Report

|  |  |  |
| --- | --- | --- |
| 姓名 | 学号 | 贡献 |
| 林木得 | 11354010 | 33% |
| 康乐 | 11332008 | 33% |
| 李梦琦 | 11331172 | 34% |

Project: Stereo Calibration and Scene Reconstruction

Implement the system of stereo calibration and scene reconstruction to reconstruct a 3-D scene from a stereo pair of images.

Description:

(Overview) Stereo vision is the process of recovering depth from camera images by comparing two or more views of the same scene. The output of this computation is a 3-D point cloud, where each 3-D point corresponds to a pixel in one of the images. Binocular stereo uses only two images, taken with cameras that were separated by a horizontal distance known as the "baseline. Calibrating the stereo camera system allows us to compute the 3-D world points in actual units, such as millimeters, relative to the cameras.

This example shows how to calibrate a camera system consisting of two web-cams. It then uses the parameters estimated by calibration to rectify a stereo pair of images and reconstruct the corresponding 3-D scene. Applications of this process include measurement, aerial photogrammetry, and robot navigation.

The process consists of the following steps:

* Calibrate the stereo camera system
* Rectify a pair of stereo images
* Compute disparity
* Reconstruct the 3-D point cloud

Matlab Implementation

Tool: MATLAB 2014a

Procedures:

1. Specify Calibration Images

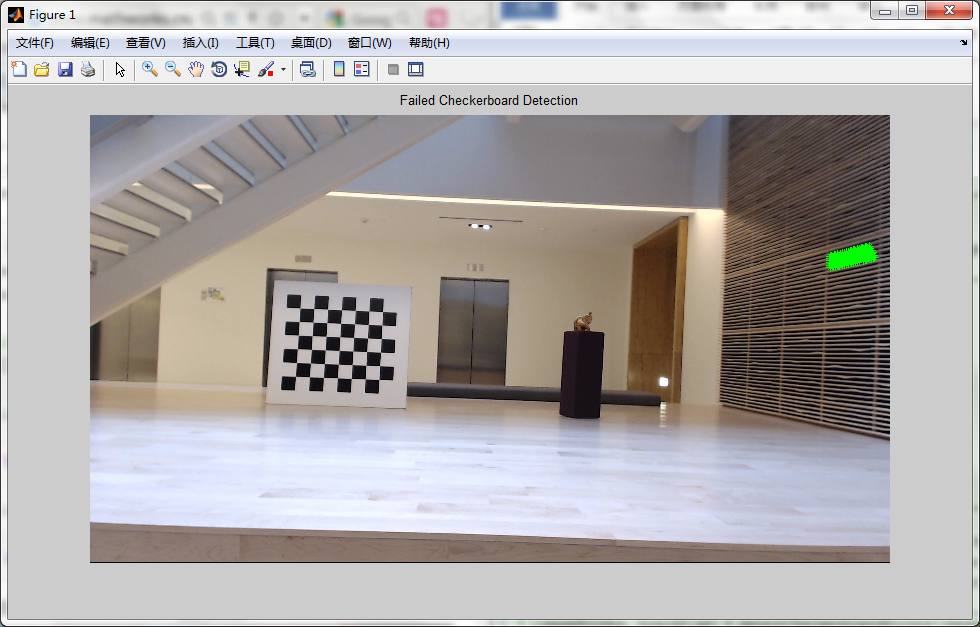
Create two cell arrays containing file names of calibration images taken with each camera.

|  |
| --- |
| numImagePairs = 10;  imageFiles1 = cell(numImagePairs, 1);  imageFiles2 = cell(numImagePairs, 1);  imageDir = fullfile(matlabroot, 'toolbox', 'vision', 'visiondemos', ...  'calibration', 'stereoWebcams');  for i = 1:numImagePairs  imageFiles1{i} = fullfile(imageDir, sprintf('left%02d.png', i));  imageFiles2{i} = fullfile(imageDir, sprintf('right%02d.png', i));  end |

2. Detect Checkerboards

Detect the checkerboard in the images. Unfortunately, in these images the texture of the wall on the right causes the checkerboard detection to fail.

|  |
| --- |
| % Try to detect the checkerboard  im = imread(imageFiles1{1});  imagePoints = detectCheckerboardPoints(im);    % Display the image with the incorrectly detected checkerboard  figure; imshow(im, 'InitialMagnification', 50);  hold on;  plot(imagePoints(:, 1), imagePoints(:, 2), '\*-g');  title('Failed Checkerboard Detection'); |



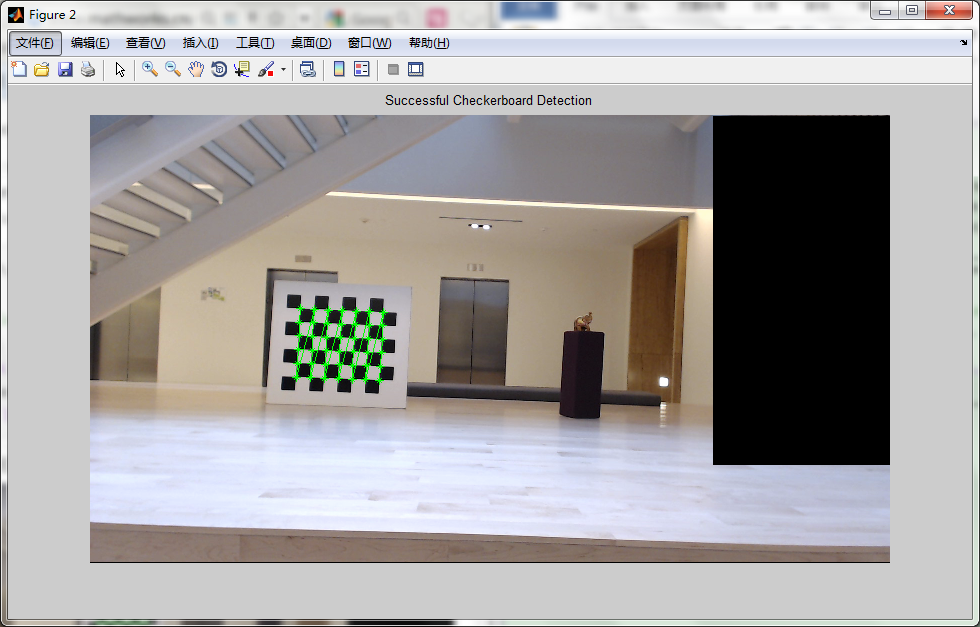
3. Mask out the wall on the right. Determine the coordinates of the region containing the problematic texture using imtool or imrect.

|  |
| --- |
| images1 = cast([], 'uint8');  images2 = cast([], 'uint8');  for i = 1:numel(imageFiles1)  im = imread(imageFiles1{i});  im(3:700, 1247:end, :) = 0;  images1(:, :, :, i) = im;    im = imread(imageFiles2{i});  im(1:700, 1198:end, :) = 0;  images2(:, :, :, i) = im;  end |

4. Detect Checkerboards in the Modified Images

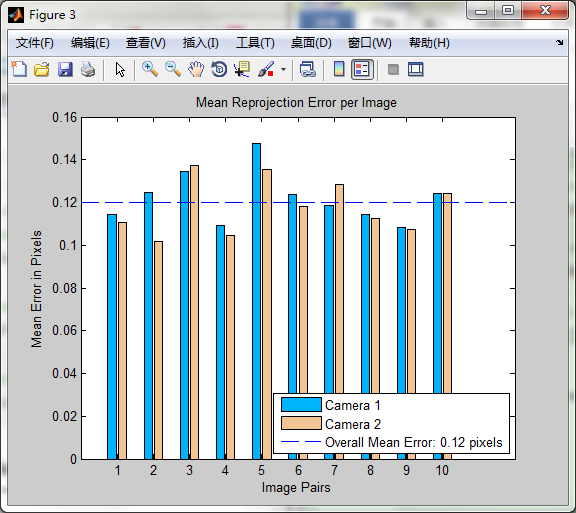
Detect the checkerboard in all stereo pairs of images. Pass in the images as two sets, one from each camera. The images in both sets must be in the same order to form stereo pairs. The detectCheckerboardPoints function returns imagePoints, a 4-D array, such that imagePoints(:,:,:,1) are points from camera 1, and imagePoints(:,:,:,2) are points from camera 2.

|  |
| --- |
| [imagePoints, boardSize] = detectCheckerboardPoints(images1, images2);    % Display one masked image with the correctly detected checkerboard  figure; imshow(images1(:,:,:,1), 'InitialMagnification', 50);  hold on;  plot(imagePoints(:, 1, 1, 1), imagePoints(:, 2, 1, 1), '\*-g');  title('Successful Checkerboard Detection'); |



5. Calibrate the Stereo Camera System

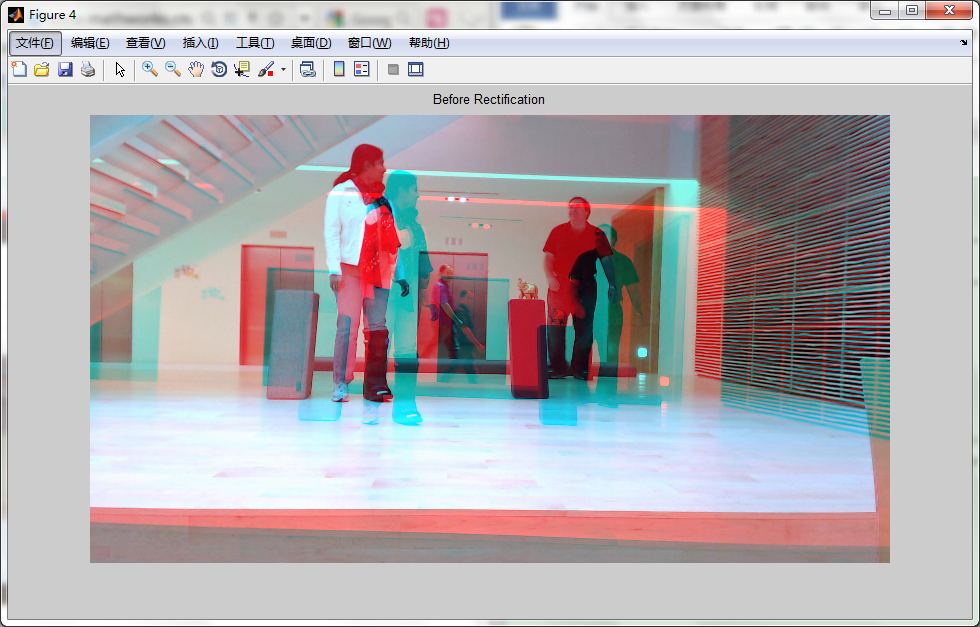
|  |
| --- |
| % Generate world coordinates of the checkerboard points.  squareSize = 108; % millimeters  worldPoints = generateCheckerboardPoints(boardSize, squareSize);    % Compute the stereo camera parameters.  stereoParams = estimateCameraParameters(imagePoints, worldPoints);    % Evaluate calibration accuracy.  figure; showReprojectionErrors(stereoParams); |

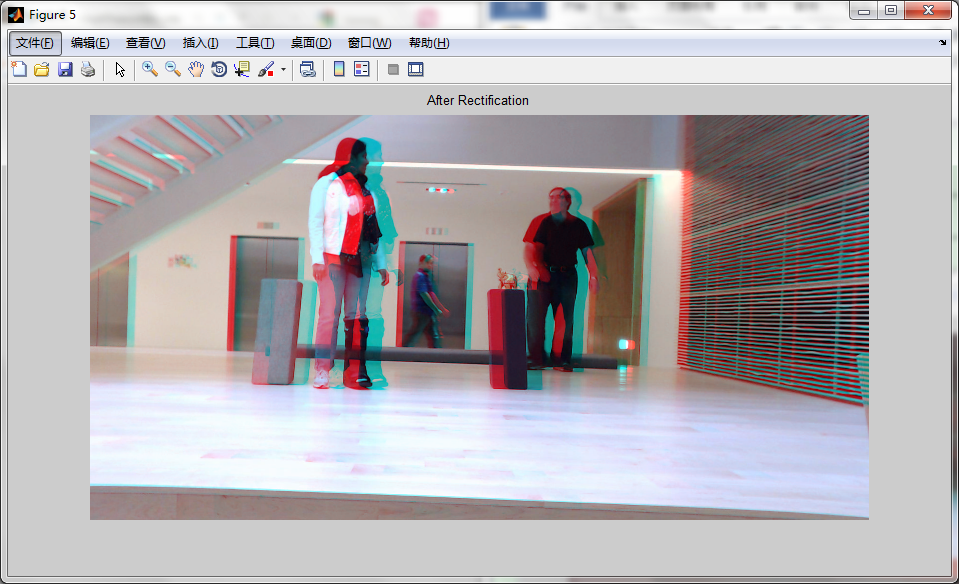


6. Rectify a Stereo Pair of Images Taken with the Calibrated Cameras

Rectification is an important pre-processing step for computing disparity, because it reduces the 2-D stereo correspondence problem to a 1-D problem. Rectified images appear as though the two image planes are parallel and row-aligned. This means that for each point in image 1, its corresponding point in image 2 lies along the same row.

|  |
| --- |
| % Read in the stereo pair of images.  I1 = imread('sceneReconstructionLeft.jpg');  I2 = imread('sceneReconstructionRight.jpg');    % Rectify the images.  [J1, J2] = rectifyStereoImages(I1, I2, stereoParams);    % Display the images before rectification.  figure; imshow(cat(3, I1(:,:,1), I2(:,:,2:3)), 'InitialMagnification', 50);  title('Before Rectification');    % Display the images after rectification.  figure; imshow(cat(3, J1(:,:,1), J2(:,:,2:3)), 'InitialMagnification', 50);  title('After Rectification'); |

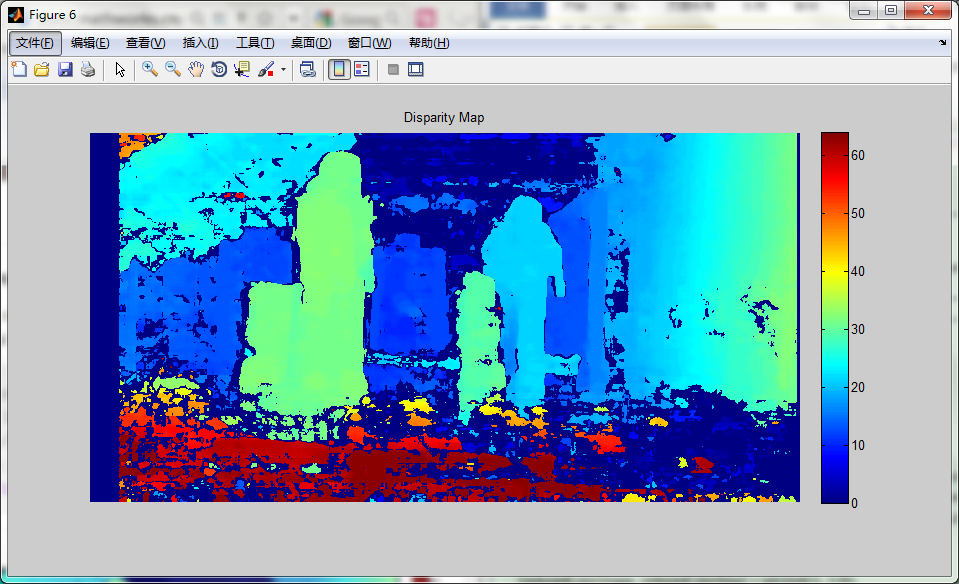




7. Compute Disparity for 3-D Reconstruction

The distance in pixels between corresponding points in the rectified images is called disparity. The disparity is used for 3-D reconstruction, because it is proportional to the distance between the cameras and the 3-D world point.

|  |
| --- |
| disparityMap = disparity(rgb2gray(J1), rgb2gray(J2));  figure; imshow(disparityMap, [0, 64], 'InitialMagnification', 50);  colormap('jet');  colorbar;  title('Disparity Map'); |



8. Reconstruct the 3-D Scene

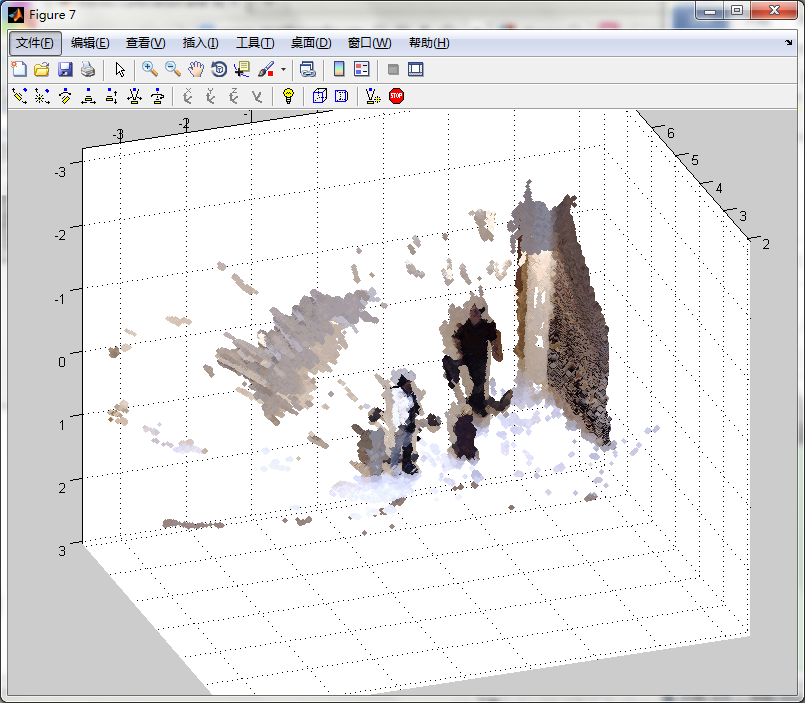
Reconstruct the 3-D world coordinates of points corresponding to each pixel from the disparity map.

|  |
| --- |
| pointCloud = reconstructScene(disparityMap, stereoParams);    % Convert from millimeters to meters.  pointCloud = pointCloud / 1000; |

9. Visualize the 3-D Scene

Plot the 3-D points by calling the plot3 function. We have to call the function in a loop for each color, because it can only plot points of a single color at a time. To minimize the number of iteration of the loop, reduce the number of colors in the image by calling rgb2ind.

|  |
| --- |
| % Reduce the number of colors in the image to 128.  [reducedColorImage, reducedColorMap] = rgb2ind(J1, 128);    % Plot the 3D points of each color.  hFig = figure; hold on;  set(hFig, 'Position', [1 1 840 630]);  hAxes = gca;    X = pointCloud(:, :, 1);  Y = pointCloud(:, :, 2);  Z = pointCloud(:, :, 3);    for i = 1:size(reducedColorMap, 1)  % Find the pixels of the current color.  x = X(reducedColorImage == i-1);  y = Y(reducedColorImage == i-1);  z = Z(reducedColorImage == i-1);    if isempty(x)  continue;  end    % Eliminate invalid values.  idx = isfinite(x);  x = x(idx);  y = y(idx);  z = z(idx);    % Plot points between 3 and 7 meters away from the camera.  maxZ = 7;  minZ = 3;  x = x(z > minZ & z < maxZ);  y = y(z > minZ & z < maxZ);  z = z(z > minZ & z < maxZ);    plot3(hAxes, x, y, z, '.', 'MarkerEdgeColor', reducedColorMap(i, :));  hold on;  end    % Set up the view.  grid on;  cameratoolbar show;  axis vis3d  axis equal;  set(hAxes,'YDir','reverse', 'ZDir', 'reverse');  camorbit(-20, 25, 'camera', [0 1 0]); |



OpenCV Implementation

Tool: openCV 2.4

**问题分析：**

我们的目标是要从照片中重新构建场景，根据现有的理论，我们还需要知道的是相机的内参数矩阵，openCV提供了测内参数矩阵的函数，而测内参数矩阵，为了实现比较好的效果，还需要利用像棋盘，这种容易识别的物体辅助测绘，openCV提供了方法帮助寻找照片的棋盘角点，知道棋盘角点，就可以测出相机内参数矩阵。

知道相机内参数矩阵之后，进一步，我们可以得到相机外参数矩阵。这时候，利用双目视觉，我们可以利用两张图片的视差，重建出3D场景。

**算法理解和分析：**

在测相机内参数矩阵中，相机内参数矩阵M,外参数矩阵W,三维坐标Q ,图像上的点q，根据 q = sMWQ其中s为比例缩放因子，Q投影成q，q,Q 均为齐次坐标。考虑到一般性，我们假定Z = 0,这样Q = [X Y 0 1]T 。而根据单应性，可以将源平面（Z = 0的平面）和图像成像平面关联起来，Pdst = H Psrc.，根据多张图像的多个点对，就可以解出单应性矩阵H.在根据H 和M,W的对应关系，我们可以得到相机内参数矩阵。

这一步骤需要注意地方是，所拍照片不能有比棋盘上的点更稳定的点，否则检测到的棋盘角点会不准确。然后拍照过程相机不能对焦，需有相同的内常数。

在三维场景重建中，根据双目视觉，我们需要有2张（左，右）图片的的视差，根据重投影矩阵T,和视差d以及2d点p,就可以计算出三维点Q，继而可以重建3纬场景。

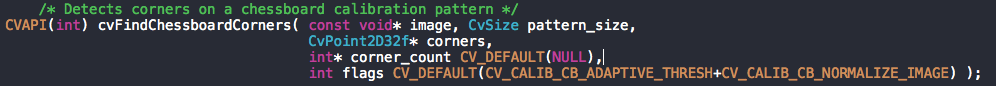
这个步骤需要注意的是，计算视差之前，需要矫正图像，使得2张左，右图像行对齐。

**实验结果及其分析：**

1. **相机标定**
2. 读入图片

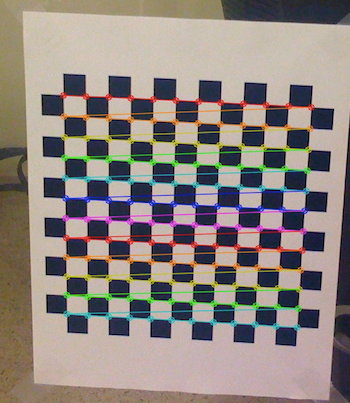
openCV，提供2种方式读入图片，一种是从照相机镜头直接获取照片，另外一种是从文件中读入照片。我们这里选用的是从文件中读入照片，也就是含有棋盘角点的图片是先拍好的，通过函数IplImage\* cvLoadImage(const char \*imagePath).

1. 检测角点

openCV中提供检测图像棋盘角点的函数是

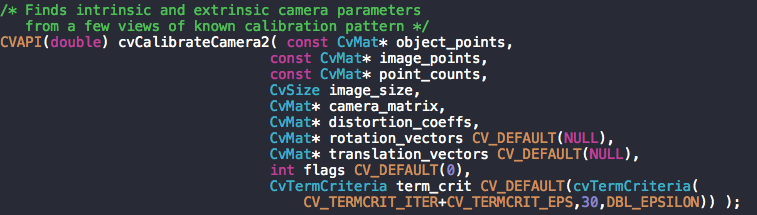
其检测到的角点坐标将会保存在表量，corners，这个向量使用前，必须先分配好空间，并且其大小必须大于等于角点个数。Image必须为灰度图。

函数运行结果，角点如下：

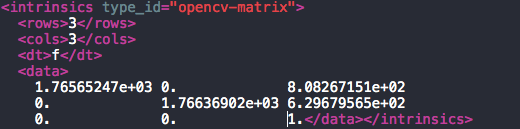


1. 相机标定

openCV提供标定相机的函数是：

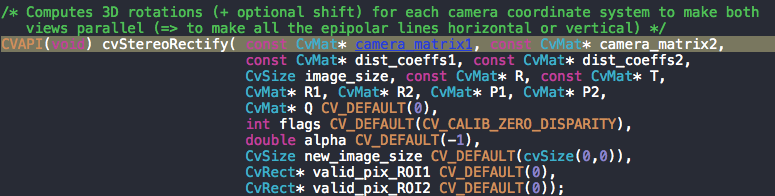


其中object\_points,是物体的这里用的是相对坐标，即第一个点（左上）坐标为(0，0，0)，image\_points为角点坐标即上个步骤检测到的角点坐标。相机内参数矩阵保存在camera\_matrix变量中，结果为：



1. **场景重建**
2. 矫正图片

矫正图片，让左右图片行对齐，openCV提供的函数是：



效果如下：



1. 计算视差

openCV中提供的函数是：



视差将被保存到disparity矩阵中

视差效果如下：



1. 3纬重建

根据视差，以及重投影矩阵以及视差以及2d点，可以重建3纬场景。

## Conclusion & Experience

Well, the most impressive thing I can remember is that the tool MATLAB used in the project took up almost the whole disk of my computer. So terrifying!

This project is really helpful for our understanding of the course. We’ve listened to our teacher’s lectures for a whole semester talking about series of amazing and wonderful achievements and theories in the field of Computer Vision and Pattern Recognition. We are attracted by the fantastic technologies applying those theories and immersed in the brilliant knowledge. It is really a great opportunity for us to further explore the field by applying theory into practice.

By doing the project on our own, we gradually comprehend the knowledge in a deeply way. Besides, we discover many interesting new knowledge in the process, which Is very meaningful.

We get to know the procedure of reconstructing a scene from 2 images, which also deepens our understanding of other technologies related to scene recognition. It’s really useful.