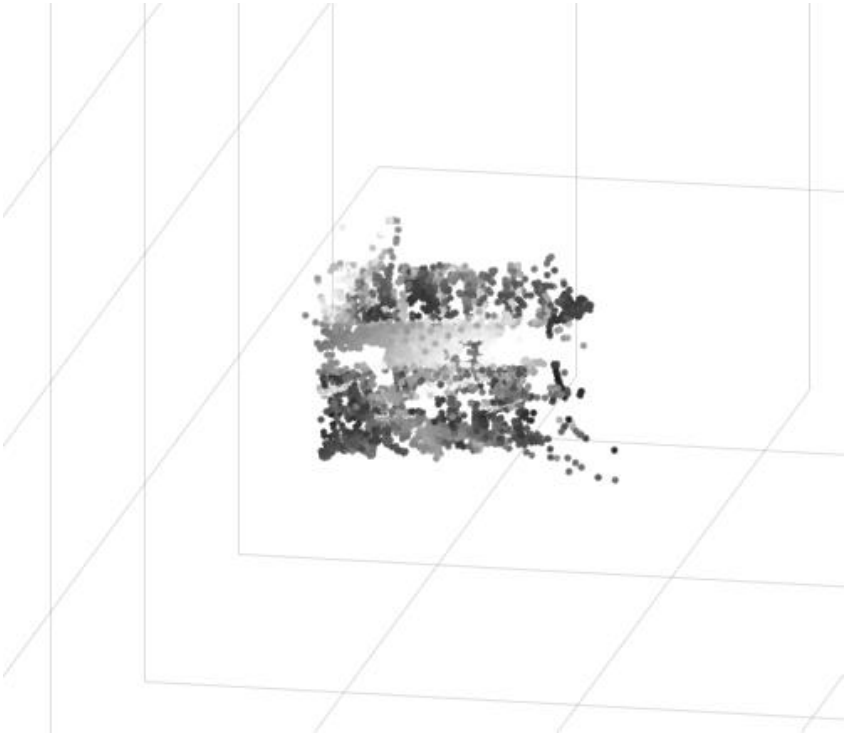


Figure 7.2 Terrace

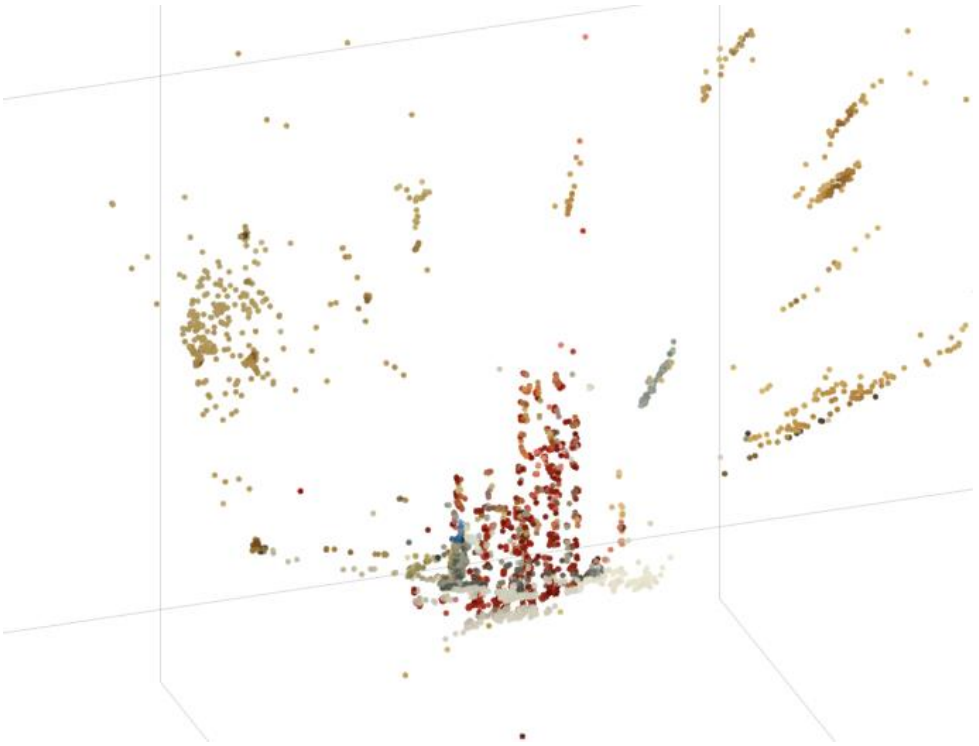


Figure 7.3 Sword

Figure 7.4 Playground

8. Plots and Result

By changing the relative path in ‘challenge.m’, we can all the variables which are requested in the project, which contains R, T, disparity map, psnr and elapsed time. All the results are shown as followings:

1. motorcycle:

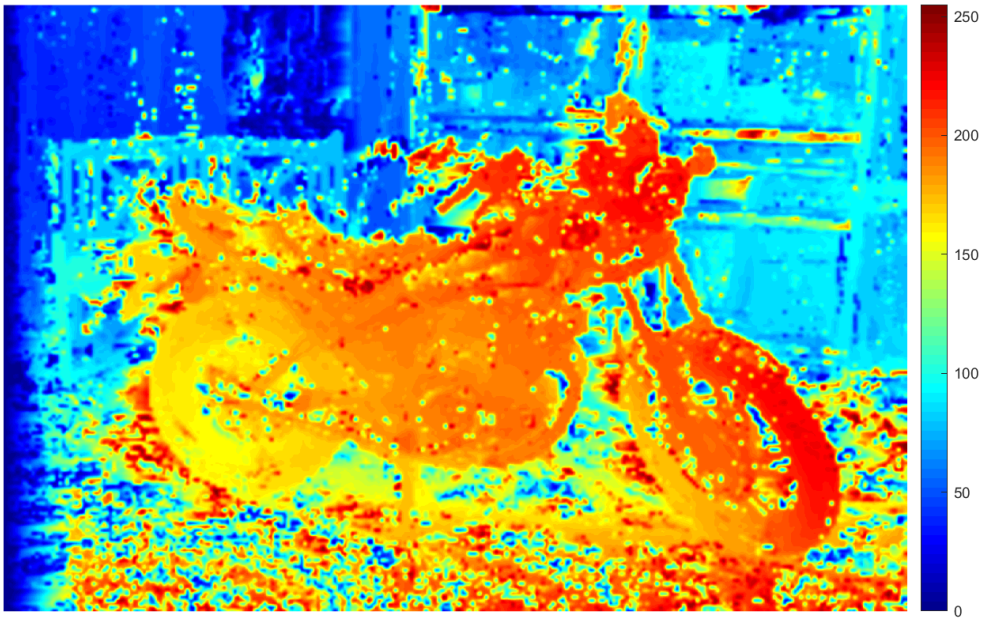


Figure 8.1 Motorcycle

```
scene_path =

    'motorcycle'

Rotation Matrix R is
    1.0000    0.0000    0.0000
    0.0000    1.0000    0.0001
    0.0000    0.0001    1.0000

Translation Vectro T(in meter) is
    0.1930
   -0.0000
   -0.0000

PSNR is 14.1922dB
Elapsed time is 81.489s
```

2. sword

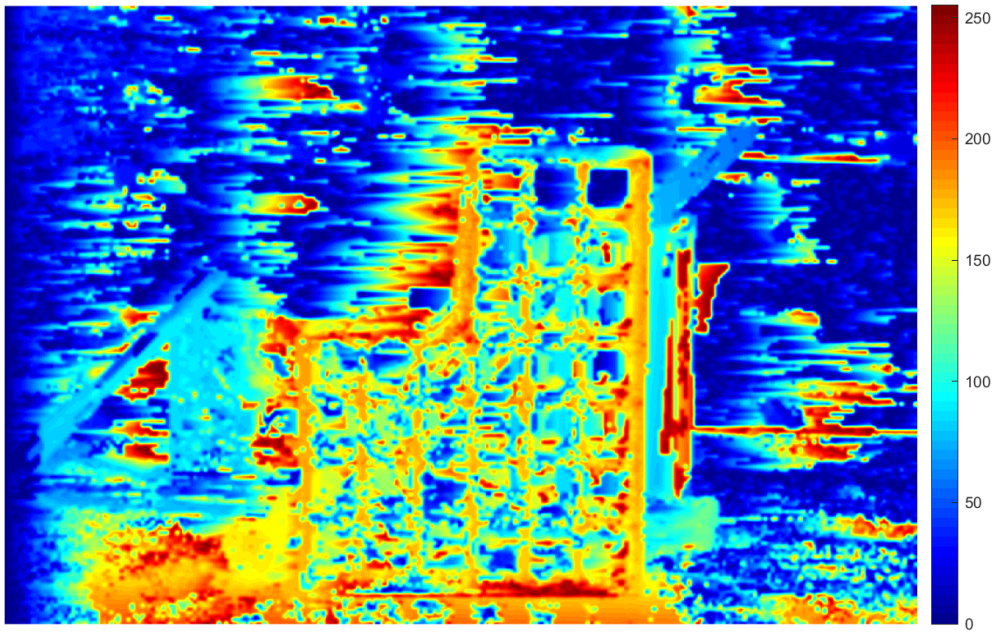


Figure 8.2 Sword

```
scene_path =

    'sword'

Rotation Matrix R is
    0.9785   -0.0798   -0.1900
    0.1012    0.9892    0.1058
    0.1795   -0.1227    0.9761

Translation Vectro T(in meter) is
    0.0495
   -0.0177
   -0.1614

PSNR is 11.4351dB
Elapsed time is 85.3601s
```

3. playground

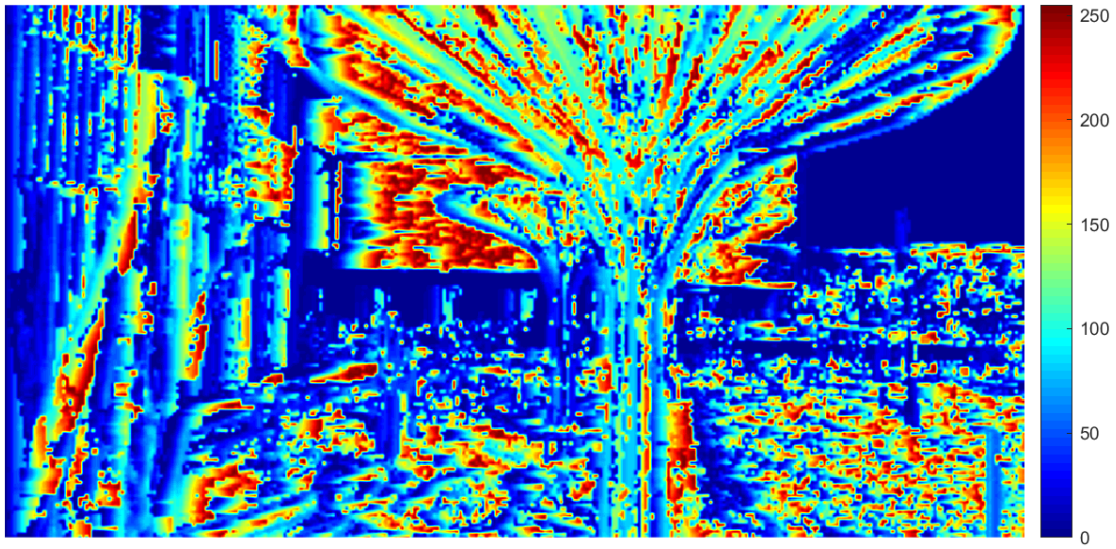


Figure 8.3 Playground

```
scene_path =

    'playground'

Rotation Matrix R is
    0.9999   -0.0122   -0.0022
    0.0122    0.9999   -0.0021
    0.0023    0.0021    1.0000

Translation Vectro T(in meter) is
    0.0140
   -0.0013
    0.0579

PSNR is 8.2494dB
Elapsed time is 6.4256s
```

4. terrace

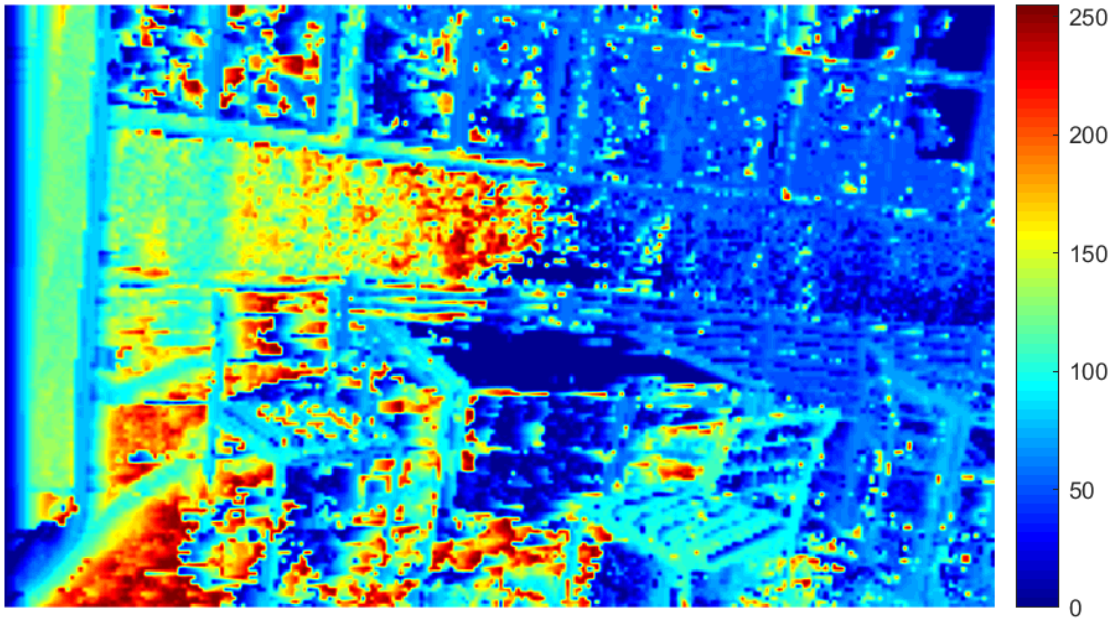


Figure 8.4 Terrace

```
scene_path =

    'terrace'

Rotation Matrix R is
    1.0000    0.0000    0.0000
   -0.0000    1.0000   -0.0000
   -0.0000    0.0000    1.0000

Translation Vectro T(in meter) is
    0.0599
   -0.0000
   -0.0000

PSNR is 9.2032dB
Elapsed time is 5.4362s
```

Reference

[1] D.Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. International Journal of Computer Vision, 47(1/2/3):7-42, April-June 2002.