

Exercise 3: Stereo Vision

In this exercise, we will implement a stereo vision pipeline. The algorithm is able to recover 3D structure from two calibrated (both intrinsics and extrinsics) cameras.

IMPLEMENTATION TIPS: Matlab uses a row major convention to access the elements of a matrix. For example, `M(1,2)` gives the element of the first row and the second column. However, in computer vision, a commonly used convention is a column major one, which fixes the origin of the image coordinates to the top-left corner of the image and the x -axis is along the image width (Fig.1). *We use the column major convention consistently in this exercise.* Therefore you may need to switch the coordinate components when you need to access an image pixel directly.

1 Exercise setup

We work with a simplified case: the two stereo cameras are calibrated and aligned. The exercise setup is shown in Fig. 1

The left camera is located at the origin and is taken as the reference (thus its rotation should be an identity matrix and translation zero). The right camera only has a translational motion along the x -axis with respect to the left camera and we call the length of this translation the *baseline* of the stereo pair. We use \mathbf{I}_0 and \mathbf{I}_1 to denote the images from the left and right cameras respectively.

This setup can be characterized by the projection matrices of the two cameras. A projection matrix M converts the homogeneous coordinates of 3D point \mathbf{P} to the homogeneous image coordinates \mathbf{p} and can be formulated as:

$$\begin{aligned} M &= KR[I| -C] \\ &= K[R|t] \end{aligned} \tag{1}$$

where $t = -RC$, R is the rotation matrix (identity in our exercise) and C is the coordinates of the projection center. K is the intrinsic matrix:

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

where f_x and f_y are the focal lengths; c_x and c_y are the image center coordinates.

EXERCISE 1: implement the function `projectionMatrix.m`. Read the comment at the beginning of the function to be clear about the interface.

Use the following command to load the parameters of the exercise setup:

```
load('playroom_calib');
```

The data in `playroom_calib.mat`¹ are:

- `f0`, `f1`: focal lengths of the left and right cameras
- `c0`, `c1`: image centers of the left and right cameras
- `img_size`: the size of the images, the same for both cameras
- `baseline`: the length of the stereo pair baseline

¹the data in this exercise is from Middlebury, a famous benchmark dataset for computer vision.

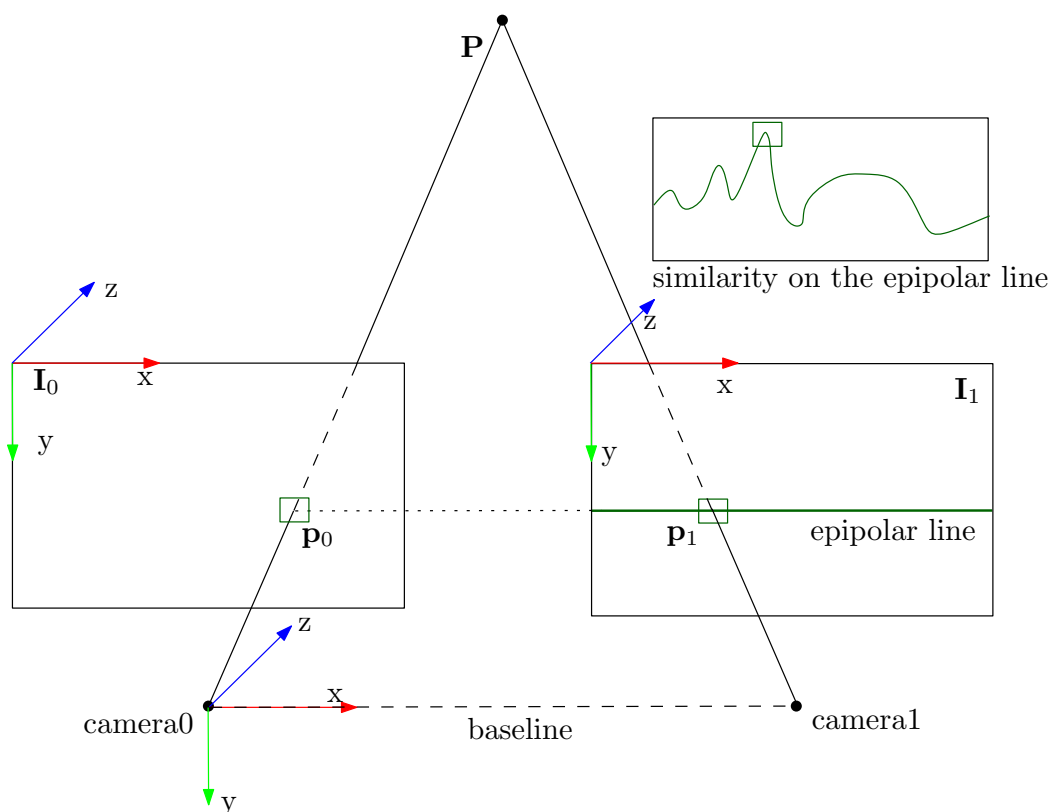


Figure 1: Exercise setup and epipolar search

- **D_range**: disparity range, used in *EXERCISE 5*

Then use the `projectionMatrix` function to calculate the projection matrices of the stereo cameras. This function will also be used in the following exercises.

2 Correspondence search

Before the 3D structure can be recovered, we need to establish correspondences between the images.

When the relative pose between two cameras is known, for a specific pixel \mathbf{p}_0 in \mathbf{I}_0 , the corresponding pixel \mathbf{p}_1 in \mathbf{I}_1 can only appear on the epipolar line. Furthermore, in the simplified case, where the images are aligned, the epipolar line should be a row in \mathbf{I}_1 .

For each pixel on the epipolar line in \mathbf{I}_1 , a similarity score can be calculated with respect to \mathbf{p}_0 . Commonly used similarity measurements are SSD (Sum of Squared Differences), SAD (Sum of Absolute Differences) and ZNCC (Zero-mean Normalized Cross Correlation)².

Then the pixel on epipolar line which is most similar to \mathbf{p}_0 is taken as the correspondence in \mathbf{I}_1 . The correspondence search process is illustrated in Fig.1.

EXERCISE 2: finish the function `searchEpipolar.m` and run `run_stereo_interactive.m`. The script will visualize the epipolar line and the correspondence for the pixel you click in the left image, along with the scores with different similarity measurements. First read the comment at the beginning of the `searchEpipolar.m` function carefully to be clear about the interface. You will need to do the following:

1. implement the function `rectifiedEpipolarLine.m` which returns the epipolar line in the right image.

²refer to the feature detection slide (page 12) for details

2. implement three similarity measurements: `SSD.m`, `SAD.m` and `ZNCC.m`.
3. find the best match on the epipolar line based on the similarity score (implement in `searchEpipolar.m`).

After each step, you can run `run_stereo_interactive.m` for visualization. The result after the above steps should be similar to Fig.2.

3 Triangulation

For two cameras whose intrinsics and extrinsics are already known, triangulation is the process of estimating the 3D point \mathbf{P} from a pair of correspondences \mathbf{p}_0 and \mathbf{p}_1 . The 3D point and image points are related by:

$$\begin{aligned}\lambda_0 \mathbf{p}_0 &= M_0 \mathbf{P} \\ \lambda_1 \mathbf{p}_1 &= M_1 \mathbf{P}\end{aligned}\tag{3}$$

Note that both \mathbf{p} and \mathbf{P} are the homogeneous coordinates and λ_0/λ_1 are scalars. Based on the definition of vector cross product, we have $\mathbf{p}_0 \times (M_0 \mathbf{P}) = 0$, $\mathbf{p}_1 \times (M_1 \mathbf{P}) = 0$. Also, the cross product of two vectors can be expressed using the skew-symmetric matrix, therefore the following equations hold:

$$\begin{aligned}[\mathbf{p}_0]_{\times} M_0 \cdot \mathbf{P} &= 0 \\ [\mathbf{p}_1]_{\times} M_1 \cdot \mathbf{P} &= 0\end{aligned}\tag{4}$$

where $[\cdot]_{\times}$ stands for the skew-symmetric matrix from a vector. Eq.4 gives an (overdetermined) equation system and can be solved for \mathbf{P} . For a detailed derivation, please refer to the slide from the lecture³.

EXERCISE 3: complete `triangulatePoint.m`. Part of the function is already given. Then uncomment the corresponding part in `run_stereo_interactive.m` and run the script to observe the triangulated point (Fig.2, bottom left window).

4 Reprojection

Once the 3D point \mathbf{P} is determined, we can project it into \mathbf{I}_0 and \mathbf{I}_1 and compare the projected pixels with the original one. The projection is done via Eq.3. Remember to normalize the last component of the homogeneous coordinates to 1 to get the image coordinates.

EXERCISE 4: implement `projection3d.m` and run `run_stereo_interactive.m` to observe the reprojection error (output in the Matlab console). Note that in the simplified case, which has a perfect calibration, the reprojected pixels should be exactly the same as the original ones and the reprojection errors should be very small (only numerical error).

5 Disparity map

One useful representation for dense 3D reconstruction is the disparity map. To construct a disparity map, we need to perform the aforementioned correspondence search for each pixel in \mathbf{I}_0 and calculate the disparity as:

$$d = \|\mathbf{p}_1 - \mathbf{p}_0\|\tag{5}$$

From the disparity, the depth z can be recovered:

$$z = \frac{f_x b}{d}\tag{6}$$

where b denotes the baseline length. Calculating the dense disparity map is computationally expensive since the correspondence search is done for every pixel. Often we can limit the range of

³multiple view geometry-1, page 64-67

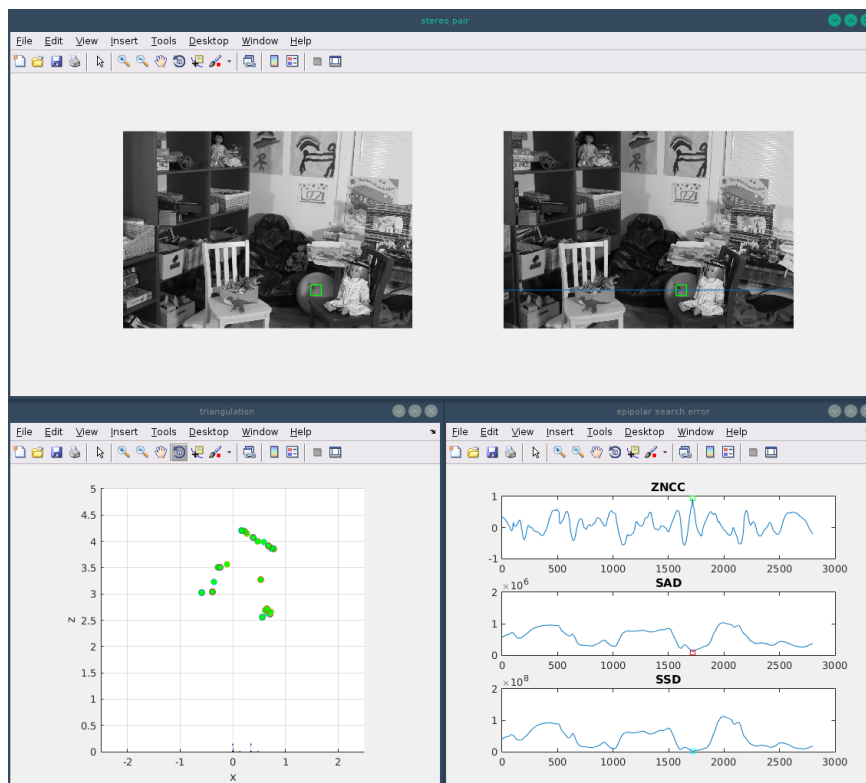


Figure 2: Exercise Snapshot

search to one part of the epipolar line based on the knowledge about the structure. As can be seen from Eq.6, if we know the depth range of the scene (i.e., the range of z), the range of disparity can also be determined. Then only the pixels that fall in that range need to be checked. In the following exercise, we use a downsampled dataset (`playroom_s_calib.mat`, `playroom_s0.png` and `playroom_s1.png`) to reduce the running time.

EXERCISE5: First add an extra argument to `rectifiedEpipolarLine.m` to make the function work with limited disparity range. Note that `searchEpipolar.m` also needs to be changed accordingly, since it calls `rectifiedEpipolarLine.m`. Then finish `run_dense_disparity.m` to generate the disparity map and calculate the corresponding depth map. The disparity map should look like Fig.3.

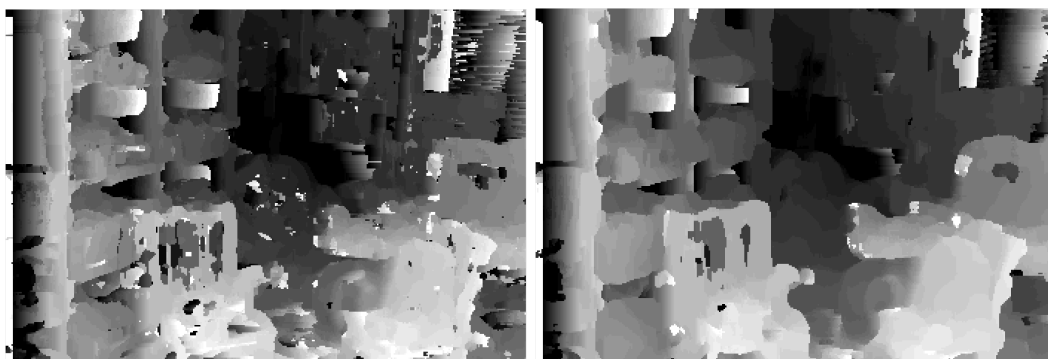


Figure 3: Disparity map using window size of 7 and 13 pixel respectively.