CMPE 264: Project 2

Mariette Souppe
msouppe@ucsc.edu

Andrea David
andavid@ucsc.edu

December 9, 2018

Github Repository

Part 1: Camera calibration (intrinsic parameters)

In order to calibrate the camera, we first had to take ten images from different perspectives of a checkerboard pattern. After getting those images we were able to calibrate the camera from the OpenCV tutorial. The first function that was used was called $findChessboardCorners()$, which finds the internal corners in the chessboard and returns the corner points. Next we used the $cornerSubPix()$ function to increase the accuracy of the corner points from the previous function. Next, using the $calibrateCamera()$ function we are able to obtain the intrinsic matrix, radial distortion coefficients, rotation vector, and translation vector.

As a result we obtained out intrinsic matrix, *K*,

$$K = \begin{vmatrix} 2454.69 & 0 & 1576.69 \\ 0 & 2449.58 & 937.02 \\ 0 & 0 & 1 \end{vmatrix}$$

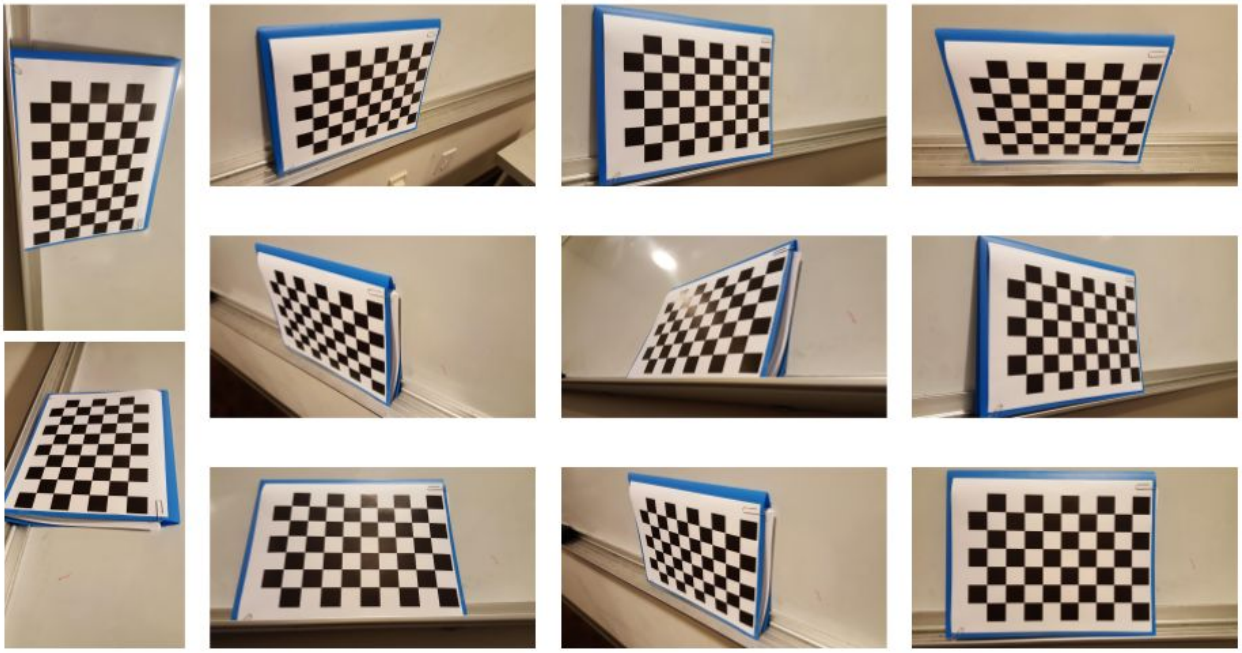
and radial distortion coefficients,

$$Radial\ distortion\ coefficients = [0.09672 \quad 0.12303 \quad -0.00296 \quad -0.00547 \quad -1.30025]$$

Lastly, in order to make sure that our camera calibration was good enough we had to look at the mean square error using the data that was obtain from the previous function, $calibrateCamera()$. The mean square reprojection error needed be less than 1 pixel or more ideally less than 0.5 pixels. Below is the result of our mean square reprojection error:

$$MSE = 0.17480$$

Below are the images that were used for the camera calibration:

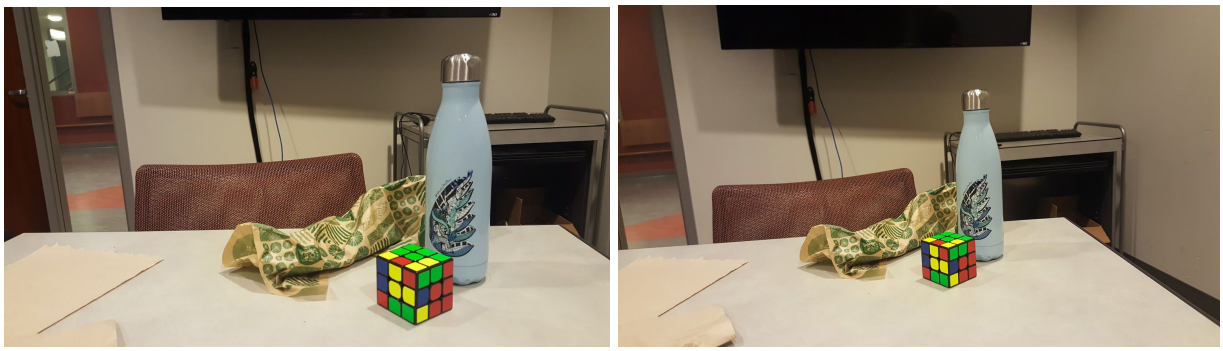


Part 2: Take the pictures

In this step, pictures of a scene with objects at different distances were taken. It is important to note that in order for the remainder of the project to turn out well these pair of images needed to follow the following criteria:

1. A good overlap of common area between both images
2. Diverse amount of texture in the images
3. Objects are from different distances
4. There is no point of infinity in either pair of the images
5. Enough parallax between the images, meaning from the left image to the right image the camera must have been translated and slightly rotated

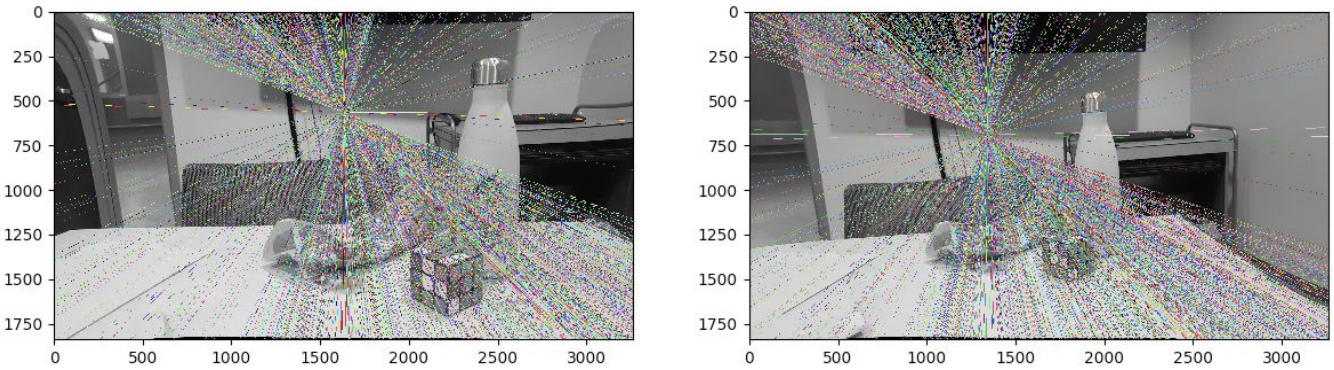
Below are the pair of images that will be used throughout the remainder of the project for further computation.



Part 3: Compute the relative camera pose $R_L^R,\ \boldsymbol{r}^R$

After taking our two images of a particular scene, we can now find the rotation matrix from the left camera to the right camera and translation matrix from the left camera's reference frame.

Using the intrinsic matrix, K, and radial coefficients from part 1 we first undistort the images using the $undistort()$ function. The $undistort()$ function returns a new undistorted image. With the new image and the camera calibration outputs, we can then obtain key points and descriptors for each image from $create\_feature\_points()$. With those key points and descriptors of an image we can get the fundamental matrix, F, and match feature points from both images with $match\_feature\_points()$. This allows epipolar lines to be drawn on both images. Below is our result:

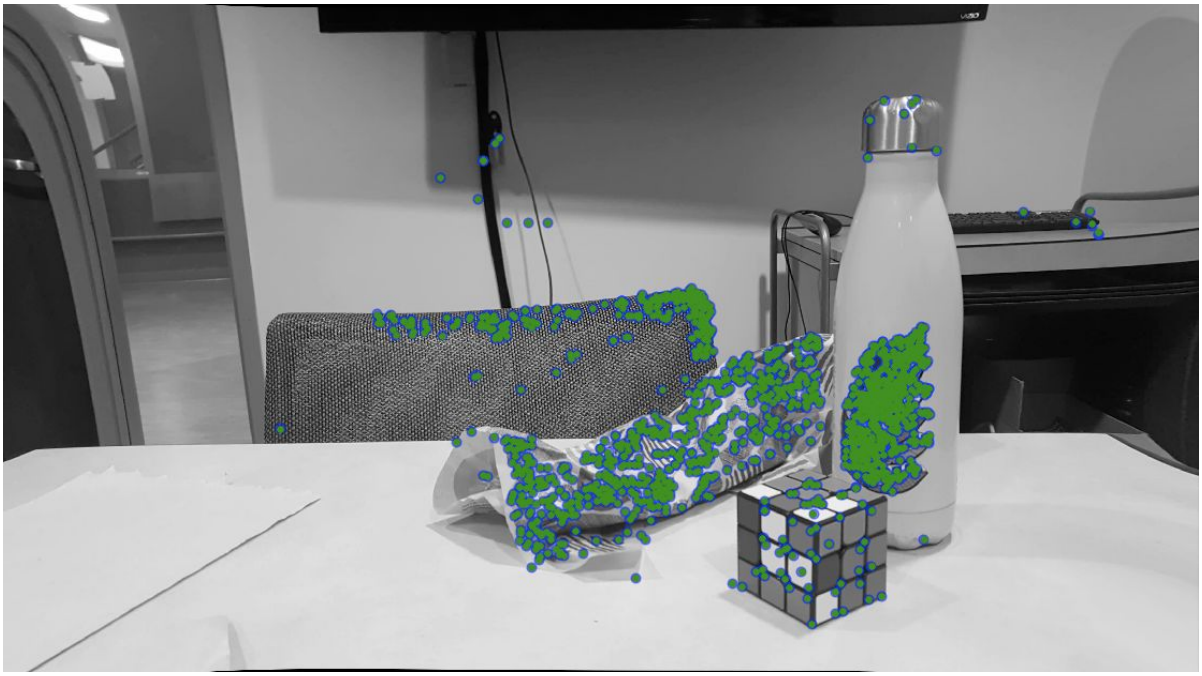


Using the feature points and the intrinsic matrix, K, we can find the essential matrix, E, using $findEssentialMat()$ and obtain our rotational matrices and translation matrix using $decomposeEssentialMat()$. It is also important to note that $decomposeEssentialMat()$ has four possible solutions; $(R1,\ r),\ (R1,\ -r),\ (R2,\ r),\ (R2,\ -r)$. We need to choose the right combination that gives positive *z* for all the feature points in the correct camera pose. As a result we obtained our rotation matrix, R_L^R, and translation vector, $\boldsymbol{r}^R$ with the combination $(R1,\ -r)$:

$$R_L^R = \begin{vmatrix} \text{-0.99564} & \text{-0.01974} & \text{-0.09119} \\ 0.03989 & \text{-0.97361} & \text{-0.22470} \\ \text{-0.08435} & \text{-0.22736} & 0.97015 \end{vmatrix} \qquad \boldsymbol{r}^R = \begin{vmatrix} \text{-0.10285} \\ \text{-0.08804} \\ 0.99079 \end{vmatrix}$$

Next, with R_L^R and $\boldsymbol{r}^R$ we can obtain the $P1$ and $P2$ matrices. For this project we are going to be using the left image as the reference frame so we will need to project the matching points from the right image onto the left image. Since the left camera is our reference frame $P1 = [I \mid 0]$, where I is the identity matrix and 0 is just a zero vector. Then to find $P2$ we use R_L^R and $\boldsymbol{r}^R$ where $P2 = [R_L^R \mid r^R]$. After obtaining the projection matrices and using the

matching points found earlier we can re-projected points on the first image using $triangulatePoints()$. Below is our result, where the blue points are the re-projection points from the second image onto the first image and green are the original points.



Part 4: Plane-sweeping stereo

Next, we obtain the minimum (d_{min}) and the maximum (d_{max}) depths that we found after triangulating the feature points. That is, the minimum and the maximum depth will be the minimum and maximum of the z coordinate entries of our normalized 4D-point that we obtained as an output from $triangulatePoints()$. Below are our values,

$$d_{min} = \quad 0.418319$$

$$d_{max} = \quad 0.486281$$

Next, we choose a set of N=20 equispaced distances $\{d_i\}$ that span the interval between d_{min} and d_{max}. With the obtained d values and the given $\boldsymbol{n}^{\mathsf{L}}$ = (0,0, −1), we obtain a set of planes.

To find the homography, obtained from using the first three columns and first three rows of the matrix $P_1 P_2^{-1}$. P_1 and P_2 are the projection matrices obtained in Part 3, but augmented with their last row equal to [$\boldsymbol{n}^{\mathsf{T}}$ d_i] to make them 4x4 matrices. After obtaining homographies using all 20 distances, we use them to warp the second image.

Below are the 20 warped images:

We compute the pixel-by-pixel absolute difference between the warped second image and the first image using *cv.absdiff()*. We run a block filter, *cv.blur()*, to obtain the block-filtered differences.

To find the index, i, that gives the lowest value of $SAD_i(x,y)$. We first convert the matrix that holds our block-filtered differences into a 1D row vector using *numpy.ravel()*. Using *argmin()* we find the index of the minimum value in the matrix. We then put these values into a new array. This array thus contains the depth for each image pixel.

To normalize these values, we divide each depth by the maximum value of the equispaced distances and multiply it by 255, which is the brightest pixel value.

Displaying the computed depth image gives us the output below. Here, the dark regions represent the objects in the image that are closest to us, while the light regions represent objects that are farther away in the image.