PROJECT 2A: CAMERA MODEL AND STEREO DEPTH SENSING

Tanmay Khandait
Student ID : 1219385830
CIDSE, Arizona State University
tkhandai@asu.edu

28 February, 2021

ABSTRACT

This project is a programming project which focuses on camera models and multiple-view models. This report is divided into four parts, where every part talks about the four task as per the project document. The camera being used is an ELP synchronized stereo camera which has a baseline length of 62 mm (around 2.44 inches). All the images provided by the instructor are of dimesnion $640 * 480$. For the chessboard images, a $9 * 6$ chessboard is being used, where the length of each grid is 25.4 mm (1 inch).

1 Pinhole camera model and calibration

In this task, the idea is to estimate the projection parameters (intrinsic and extrinsic parameters) of a camera, given the 3D to 2D correspondence of multiple images.

1.1 Theory

As discussed in class, we discussed the geometry of how pinhole-camera works. Assume that we have the camera centre at C with focal length f. The object is placed X units away and the image is formed at the image plane. The corresponding dimension in the image plane will be estimated by $x = f * \frac{X}{Z}$ (look at 1).

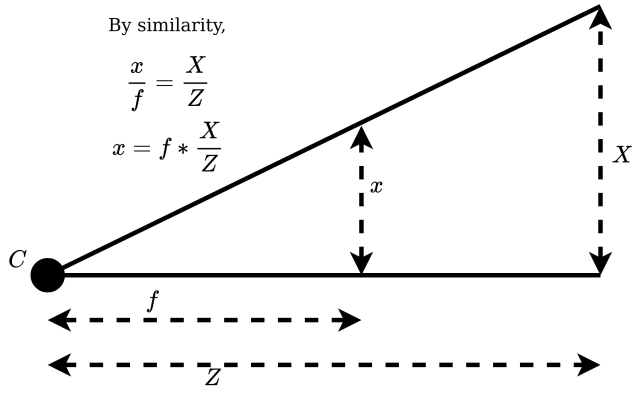


Figure 1: Pinhole camera geometry

We extend the similar idea to the higher dimension (3D) and we can get a set of corresponding equations. Assuming that X, Y, Z are the coordinates of the object in 3D and x, y are the coordinates of the object in the image plane, the mapping is from $[X, Y, Z] \rightarrow [f * \frac{X}{Z}, f * \frac{Y}{Z}]$. If both the 2D and 3D vectors are represented in homogeneous vectors,

we get

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} * \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Ideally, the camera centre is assumed to be $(0, 0)$, however, that is not usually the case. There is certain offset in the camera centres by u_0, v_0 in the camera centre. Also, images are basically a grid of pixels. Usually, these pixels are uniform and the the effect of these pixels can be incorporated by dividing the focal length in x direction by pixel length m_x and focal length in y direction by pixel height (m_y). In some cases, there are also exists some skewness amongst the basis axis of the image plane. We will denote them by s_θ. Incorporating all these, we can rewrite the equation as

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f.s_x & s_\theta & u_0 & 0 \\ 0 & f.s_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} * \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

We call the matrix as camera intrinsic matrix. If

$$K = \begin{bmatrix} f.s_x & s_\theta & u_0 & 0 \\ 0 & f.s_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = [K|0] \times \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{1}$$

where $[K|0]$is a zero vector appended to the K matrix.

Ideally, the position of the object in the world frame of reference and camera frame of reference is not the same. The 3-tuple (X, Y, Z) are the points of the object in the real world and we need to rotate and translate them to the camera frame of reference. Let us assume that the real world are represented by $\mathbf{X_w}$ and in camera frame of reference by $\mathbf{X_c}$. If C is the position of camera with respect to real world and is rotated by R with respect to real world, we can easily notice that

$$\begin{aligned} \mathbf{X_c} &= R * (\mathbf{X_w} - C) \\ \mathbf{X_c} &= R * \mathbf{X_w} - R * C \\ \mathbf{X_c} &= R[I| - C]\mathbf{X_w} \quad \text{where} R[I| - C] = \text{is } 3 \times 4 \text{ matrix} \end{aligned} \tag{2}$$

Notice that, we can combine 1 and 2 in this way

$$\begin{aligned} \mathbf{x} &= KR[I| - C]\mathbf{X} \\ \mathbf{x} &= P\mathbf{X} \quad \text{where } P = KR[I| - C] \end{aligned} \tag{3}$$

where **x** and **X** are the homogeneous vector representation of the object in image plane and real world respectively.

In this task, we are given the 3D and 2D correspondences (**X** and **x** respectively). The idea is to to find the matrix P (3) suing some sort of optimization problem and then decomposing this P to get the camera intrinsic and camera extrinsic matrices. To solve this optimization problem, the idea is to somehow rearrange the equations in the form of $Ax = 0$ where A is combination of elements of the 2D and 3D points, and x is a linear vector of elements of P. This can be easily done by assuming by assuming P to be a 3×4 matrix and opening and homogenizing 3. Rearranging these equation by bringing them on the same side and separating the elements of P will give rise to a matrix A of size 2×12 and vector x of size 12×1. Solving these set gives us the values of P and then these can be decomposed (for ex by RQ decomposition) to get the intrinsic and extrinsic matrix [1, 2].

Lenses in real life are often not perfect introduce distortions in the image. Radial distortion and Tangential distortion are some of them. Radial distortion occurs sue to the irregularity in the size of the lenses. The light at the edges of the lenses start bending more that the centre of the lens. Pincushion (negative radial) distortion and Barrel (positive radial) distortion are some of distortions caused due to this effect. Tangential distortion occurs when the axis of the lens and the principal axis of the image plane is misaligned.[1] [1].

[1]Content taken from here: https://www.mathworks.com/help/vision/ug/camera-calibration.html

1.2 Experiment

The images are read, and the corners are found by running cv2.findChessboardCorners(). It is noted that dimension used here are in inches. Each chessboard grid has a side of length 1 inch as described in the abstract. We now have the 3D as well as 2D points of the chessboard. These are used to find the intrinsic, extrinsic and distortion parameters of the camera which are used to calibrate the images (using the cv2.calibrateCamera()). The key idea is here is that once the calibration is done, straight lines straight (and not curved). The intrinsic and distortion matrices for the left and right camera (rounded off to 3 decimal places) are

$$K_{left} = \begin{bmatrix} 423.853 & 0. & 341.105 \\ 0. & 421.838 & 269.598 \\ 0. & 0. & 1. \end{bmatrix}$$

$$UD_{left} = [-0.436 \quad 0.273, -0. \quad 0.001, -0.114]$$

$$K_{right} = \begin{bmatrix} 421.021 & 0. & 352.165 \\ 0. & 418.858 & 264.557 \\ 0. & 0. & 1. \end{bmatrix}$$

$$UD_{right} = [-0.415 \quad 0.2 \quad -0. \quad -0.001 \quad -0.051]$$

The camera parameters have been stored in left_camera_intrinsic.xml and right_camera_intrinsic.xml. Here are the results from the experiment.

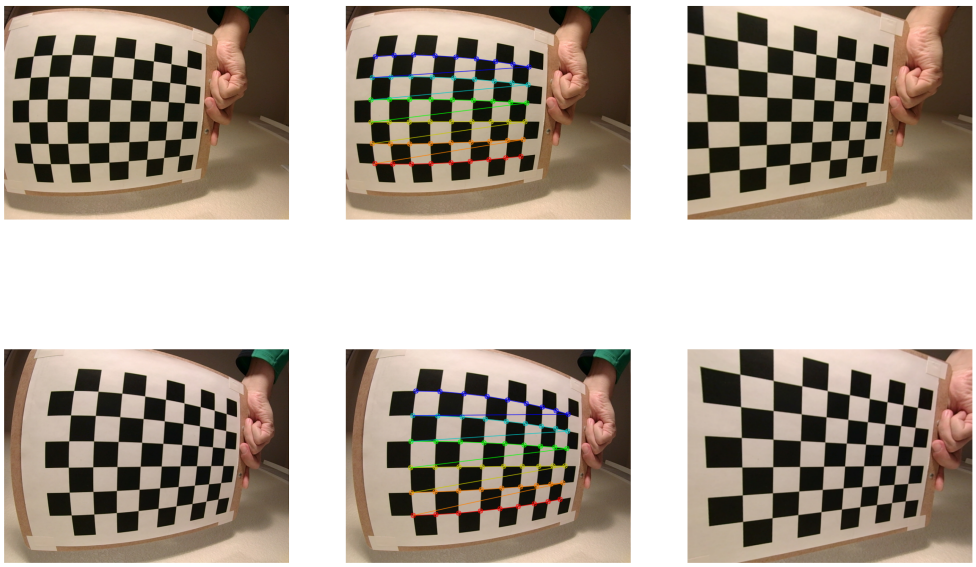


Figure 2: Images used are left_2 and right_2 to show calibrated images. Top row from (left to right): Original left_2.png image (first column). Corners detected on the image (second column). Calibrated and undistorted image (column 3). Bottom row from (left to right): Original right_2.png image (first column). Corners detected on the image (second column). Calibrated and undistorted image (column 3)

2 Stereo Calibration and Rectification

In this task, we have an two images of the same object photographed from left and right camera. Given these cameras, there will be a rotation and translation between these two images, however, these are usually not known. If given a pair of 3D to 2D correspondences for the objects, we can estimate these parameters. Using the calibration from the earlier task, we can recover the transformation of one camera with respect to the other. Ideally, we expect there to be just translation between the images obtained from the two cameras. However, due to errors in the camera rigs, there are cases when the principal axis of the cameras are not aligned, which leads to some transformation (including rotation and translation). The idea is to align their axis so that there is only translation between the images. This process is called stereo rectification. Since the underlying theory for stereo rectification is essentially the same as earlier, I will delve into the theory behind this. I consulted [2, 1, 3] for this task.

2.1 Experiments

The camera parameters for both the left and the right camwra are read from the file left_camera_intrinsic.xml and right_camera_intrinsic.xml. The next step is to go ahead with the calibration process. The function cv2.stereoCalibrate() is used to get the the rotation and the translation of the left camera with resepct to right camera. The correctness of the R and T vectors is checked by looking at the sign of the translation vector. Since left camera will be on the left of the right camera, we expect a positive translation vector, which is what we get. The R and T matrices have been rounded off to three decimal places.

$$R = \begin{bmatrix} 1. & 0.007 & 0.014 \\ -0.007 & 1. & -0.006 \\ -0.014 & 0.006 & 1. \end{bmatrix}$$

$$T = \begin{bmatrix} 2.465 \\ 0.014 \\ 0.011 \end{bmatrix}$$

Now that we know the transformation between the two camera, we move ahead to the stereo rectification process. The idea is to start off with undistorting the images and going ahead with rectification process whihc transforms the images so that their principal axis are aligned. This is being done via cv2.stereoRectify() function. The original images, undistorted and not rectified, and undistorted and rectified images for three different combinations (left+right, left+left, right+right) are shown in 3, 4, 5. To ensure the correctness of rectification, I have plotted the rectified images along with straight lines to ensure that corresponding images locations are matching or not. The images have been correctly rectified as one can see in 6. The 3D position of the image points were obtained using cv2.triangulatePoints(). These points were used to triangulate the position of the detected corneres o chessboard in 3D space. The two camera poses before and after rectification are shown in 7, 8, 9.

Figure 3: Two views (using left_0.png and right_0.png): Original (top). Undistorted but not rectified (middle). Undistorted and Rectified (bottom)

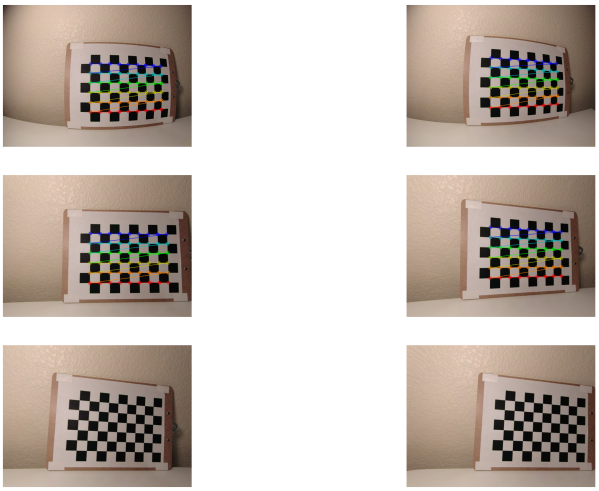


Figure 4: Two views (using left_0.png and left_1.png): Original (top). Undistorted but not rectified (middle). Undistorted and Rectified (bottom)

Figure 5: Two views (using right_0.png and right_1.png): Original (top). Undistorted but not rectified (middle). Undistorted and Rectified (bottom)

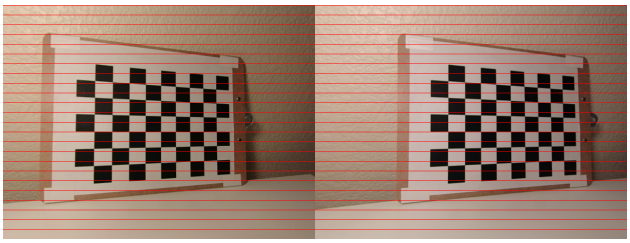
3 Sparse Depth Triangulation

Up until now, we have been able to calibrate the cameras and estimate the transformation between the camera system, using the feature points. Using these feature points from the two views, we have been able to triangulate the position of the feature points. In this task, the idea is to go ahead and reconstruct the 3D view of the object.

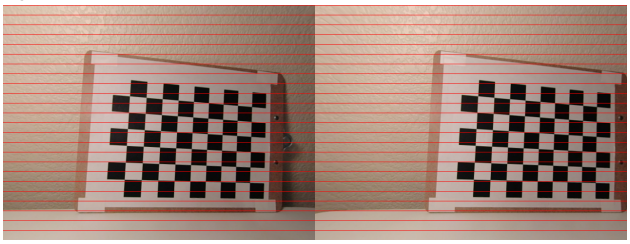
3.1 Theory

This project will require us to understand the geometry between the two camera and the object. The idea here is that we want want to search for corresponding points in stereo matching. Refer to the figure 10 for more details regarding the terms.

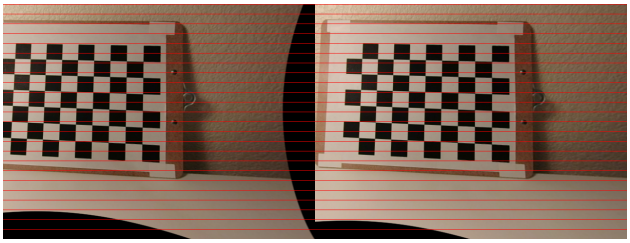
C_l and C_r are the two camera and P is the object. The plane joining the two cameras and object is called the epipolar plane π. The line connecting the cameras C_l and C_r is called the baseline (It is also called the epipole). The intersection of the image plane with the epipolar plane gives rise to a line, which is called the epipolar line. Notice that the the intersection of the plane π with the image plane of the left camera(R_l) gives rise to a line, which can be denoted by l_{pl}.



(a) Rectified images: left_0.png on the left and right_0.png on right



(b) Rectified images: left_0.png on the left and left_1.png on right



(c) Rectified images: right_0.png on the left and right_1.png on right

Figure 6: Images post rectification

Since the plane π contains the point p_l as well the line l_{pl}, the corresponding line l_{pr} is also bound to be contained in the similar plane [2, 4].

Consider the right image plane I_r. The line l_{pr} can be found by the cross product of the points e_r and p_r.

$$l_{pr} = e_r \times p_r$$

. The cross product can be written as matrix multiplication of a skew symmetric matrix and a vector by

$$\begin{aligned} l_{pr} &= e_r \times p_r \\ l_{pr} &= [e_r]_\times p_r \end{aligned} \tag{4}$$

The point p_r can also be written as a a transformation of the point p_l. Assuming that there is some transformation H, we can write $p_r = Hp_l$. Substituting this in equation 4, we can see that

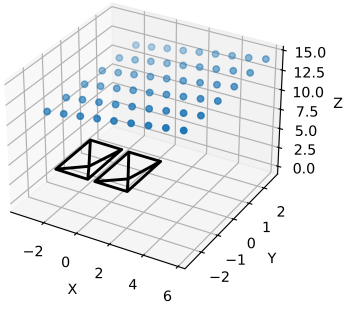
$$\begin{aligned} l_{pr} &= [e_r]_\times H p_r \\ l_{pr} &= F p_l \qquad \text{where } F = [e_r]_\times H \end{aligned}$$

Since the point the p_r lines on the line l_{pr}, the dot product of this point and line lies is equal to 0. We can come up with a constraint for epipolar lines, which is

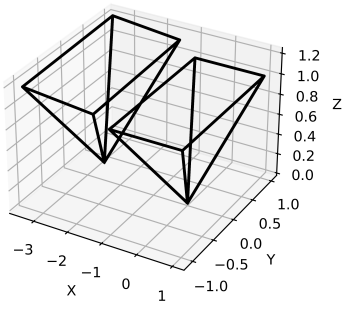
$$p_l^T F p_r = 0$$

3.2 Experiments

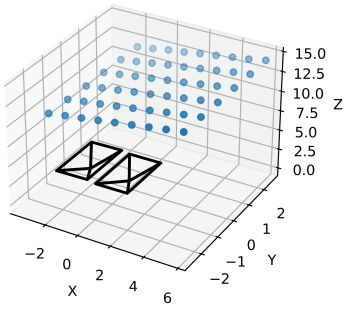
We do the calibration and undistortion using the parameters obtained from the first two experiments. The key process from now is to detect good features of the image and match these features.



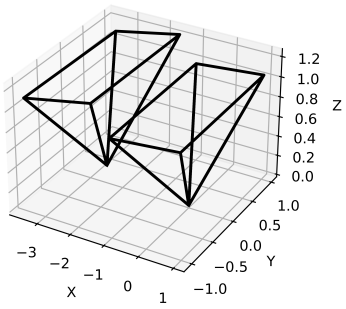
(a) Camera Pose before rectification (with image points)



(b) Camera Pose before rectification (without image points)



(c) Camera Pose after rectification (with image points)



(d) Camera Pose after rectification (without image points)

Figure 7: Camera poses before and after rectification using left_0.png and right_0.png

For the process of detecting features in the images, ORB algorithm [5] was used (cv.ORB_create()). The first step is to get the keypoints in the two images independently. Since the number of keypoints produced are usually very high and very close, we want to ensure that we retain only the good keypoints (the ones with maximum response in a given area). This is called non-maximal suppression. I implemented the non-maximal suppression from scratch (the complexity of which was of the complexity $\mathcal{O}(n^2)$. From this, we now have a pair of good features for both the views. This is shown in 11

Once these features are generated, we go ahead to match these features. Pairs of matches generated initially using the matcher function. These provide a pair of matches for the left and the right views. We apply the epipolar constrain tfor every match in order to detect if it is a good match or not. Ideally, the value should be 0, but this is often not the case in practice. For the experiments, I set up an epsilon value ($\epsilon = 0.5$ after a lot of hit and trial). If the value of the epipolar constraint was less than this epsilon, the match was kept, other wise it was discarded. The output is shown in 12

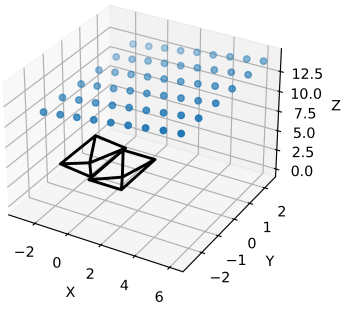
One we have the feature points and the know transformation between the images, we can easily go ahead to do the sparse depth triangulation. The process is essentially the same as experiment 2. The difference is in the way the matching feature points, which are generated using the process mentioned above. We utilise the cv2.triangulatePoints() and the output corrdinates are de-homogenized in order to get the 3D points of the features. Look at 13.

4 Dense Depth Triangulation

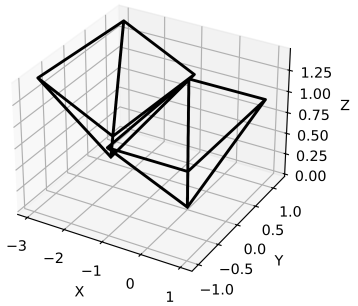
The motive behind this task is to generate a depth map, given the two views of an object.

4.1 Theory

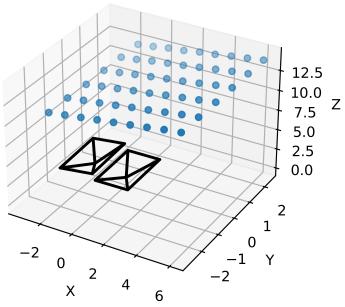
The key idea here is that when we have a rectified set of images, we can obtain the depth of every pixel. The key idea here is that rectification aligns the images such that all the features for the two images lie on a single x-coordinate line. This allows us to check the difference between the two pixel values at corresponding x-coordinates. This difference



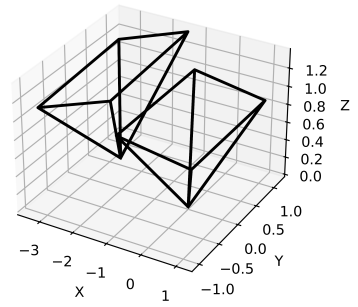
(a) Camera Pose before rectification (with image points)



(b) Camera Pose before rectification (without image points)



(c) Camera Pose after rectification (with image points)



(d) Camera Pose after rectification (without image points)

Figure 8: Camera poses before and after rectification using left_0.png and left_1.png

map is essentially the disparity map. Consider the figure 14 for a proof which relates the depth estimation and disparity map [6]. By similar triangles (orange and yellow triangles), we can write the equation as

$$\begin{aligned} \frac{T}{Z} &= \frac{T-(X_l-X_r)}{Z-f} \\ T(Z-f) &= Z(T-(X_l-X_r)) \\ TZ-Tf &= ZT-ZX_l+ZX_r \\ Tf &= ZX_l-ZX_r \\ Z &= \frac{Tf}{X_l-X_r} \end{aligned} \tag{5}$$

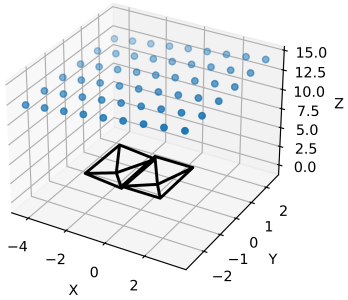
The idea behind match blocking is that it searches for a match in the same x direction. This gives us a matching score which can be used for calculating the disparity map. From 5, we notice that the depth and disparity are inversely proportional to each other.

4.2 Experiments

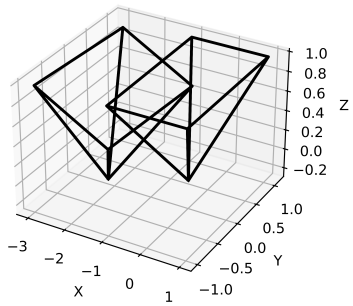
I start with the undistorted and rectified images, and went ahead to calculate the initial disparity map using the block matching algorithm (cv2.stereoSGBM. After trying out a lot of parameters, I couldn't obtain the result as shown in the class. I used the parameters which were shown in the class and the disparity maps started working well. Results from this experiment is in 15.

5 How to run the code?

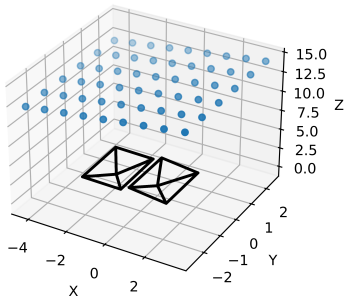
Please keep the base directory as project_2a. From here, for task_1, run the command python code/task_1/task_1.py From here, for task_1, run the command python code/task_2/task_2.py



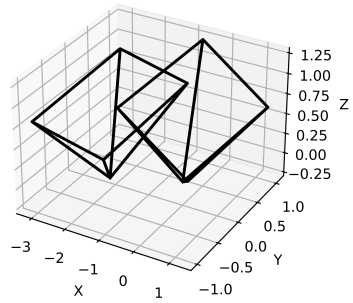
(a) Camera Pose before rectification (with image points)



(b) Camera Pose before rectification (without image points)



(c) Camera Pose after rectification (with image points)



(d) Camera Pose after rectification (without image points)

Figure 9: Camera poses before and after rectification using right_0.png and right_1.png

From here, for task_1, run the command python code/task_3/task_3.py
From here, for task_1, run the command python code/task_4/task_4.py

The necessary libraries are:

- opencv-contrib-python: cv2 version: 4.5.1
- matplotlib version: 3.3.4
- os
- numpy version: 1.20.1
- random

References

[1] Peter Corke. *Robotics, Vision and Control: Fundamental Algorithms in MATLAB*. Springer Publishing Company, Incorporated, 1st edition, 2013.

[2] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, USA, 2 edition, 2003.

[3] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer Vision in C++ with the OpenCV Library*. O'Reilly Media, Inc., 2nd edition, 2013.

[4] Xinghua Chai, Fuqiang Zhou, and Xin Chen. Epipolar constraint of single-camera mirror binocular stereo vision systems. *Optical Engineering*, 56(8):1 – 8, 2017.

[5] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International conference on computer vision*, pages 2564–2571. Ieee, 2011.

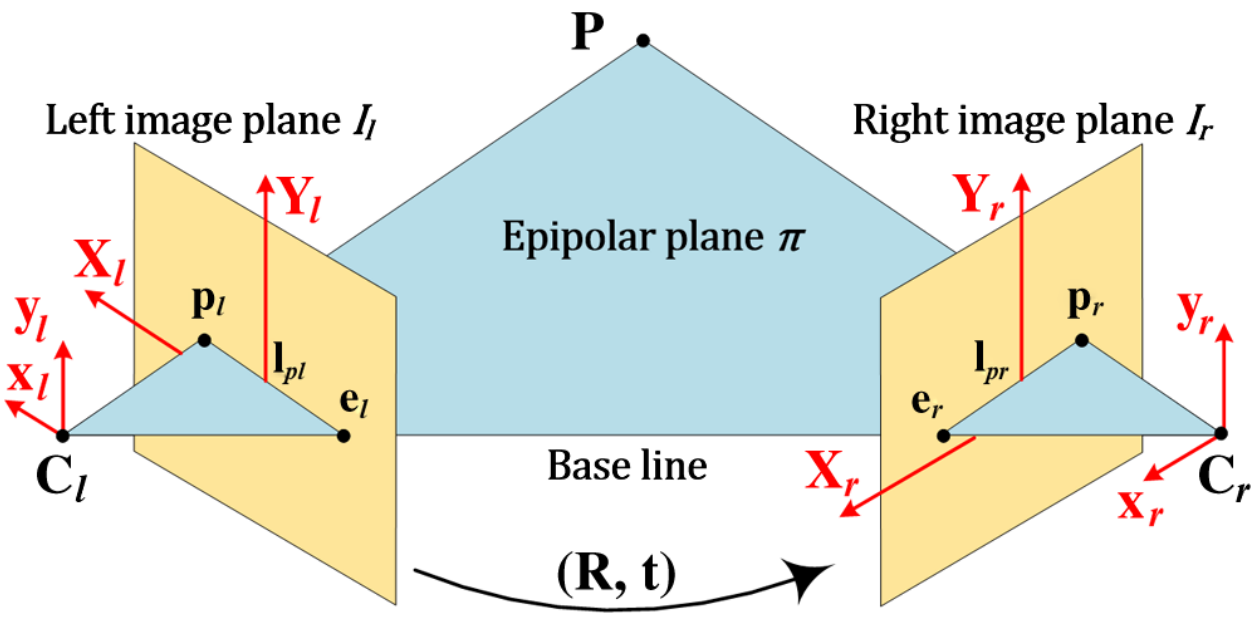
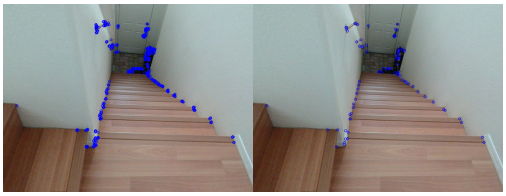


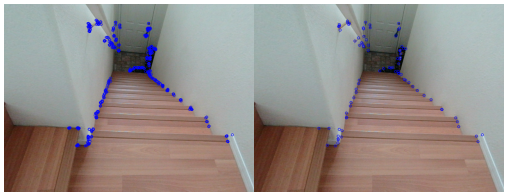
Figure 10: C_l and C_r are the two camera and P is the object. The plane joining the two cameras and object is called the epipolar plane π. The line connecting the cameras C_l and C_r is called the baseline (It is also called the epipole). the intersection of the image plane with the epipolar plane gives rise to a line, which is called the epipolar line.

(a) left_0.png

(b) right_0.png



(c) left_8.png



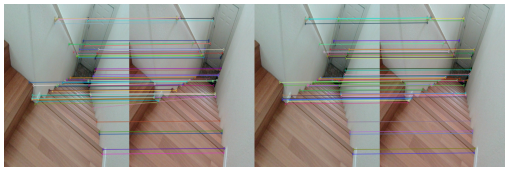
(d) right_8.png

Figure 11: Features detected in the image using ORB algorithm before and after non-maximal suppression.

(a) Matches using left_0.png and right_0.png

(b) Matches using left_8.png and right_8.png

(c) Matches using left_3.png and right_3.png

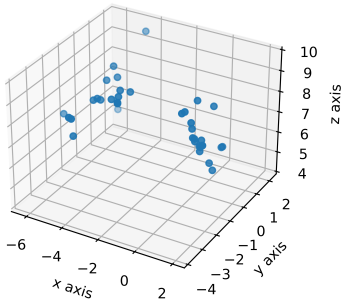


(d) Matches using left_9.png and right_9.png

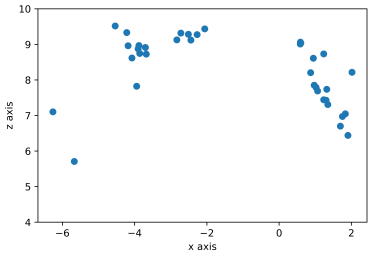
Figure 12: Initial matches found from matches function and the corresponding images show the good matches after applying the epipolar constraints. Look at figure c for a clearer view.

[6] Emanuele Trucco and Alessandro Verri. *Introductory Techniques for 3-D Computer Vision*. Prentice Hall PTR, USA, 1998.

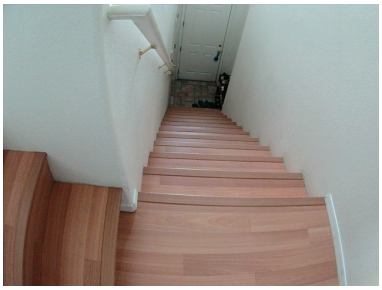
(a) Original Image



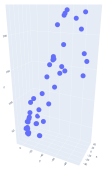
(b) Obtained 3D points



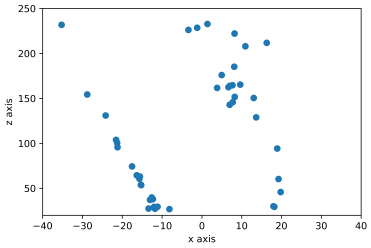
(c) x-z axis plane view



(d) Original Image

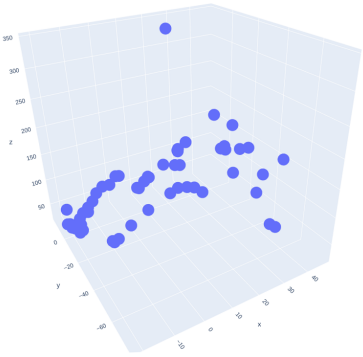


(e) Obtained 3D points

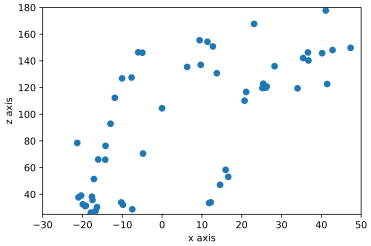


(f) x-z axis plane view

(g) Original Image



(h) Obtained 3D points



(i) x-z axis plane view

Figure 13: The views of three different scenes from 3D triangulation.

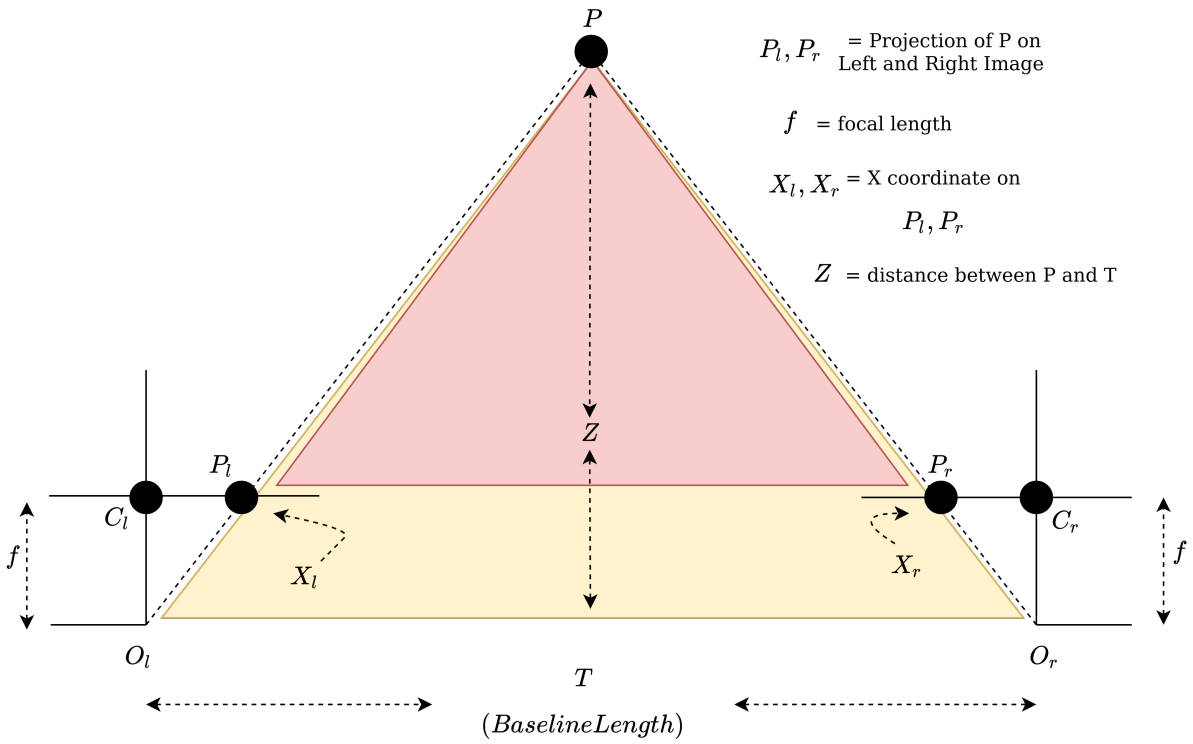
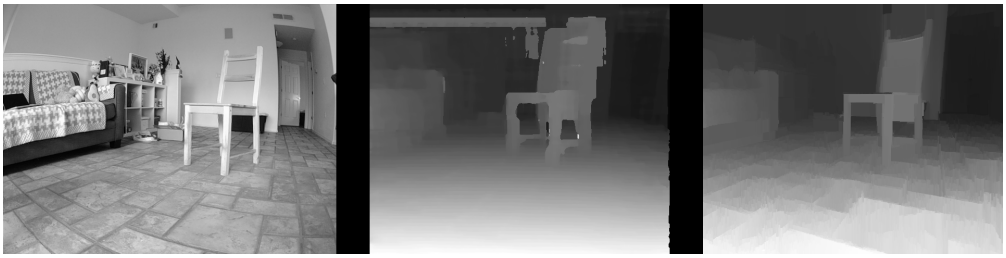


Figure 14: Derivation for depth estimation and disparity maps



(a) Disparity Maps for left_5.png and right_5.png

(b) Disparity Maps for left_7.png and right_7.png

(c) Disparity Maps for left_9.png and right_9.png

Figure 15: Original greyscale images (left most). Simple disparity maps (middle). Filtered disparity maps (right most)