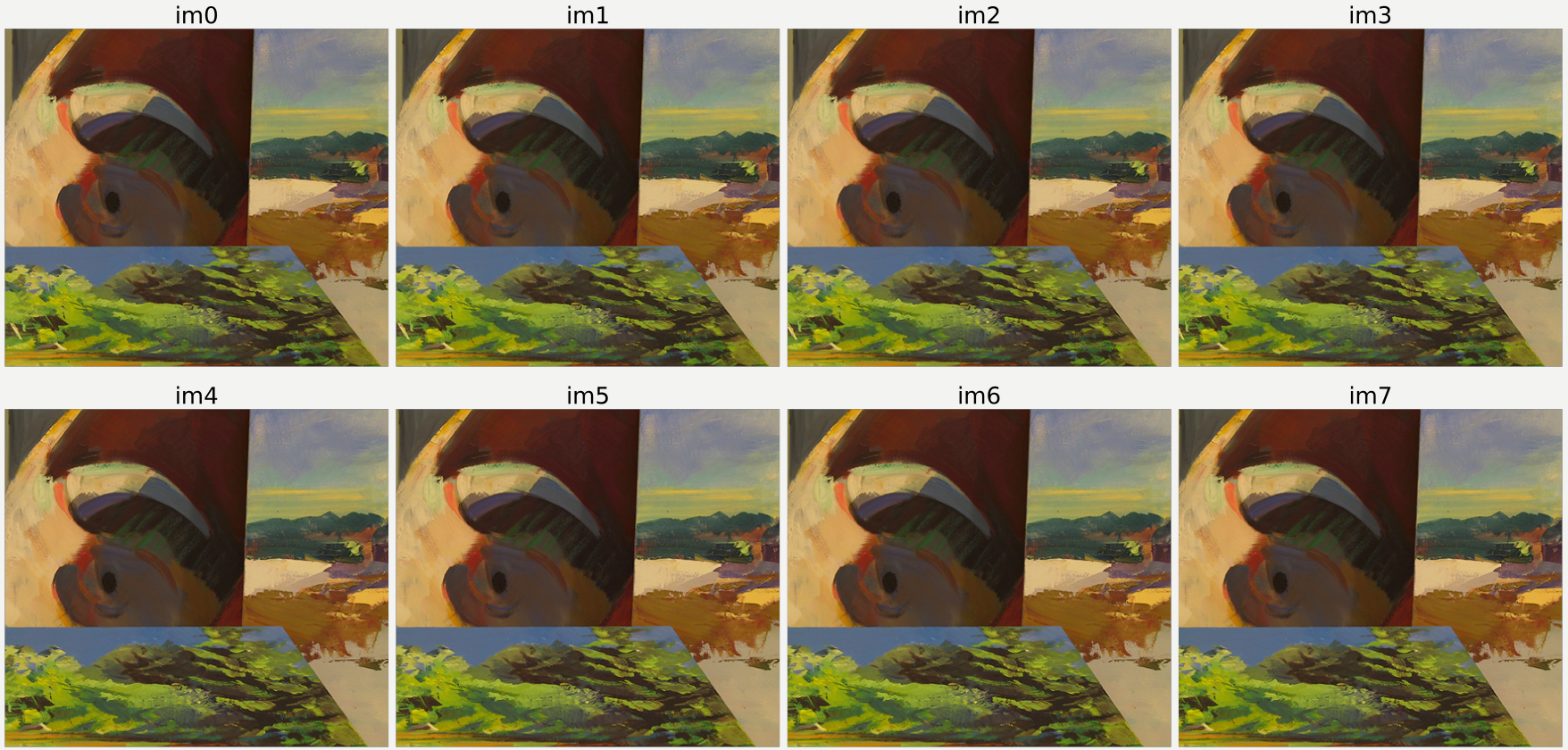
SHAPE FROM STEREO

ACTIVITY 07
PHYSICS 301

KENNETH M. LEO

STEREOMETRY

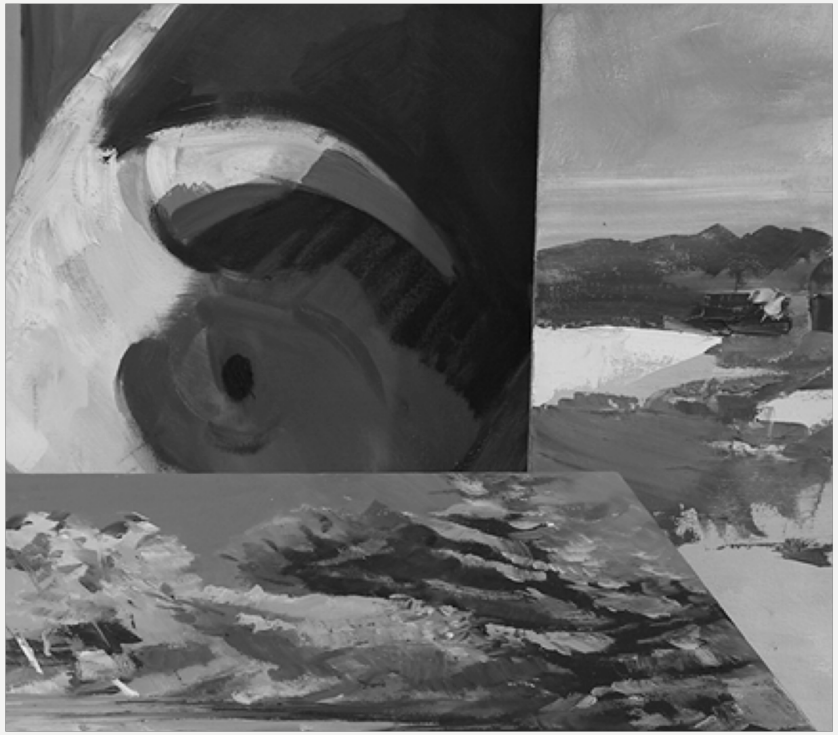
In this activity, we need photos that were taken with a camera with auto-focus turned off. When taking the pictures, the camera must be moved only on one-axis (x-axis) in order to make our computations correct.

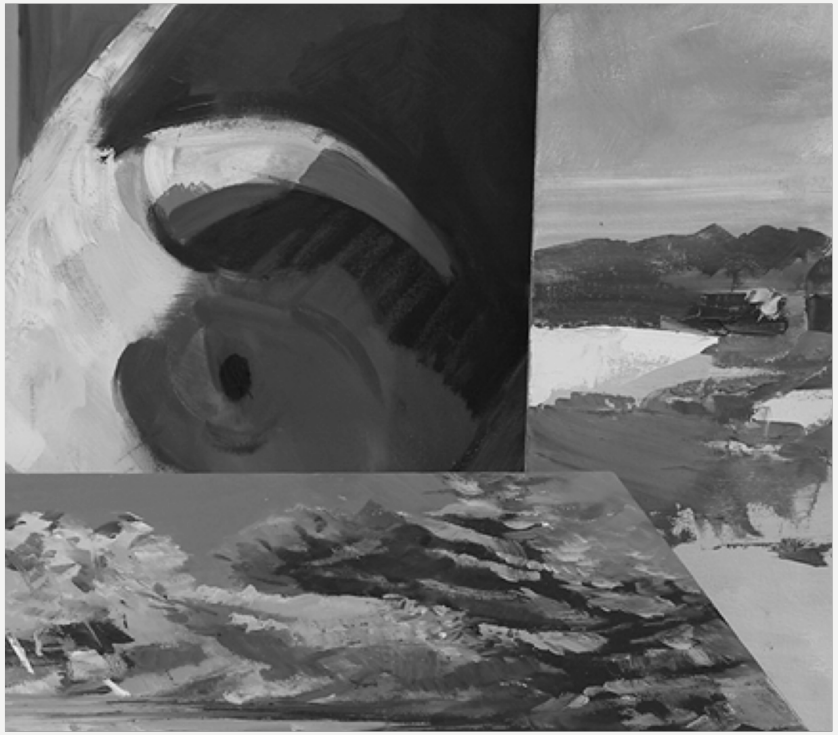


To check if my code is working (since I run all my codes from Activity 01 to 08 in Python), I used a dataset taken from https://vision.middlebury.edu/stereo/data/scenes2001/ . To be specific, I will be using their 'bull' dataset. This contains multiple photos, with some ground truth disparity maps.

LOADING IMAGE

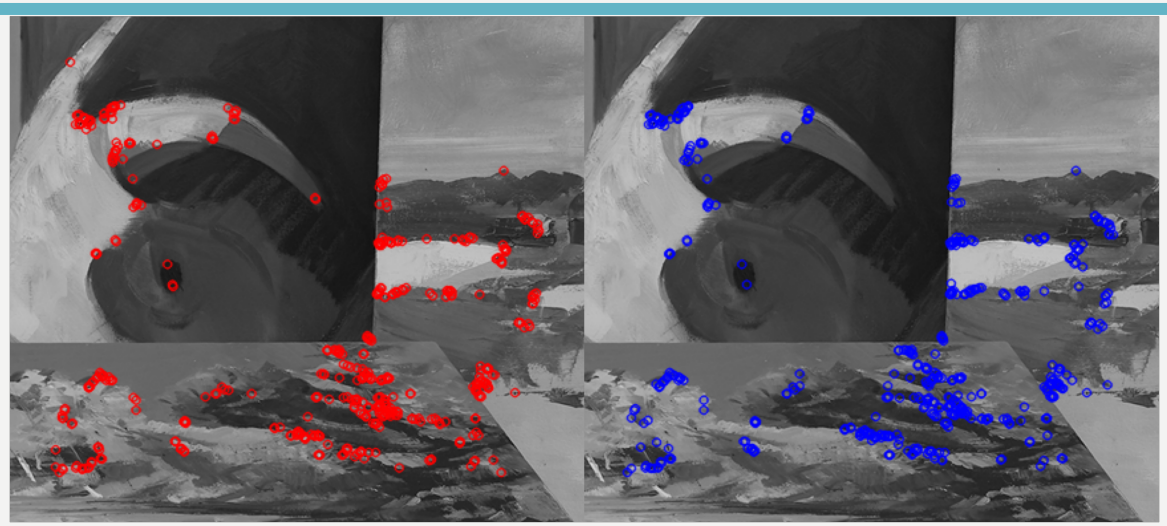
WE need to load the image first in grayscale. To do this, we can use cv2.imread(filename, cv2.IMREAD_GRAYSCALE). For the sake of demonstrating the process, we will be using the bull images with titles ‘im0’ and ‘im1’ respectively.

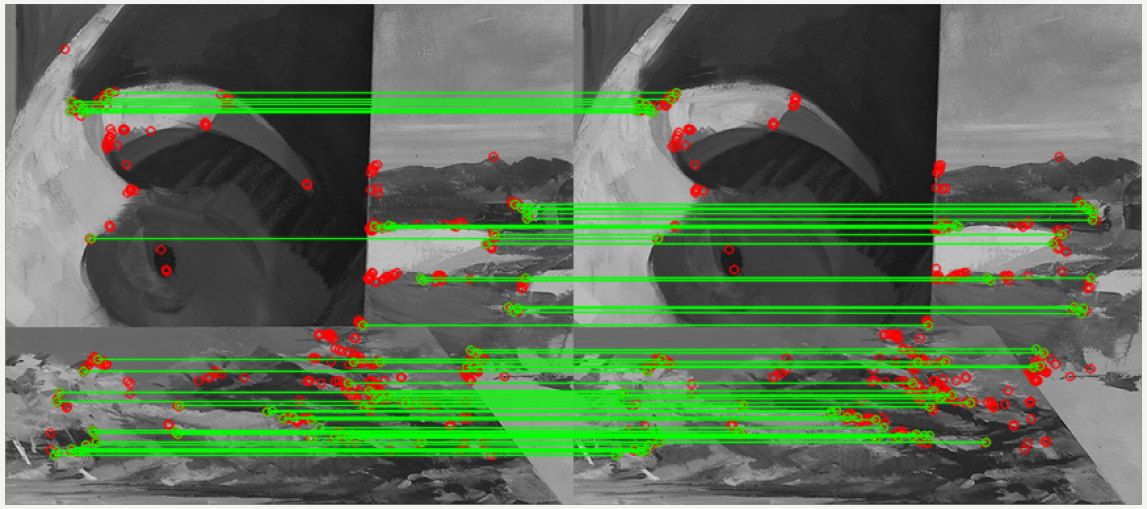




FINDING AND MATCHING KEYPOINTS

Next is we need to find the key features and descriptors of our images. In CV2, there are 3 ways to detect key features: SIFT, SURF, and ORB. SIFT and SURF are patented processes so that means its not free, so we used ORB instead. The trade-off is that SURF and SIFT can find more features, but ORB is much faster.





Match best keypoints. Next is we filter out the good features by using Lowe's ratio test. We see on the right that there are some keypoints ignored.

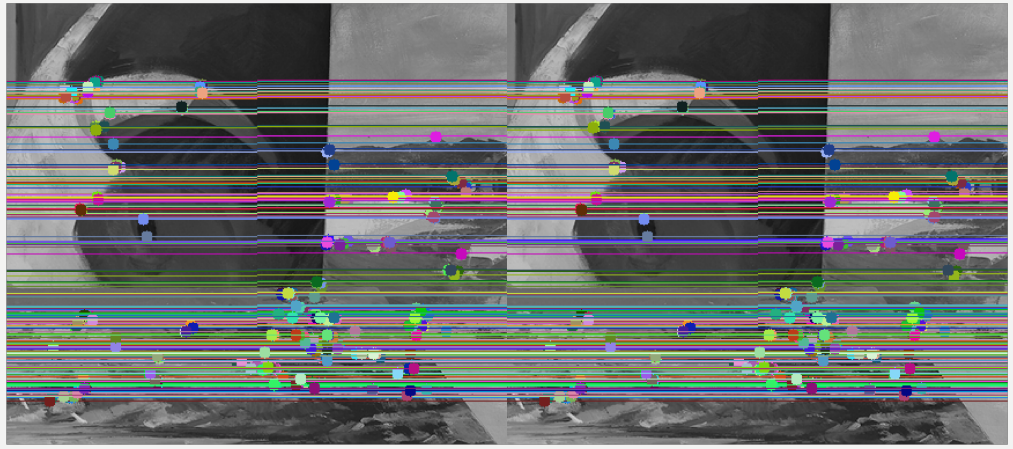
ESTIMATE FUNDAMENTAL MATRIX

Compute fundamental matrix using the good keypoints. Cv2 already has a built-in function for this which is cv2.findFundamentalMat(pts1, pts2, method = cv2.FM_RANSAC). The parameters pts1 and pts2 are the keypoints of the first and second images respectively, and the method is just the method you want to calculate the Fundamental matrix.

For the two images presented, the fundamental matrix is:

$$\begin{matrix} -1.80131457e-10 & 1.64199190e-04 & -4.58931041e-02 \\ -1.64141548e-04 & 1.28440934e-05 & -2.60345132e11 \\ 4.58881982e-02 & 2.60345132e11 & 1.00000000e00 \end{matrix}$$

RECTIFYING IMAGES



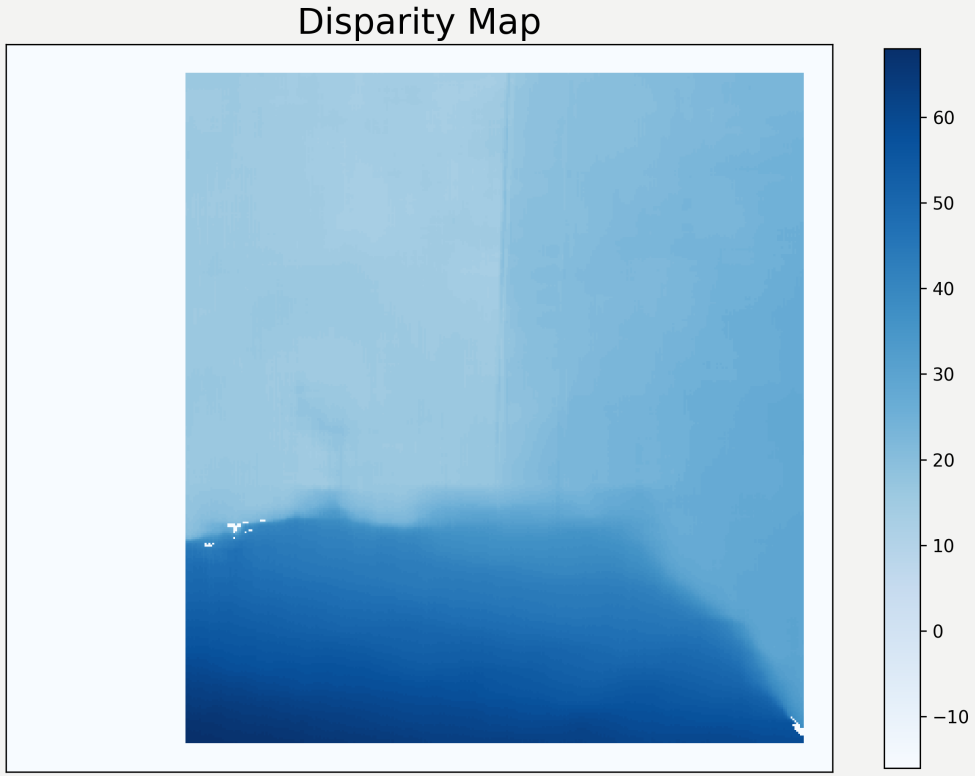
Check epilines. When taking the picture, there is a chance that the 'perspective' of the camera becomes skewed. This will make the epipolar lines of your photo at angle which would make calculations harder. So one can check the epilines of the images using cv2.computeCorrespondEpilines in Python. For the images I selected, their epilines are given by the figure on the left.

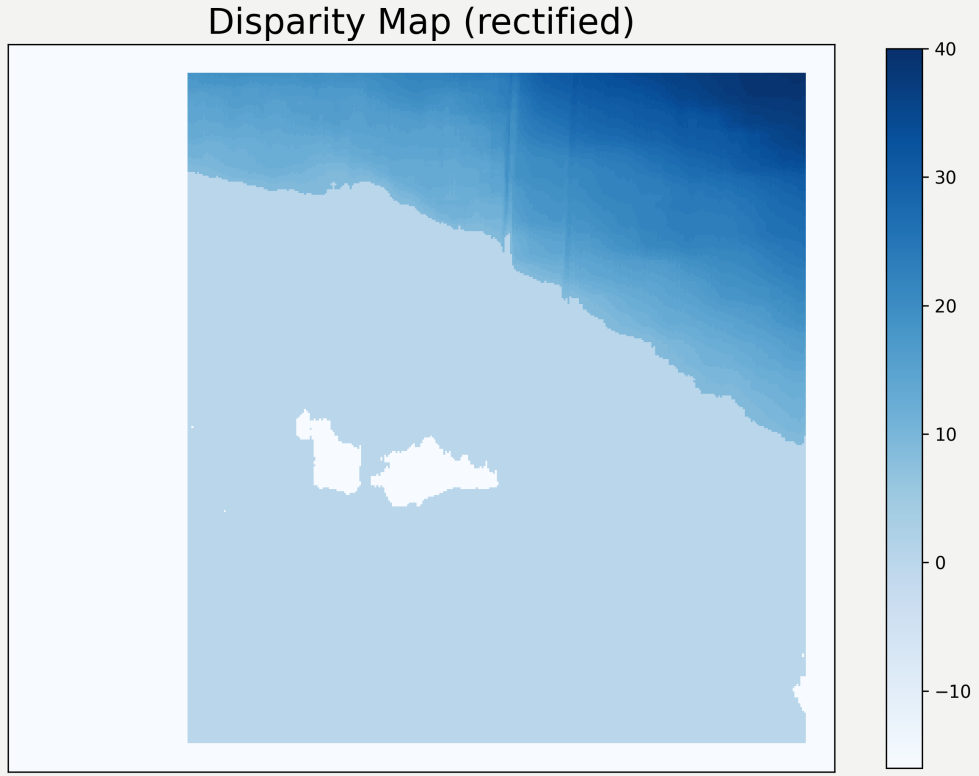
Rectify image. We can see that the epilines are someone horizontal which means that rectifying our image using cv2.stereoRectifyUncalibrated would not change the perspective of our images that much.

We see that the rectification is only small (emphasized by the red circle).

GETTING DISPARITY MAP

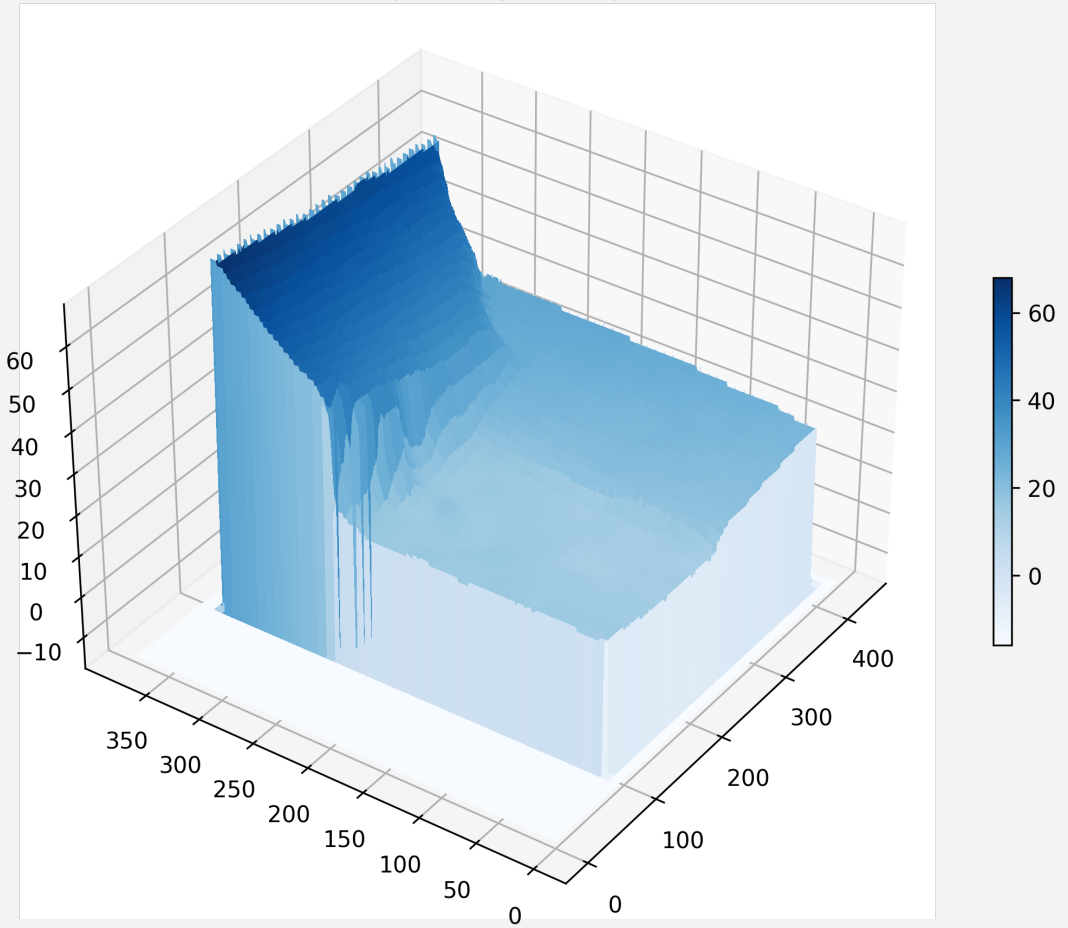
The goal of stereometry is to measure the depth from two images. So getting the disparity map would give us a view of that. To get the disparity map using cv2 in Python, we need to create a function stereo = cv2.StereoBM_create(numDisparities, blockSize) then use that to compute the disparity of the two images. The parameters just alters the accuracy of our calculations. Higher value of numDisparities means more accurate, but slower, solution and increasing blockSize (must be odd) increases the linear size of our blocks. Higher gradient or higher value of disparity means that the object is closer



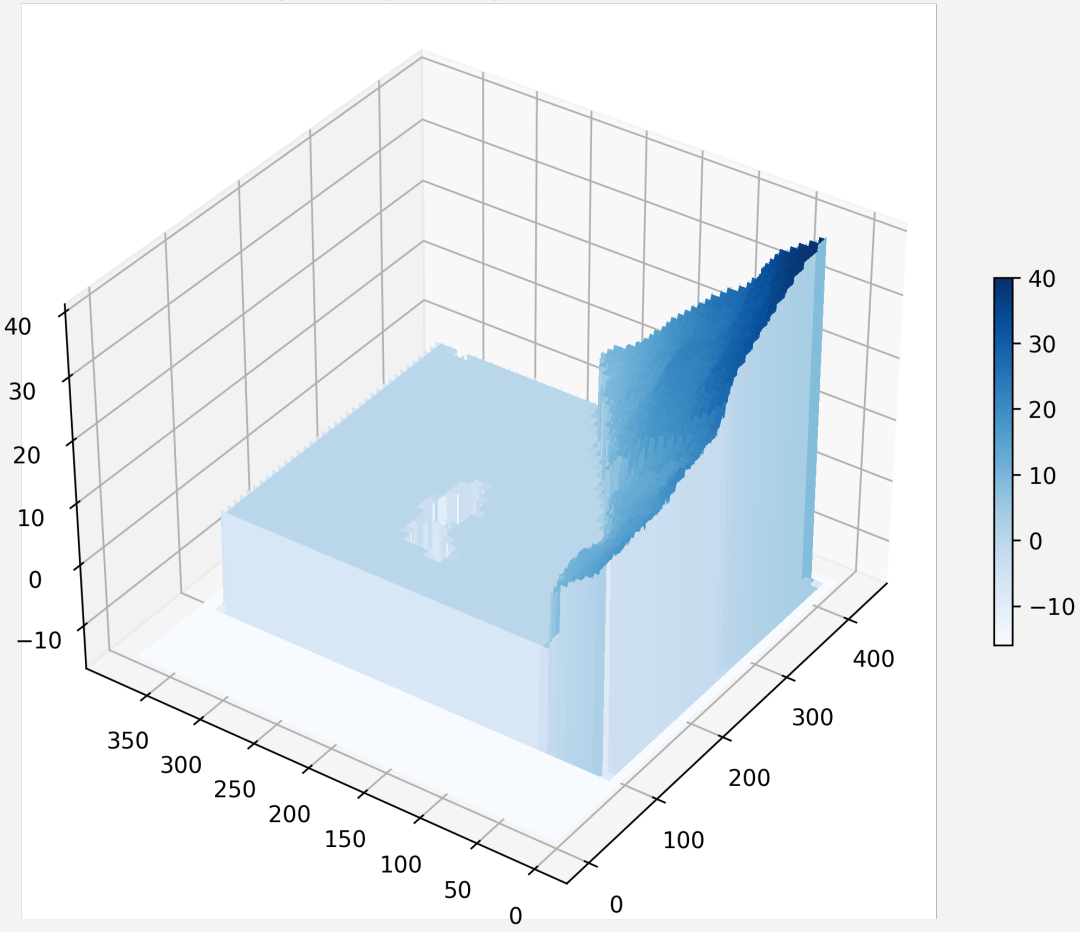


DISPARITY MAP (3D)

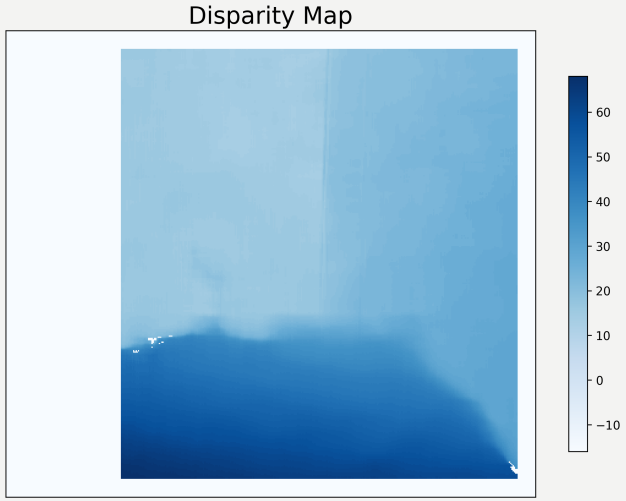
Disparity map

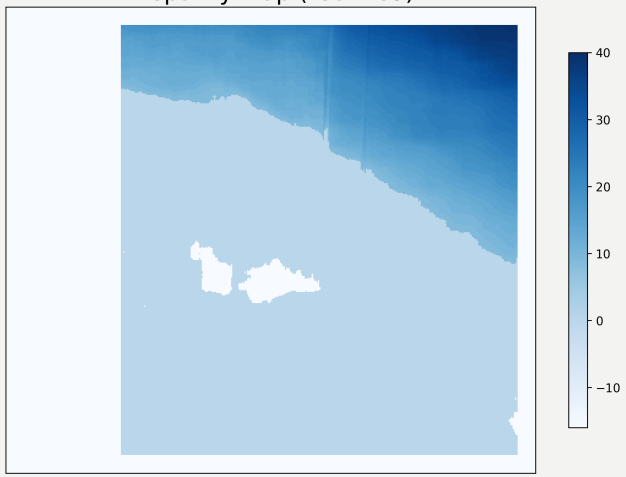


Disparity Map (rectified)



MEASURING DEPTH FROM DISPARITY MAP





Computing the depth is easy if we have the disparity map. We just need to use the formula

$$Z = \frac{bf}{x_1 - x_2}$$

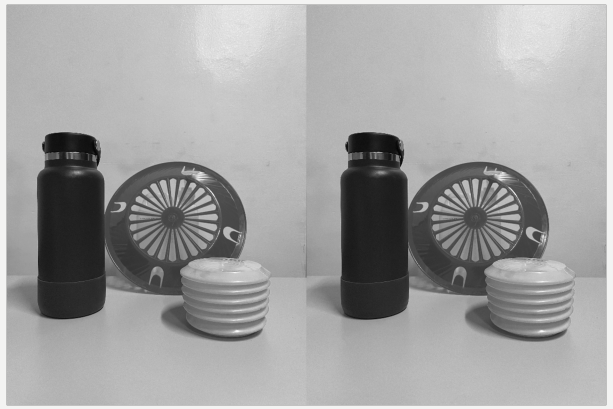
where b represents the distance between the two cameras, f is the focal length, and x1 – x2 represents the disparity map.

Sadly, from this data, we do not know the distance between the two cameras and the focal length so we stop here. But, we will try to solve the depth using actual pictures using my phone.

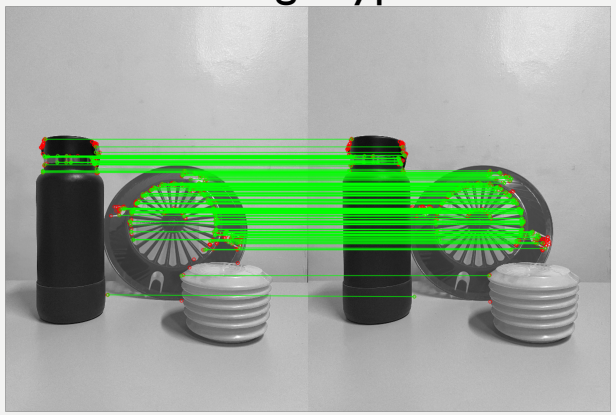
USING PHONE CAMERA

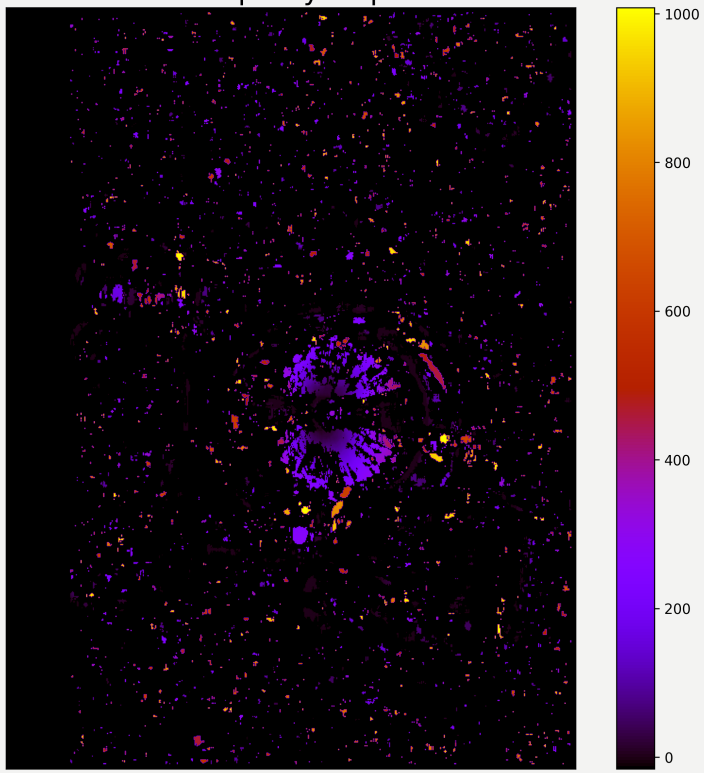
Let us look at some of the images I took using my iphone camera (f = 2.6 mm) with b = 1 cm.

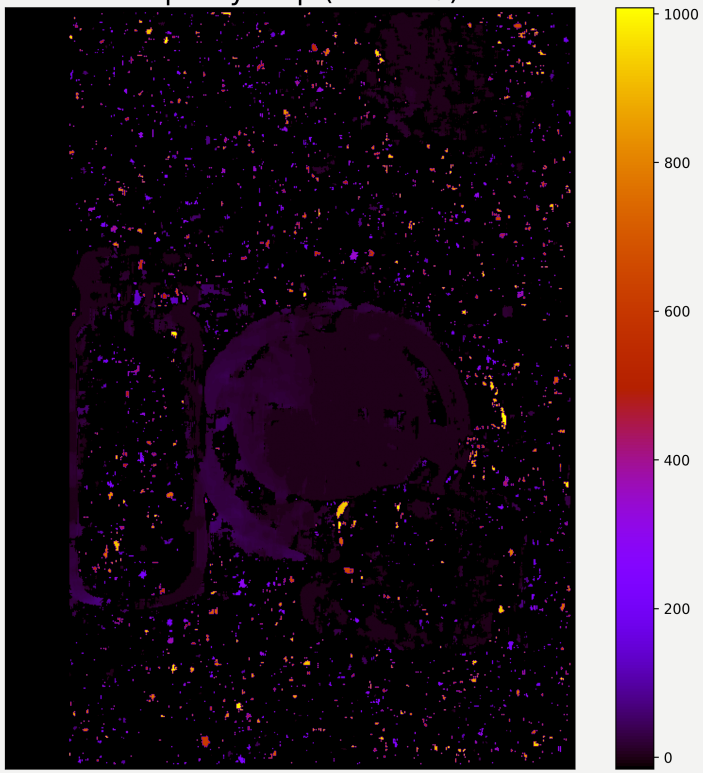
Grayscale images



Matching keypoints



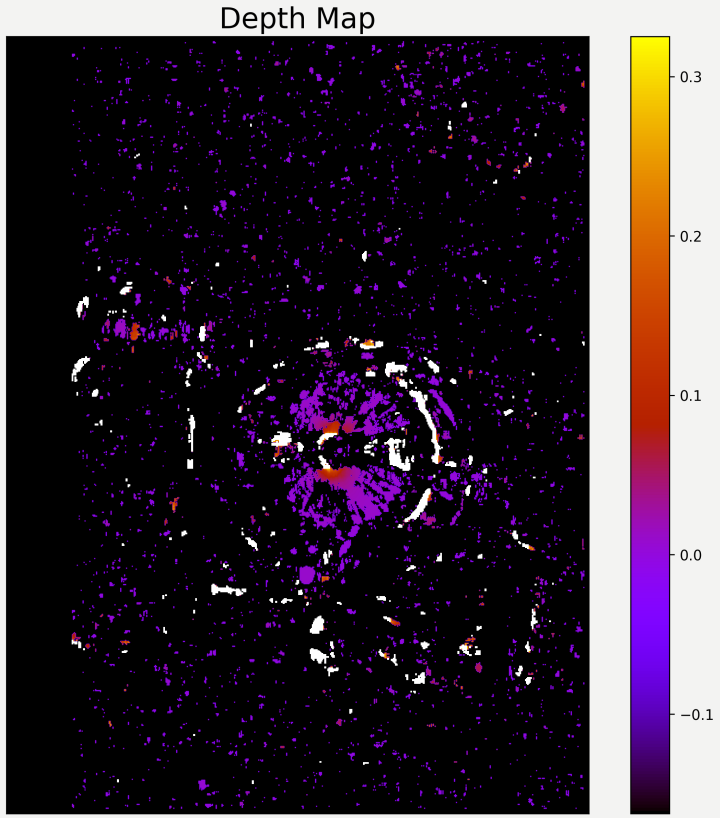


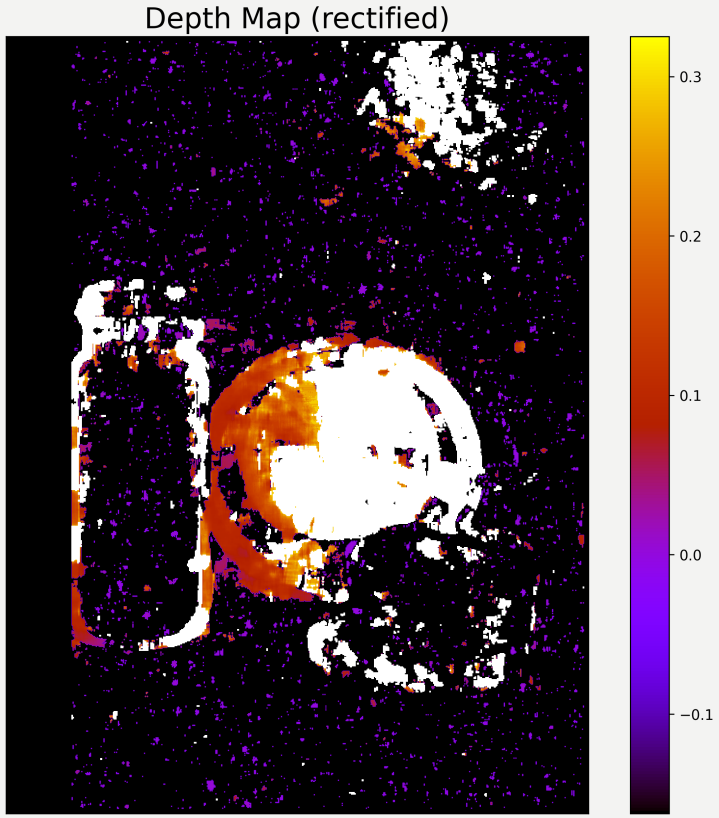


We see from this disparity maps the effect of rectifying our images. Rectifying allows us to see the separation of the objects (foreground) and the background which creates depth.

USING PHONE CAMERA

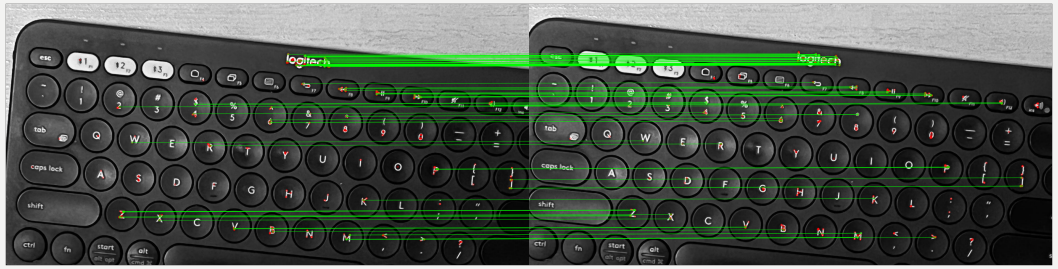
Computing for the depth, we get the following plots:

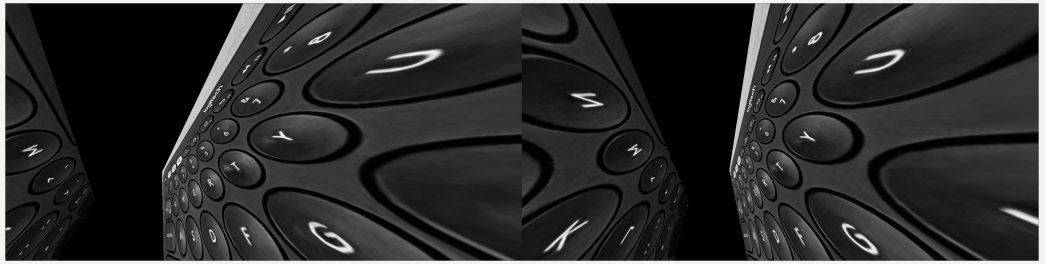


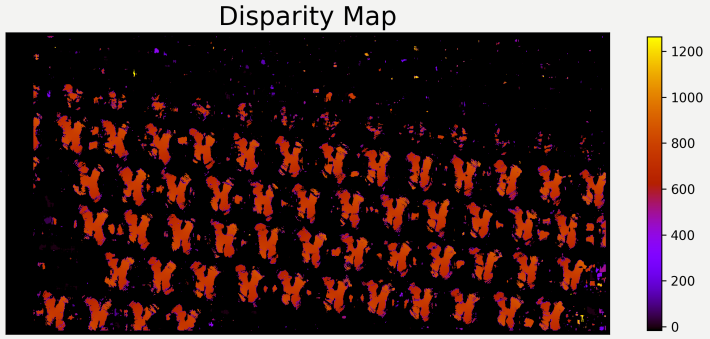


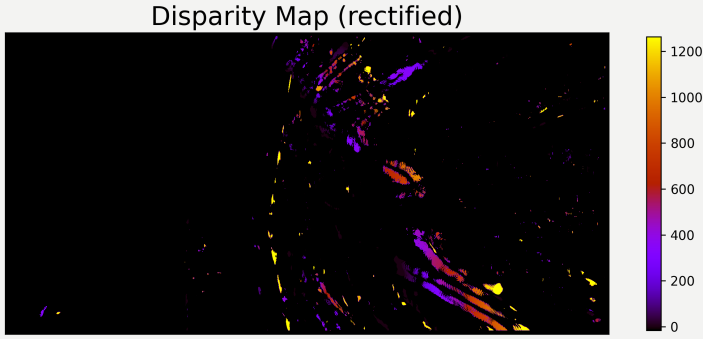
The colorbar shows that the objects are approximately 0.1 cm away from the camera which is very far from the actual value. I think one needs to calibrate the camera first before directly applying the formula to get the proper depth

OTHER EXAMPLES





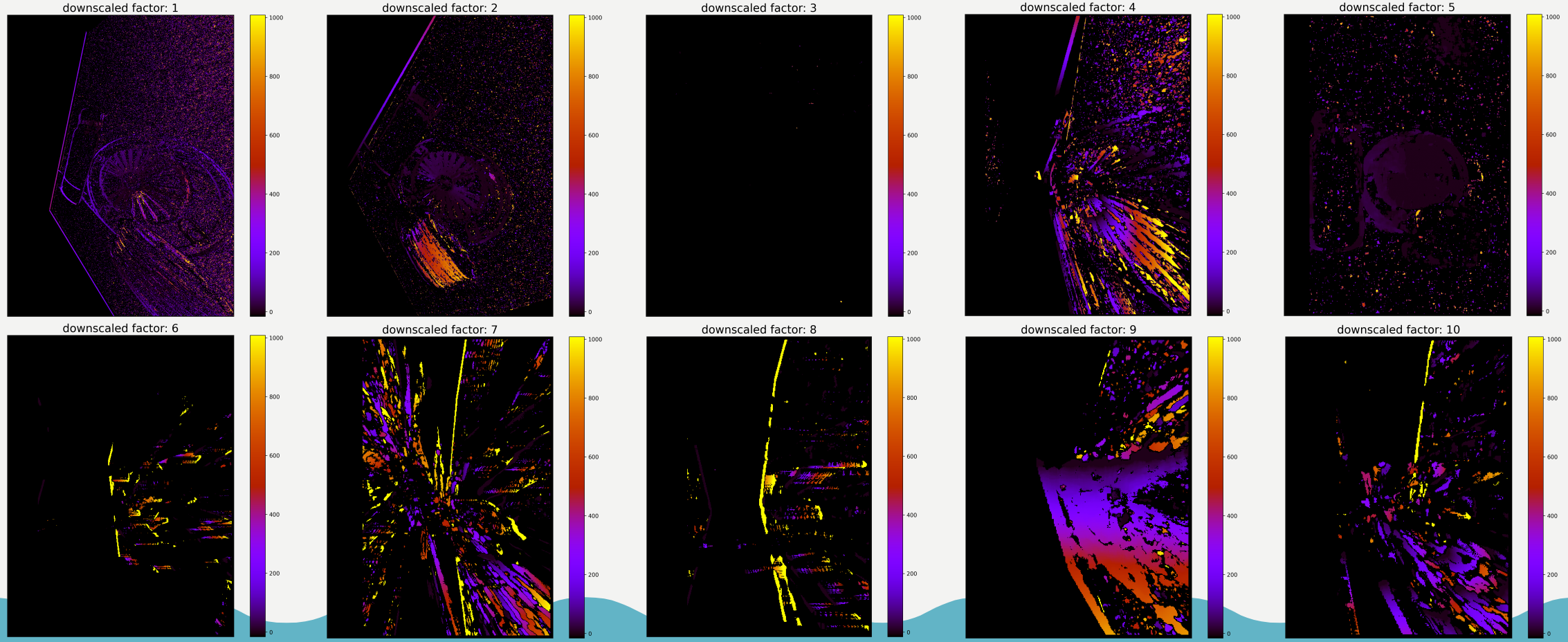




We notice from this example that using the original images produces a better disparity map compared to the rectified ones.

EFFECT OF DOWNSCALING

One possible reason why we got a very different from the effect of downscaling. What I did in the last two examples was I downscaled the size of my images in order to speed up the process. But as it turns out (from checking the effect on the disparity map) that there are different allowable downscaling for different images, so you need to try it out first before downscaling.



SUMMARY PLUS REFERENCES

Summary

I had fun here but at the same time had a hard time because most of the functions used here are relatively new so I had to read about them first before coding it. But it was nice and satisfying to figure out how to perform the stereometry. One thing that I wish I could've explored further is the calibration of the camera which can affect the brightness of my image (can have an effect) and make sure the photos were taken under constant illumination.

Score:

Technical Correctness – 28
Quality of Presentation – 30
Reflection – 30
Ownership – 10

Total – 98/100

References

[1] https://vision.middlebury.edu/stereo/data/scenes2001/
[2] parts of my code are taken from: https://www.andreasjakl.com/understand-and-apply-stereo-rectification-for-depth-maps-part-2/#zp-ID-2272-3683967-J2UT5KNX