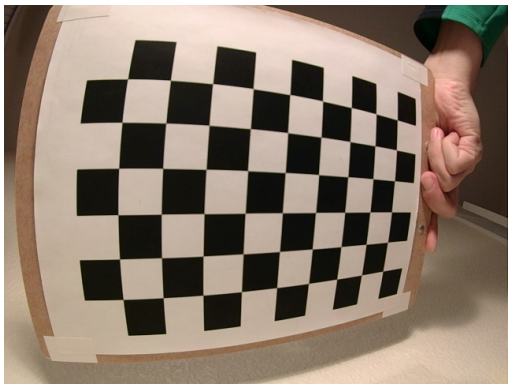
Aniket N. Prabhu
1217704488
anprabh1@asu.edu

1 Pinhole camera model and calibration

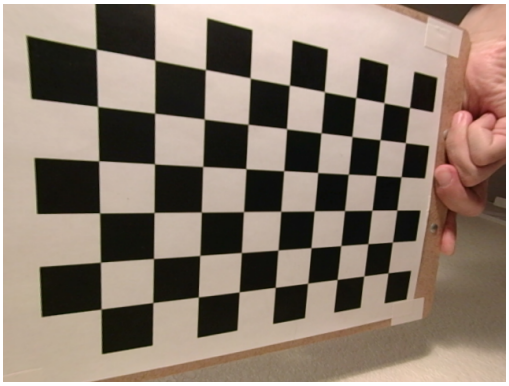
The first task involves calibrating the camera by calculating the camera intrinsic parameters with 3D-to-2D point correspondences. For this purpose, I have used all the 22 images provided, 11 per camera. The process is as follows:

1. Loading the images for each camera using the OpenCV library function ”imread()”
2. Extracting 3D-to-2D point correspondences using the OpenCV library function ”findChessboardCorners()”
3. Calculating the camera intrinsic parameters (intrinsic matrix and distortion coefficients) with the OpenCV library function ”calibrateCamera()”
4. Undistorting and checking the calibration results using the OpenCV library function ”initUndistortRectifyMap()” and ”remap()”

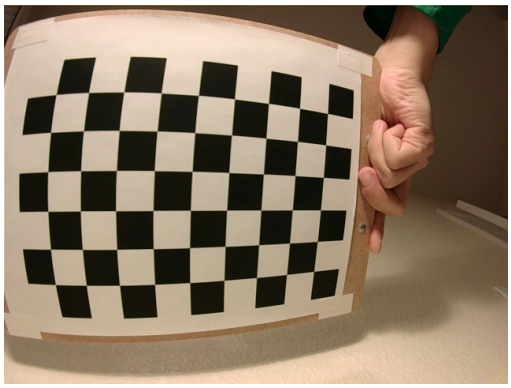
Using images *left_2.png* and *right_2.png*, we get the following results:



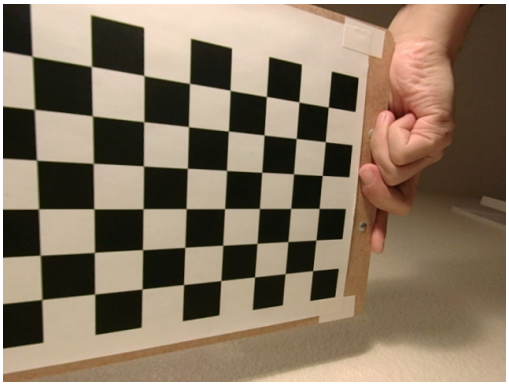
(a) **left_2.png** original



(b) **left_2.png** calibrated



(c) **right_2.png** original



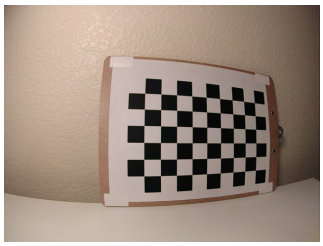
(d) **right_2.png** calibrated

An interesting observation you may have is that the image gets cropped upon calibration due to loss of pixels. Finally, the camera parameters are stored in **xml (eXtensible Markup Language)** format for convenience in latter tasks.

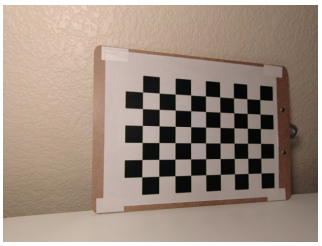
2 Stereo Calibration and Rectification

Difference between the perception of the two cameras is expressed through the *Rotation matrix* and *Translation vector*, (R, t). Individual camera parameters and correspondences obtained in task 1 can therefore be used in conjunction with (R, t) to calibrate the two camera so that they can collectively perceive the target. This is useful in triangulation wherein the camera system determines the spatial points and the shape of the target object. The process for this task is as follows:

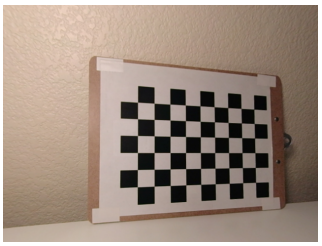
1. Loading the images and the intrinsic parameters for each camera using the OpenCV library function "imread()" and OpenCV's "FileStorage" class
2. Extracting 3D-to-2D point correspondences using the OpenCV library function "findChessboardCorners()"
3. Calibrating the stereo camera system using the 3D-to-2D point correspondences obtained earlier and the OpenCV library function "stereoCalibrate()" which gives us (R, t).
4. The results of stereo calibration **left_0.png** and **right_0.png** are as follows. The differences are minute since both the cameras were used instead of a single one (**left_0.png** and **left_1.png**). On the other hand, if we were to use two scenes from the same camera, the difference is more noticeable:



(a) **left_0.png** original

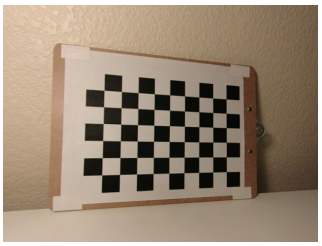


(b) **left_0.png** calibrated

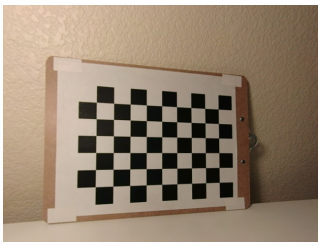


(c) **left_0.png** rectified

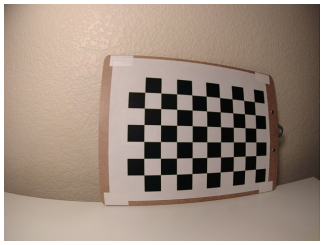
(d) **right_0.png** original



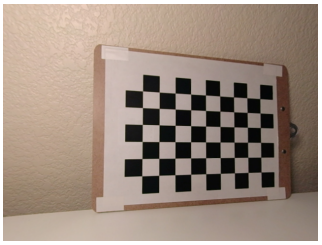
(e) **right_0.png** calibrated



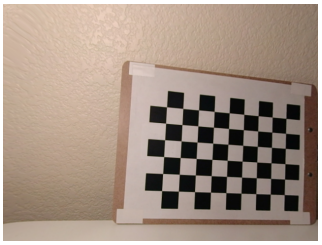
(f) **right_0.png** rectified



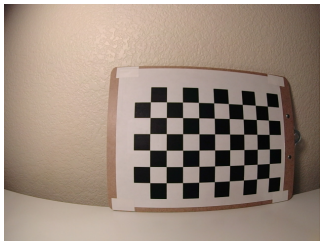
(a) **left_0.png** original



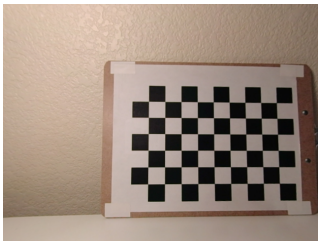
(b) **left_0.png** calibrated



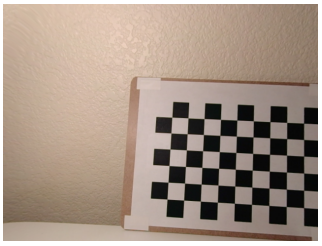
(c) **left_0.png** rectified



(d) **left_1.png** original

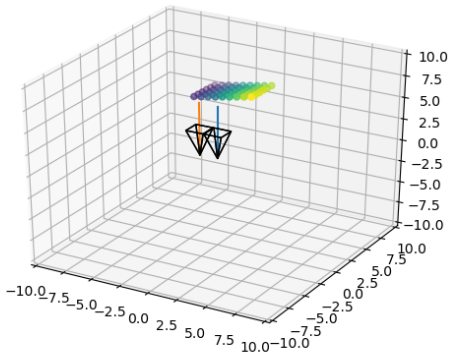


(e) **left_1.png** calibrated

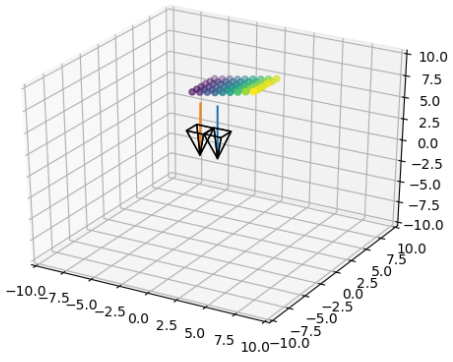


(f) **left_1.png** rectified

5. Checking the rectification results via plotting the camera positions as well as the board pattern in 3D

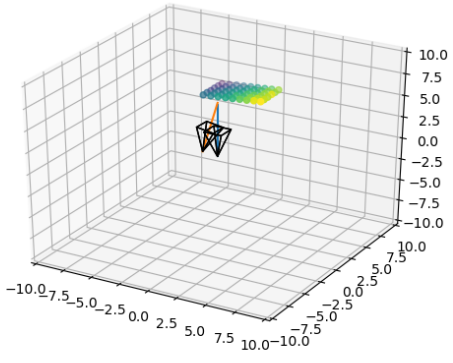


(a) for **left_0.png** and **right_0.png** original

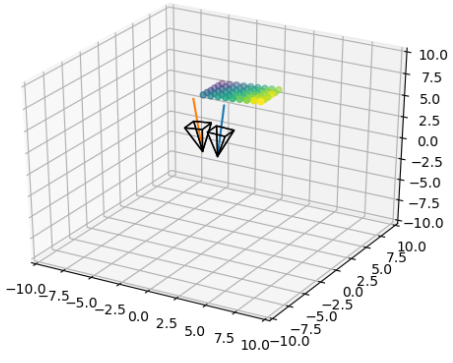


(b) for **left_0.png** and **right_0.png** rectified

Once again, the rectification differences are minute for **left_0.png** and **right_0.png** unlike for **left_0.png** and **left_1.png**:



(a) for **left_0.png** and **left_1.png** original



(b) for **left_0.png** and **left_1.png** rectified

(P. T. O.)

3 Sparse Depth Triangulation

By using two sensors, we can observe an object and map corresponding points in 3D space. The projection centres of the sensors and the perceived point on the target's surface form a spatial triangle. Hence, this process of mapping 3D points using 2D views is called triangulation. Based on keypoints detected and matched using a keypoint (feature point) matcher, we can triangulate points. Since these keypoints are sparse, this process is called Sparse Depth Triangulation. For our purpose, we have used ORB (Oriented fast and Rotated Brief) keypoint detector on two views (**left_0.png** and **right_0.png**) and then matched those keypoints using the BF (Brute Force) Matcher. The process for this task is as follows:

1. Loading images in grayscale for the *ORB detector* and the camera parameters obtained in previous tasks
2. Detecting the features using the *ORB detector* and OpenCV library function "drawKeypoints()". A feature/keypoint's response value detects how recognizable it is. To minimize errors in feature matching ahead, we have taken out a number of features which are *local maxima*. Local maxima have been found by taking out keypoints with the highest response value in a certain pixel radius (taken as 5 after trial and error). The results are as follows

(a) **left_0.png** original

(b) **right_0.png** original

(c) **left_0.png** keypoints

(d) **right_0.png** keypoints

3. Matching features using *BFMatcher* and OpenCV library function "drawMatches()". Even after isolating the local maxima, there is a possibility to get false/bad feature matches. In order to eliminate such matches, The absolute difference between the y-coordinates of the left and right keypoints has been used to select the good matches; it the difference is less than 10(selected upon trial and error), then it is a good match. The results are as follows

(P. T. O.)

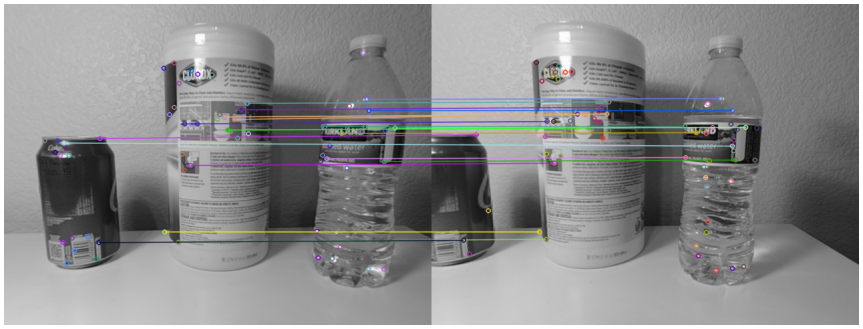
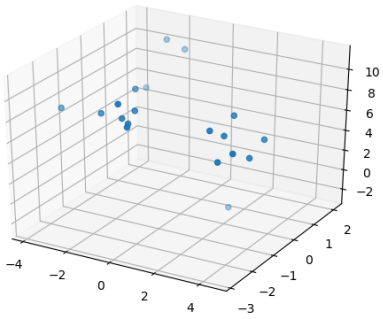
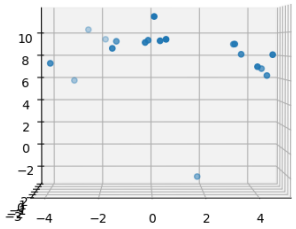


Figure 7: Good matches between left and right keypoints

4. Triangulating feature points and checking sparse depth results. To 3D-plot the sparse depth results, the keypoints needed to be converted into pixel coordinates, normalized via cv2.undistortPoints(), triangulated with cv2.triangulatePoints(), and finally converted from homogeneous to euclidean space with cv2.convertPointsFromHomogeneous(). The results are as follows: We also have the results for



(a) Sparse depth triangulation results (view 1)

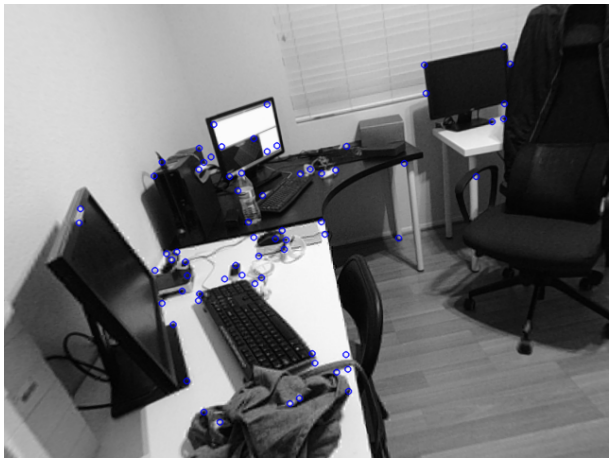


(b) Sparse depth triangulation results (view 2)

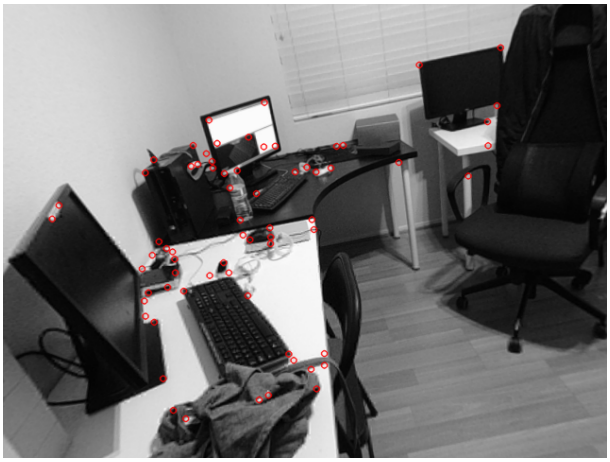
”left_6.png” and **”right_6.png”** wherein the objects are far away

(a) **left_6.png** original

(b) **right_6.png** original



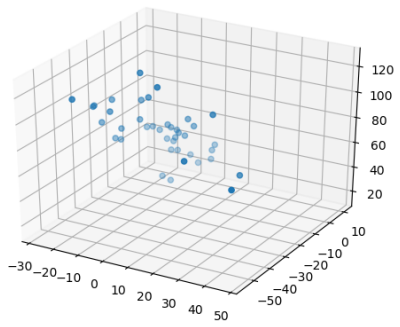
(c) **left_6.png** keypoints



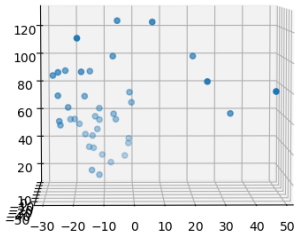
(d) **right_6.png** keypoints

Figure 10: Good matches between left and right keypoints

(P. T. O.)



(a) Sparse depth triangulation results (view 1)



(b) Sparse depth triangulation results (view 2)

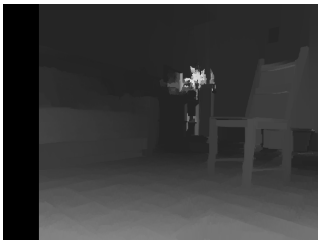
4 Dense depth triangulation

This form of triangulation is more intense as it aims to triangulate every single pixel instead of a sparse set of points. The process for this task is as follows

1. Loading images, and the camera and stereo parameters obtained in previous tasks
2. Block match for each pixel on the images to obtain a disparity map. Using two sensors to perceive the target involves taking into consideration the parallax (separation between the two sensors); Each sensor perceives the target object in its own way and due to this, the difference in the location of the target in the corresponding image seen by each sensor is called disparity. Disparity map is a 2D map of the target reduced from 3D space. Two popular algorithms to calculate the disparity map are StereoBM (Stereo Block Matching) and StereoSGBM(Stereo SemiGlobal Block Matching). I have used StereoSGBM for this task. StereoSGBM class does not rectify images on its own and prior rectification is required which I have done.
3. Calculate depth for each pixel using the disparity map. For this purpose, I have used the OpenCV library function ”reprojectImageTo3D()”. The formula to calculate depth for each pixel given the disparity is as follows:
$$d = \frac{B \cdot f}{|x_L - x_R|} = \frac{B \cdot f_{s_X}}{|u_L - u_R|}$$
wherein $|x_L - x_R|$ = Disparity metric unit, $|u_L - u_R|$ = Disparity in pixels, B = Baseline length (62mm here), f = focal length (cannot be obtained from camera intrinsics), f_{s_X} = focal length obtained from camera intrinsics
4. Checking results of disparity images and depth images. For this purpose, I have used the pairs, (”left_4.png”, ”right_4.png”) and (”left_8.png”, ”right_8.png”). The results of disparity images are as follows

(a) **rectified left_4.png**

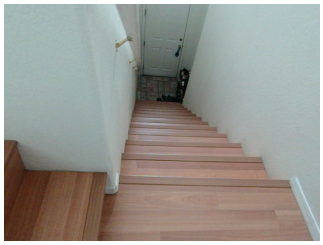
(b) **rectified right_0.png**



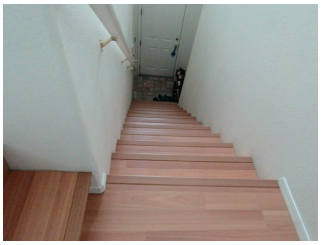
(c) **Disparity image of (”left_4.png”, ”right_4.png”) with WLS filter.png**

Figure 12: **(”left_4.png”, ”right_4.png”)** results

(P. T. O.)



(a) **rectified left_8.png**

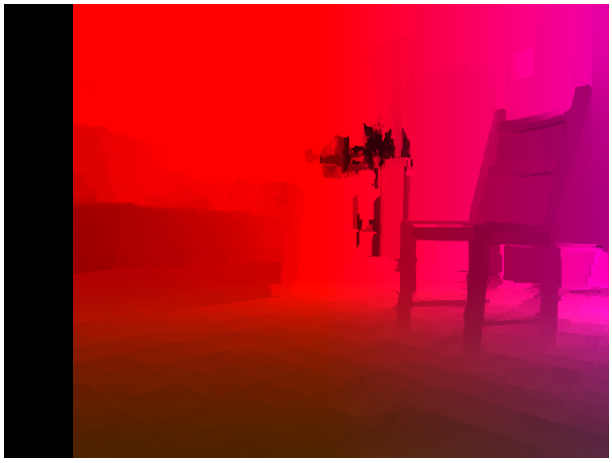


(b) **rectified right_8.png**

(c) **Disparity image of ("left_8.png", "right_8.png") with WLS filter.png**

Figure 13: **("left_8.png", "right_8.png")** disparity results

And finally, we have the depth images



(a) **Depth image of ("left_4.png", "right_4.png")**

(b) **Depth image of ("left_8.png", "right_8.png")**

Figure 14: **Depth image results left to write: ("left_4.png", "right_4.png"), ("left_8.png", "right_8.png")**

The End.