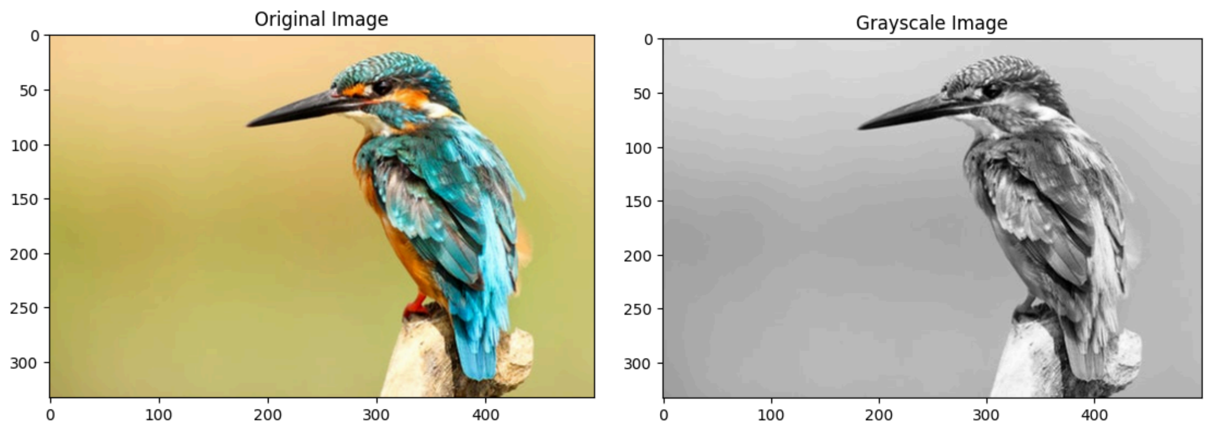
Computer Vision

Assignment 1

Siddharth Soni (B20AI041)

Question 1: Harris Corner Detection

When we load an image, it has shape 3 (height, width, channels). Depending on the number of channels, we decide whether the image is grayscaled (1 channel), RGB (3 channels) or RGBA (4 channels).



1. Convert image to grayscale:
    a. Grayscale is single channel scale which contains only intensity information.
    b. Harris corner detection only requires Intensity information.
    c. Issues with RGB:
        i. Since, RGB channels contains color intensities. This additional information is irrelevant.
        ii. and might add color sensitivity to the algorithm making it inaccurate.
        iii. Since, grayscale is color-invariant, therefore it is more robust.
        iv. Also, rgb channels will require equal computations per channel which implies more complexity for algorithm.

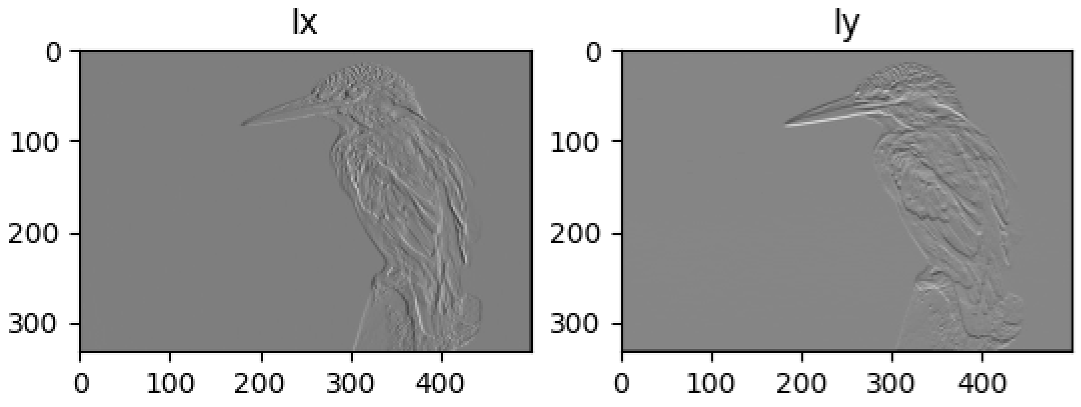
grayscale pixel is weighted sum of R,G,B pixels

Grayscale_pixel = w_R * c_R + w_G * c_G + w_B * c_B

OpenCV's conversion formula: RGB[A] to Gray :

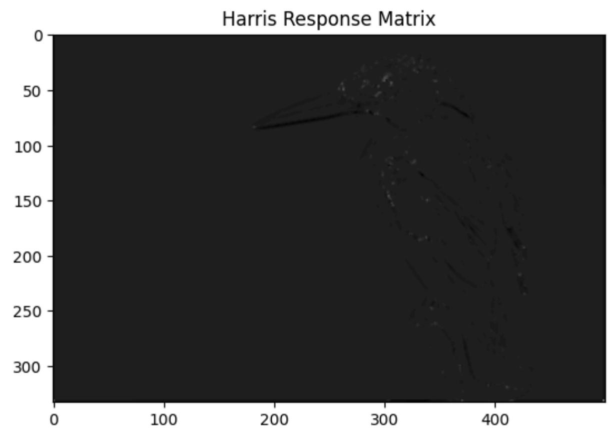
Y ← 0.299*R + 0.587*G + 0.114*B

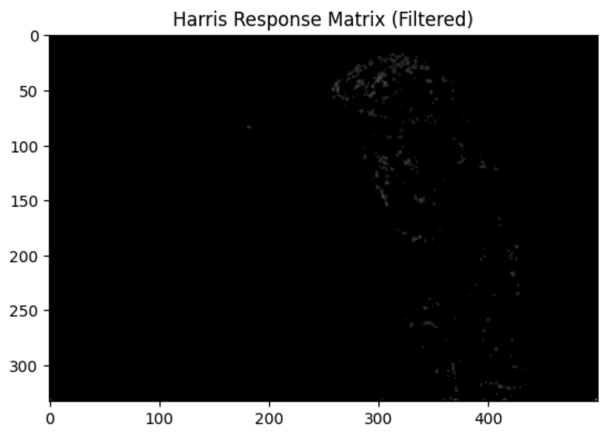
2. Compute Image gradients (Ix and Iy):
    a. x gradient filter = [-1, 0, 1]
    b. y gradient filter is transpose of x gradient filter
    c. convolution of image with respective filters gives gradients along respective axis.



3. Harris corner response:
    a. element -wise multiplication: Ix * Ix, Ix * Iy, Iy * Iy gives pixel-wise values.
    b. Convolving sum kernel to get sum of squared values Sxx, Sxy and Syy for a patch-size will help to calculate corner response for each pixel patch.
    c. M_ij = [[Sxx_ij, Sxy_ij], [Sxy_ij, Syy_ij]] is response matrix for every pixel i, j.
    d. Response = det(M_ij) - k * Tr(M_ij)^2

Using patch size (3, 3) and k = 0.04, we get response matrix for every pixel:

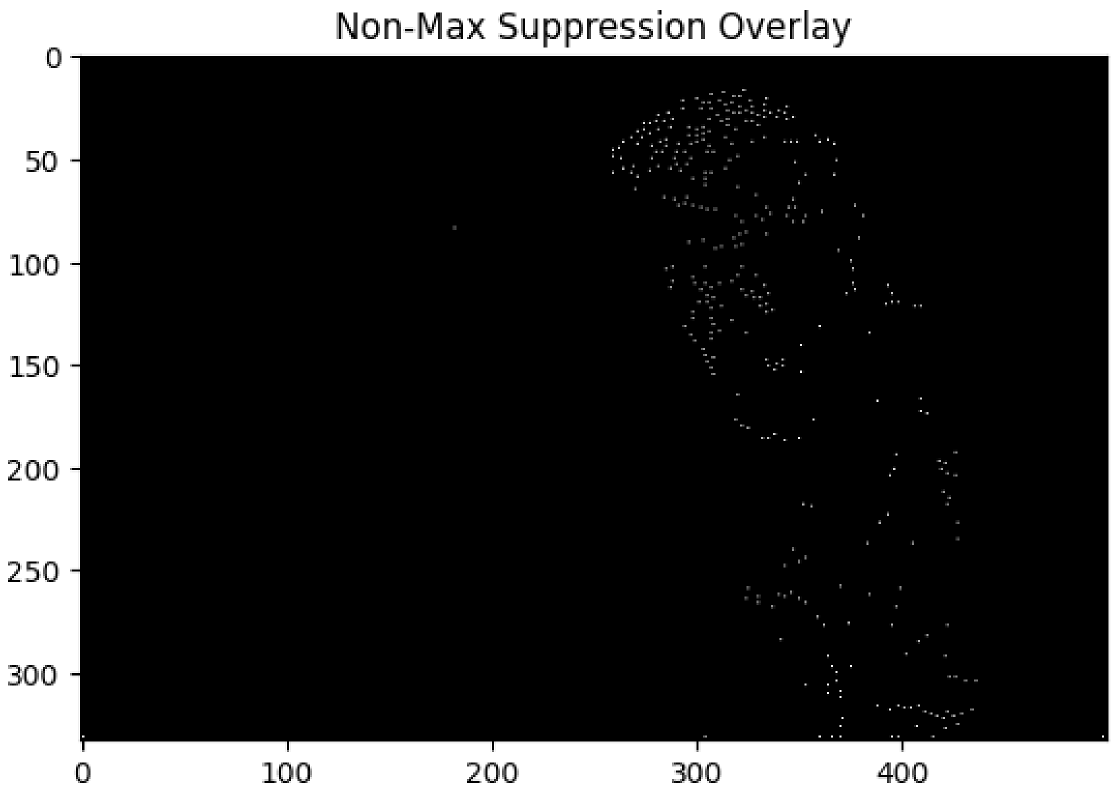




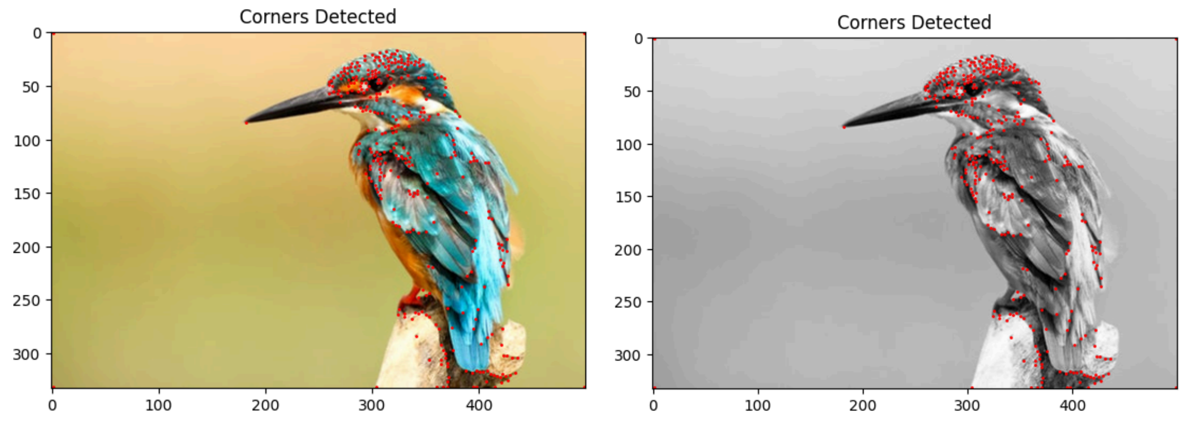
4. Filtering response matrix by threshold:
    a. Any response above certain threshold is a proposed corner.
    b. Threshold = 0.13 (this is applied on min-max normalized response, because min-max normalized response belongs 0 to 1 and is easier to assign threshold.)

5. Non-Max suppression:
    a. Since, there are many pixels in close proximity to indicate same corner.
    b. We sort the corners by descending value of response, then for every strong corner, we find weak corners in its proximity and suppress them.

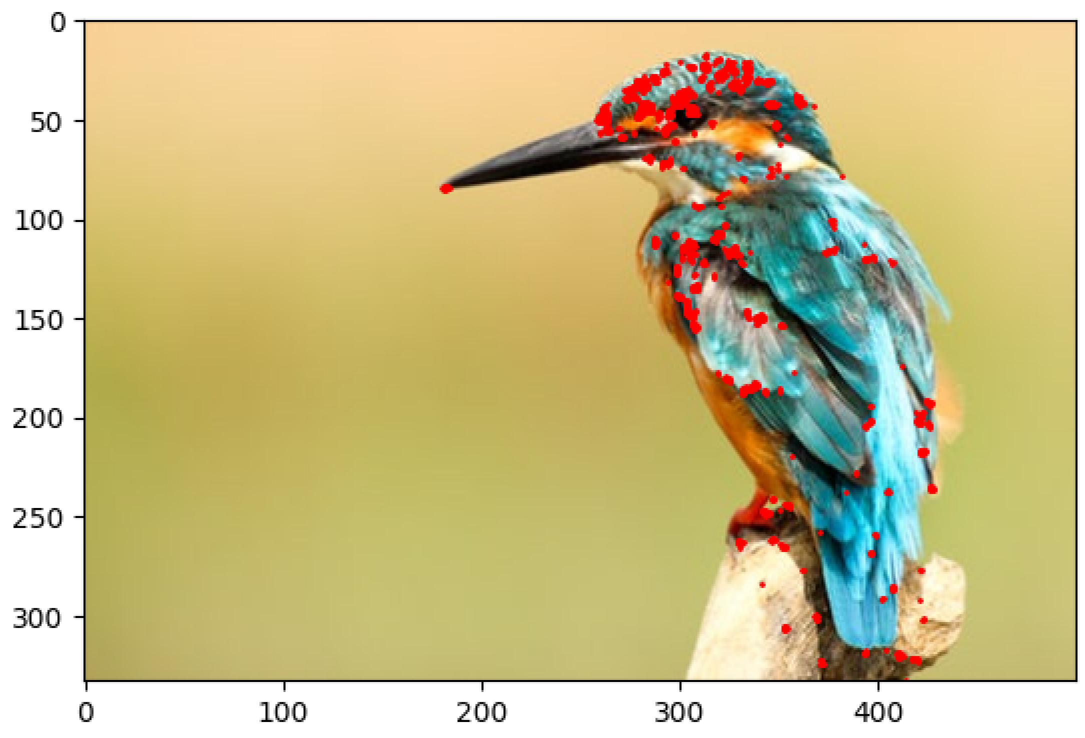
For patch size of (5, 5), the final corners are:



Corners Detected from scratch:



Comparison to OpenCV:



Observation

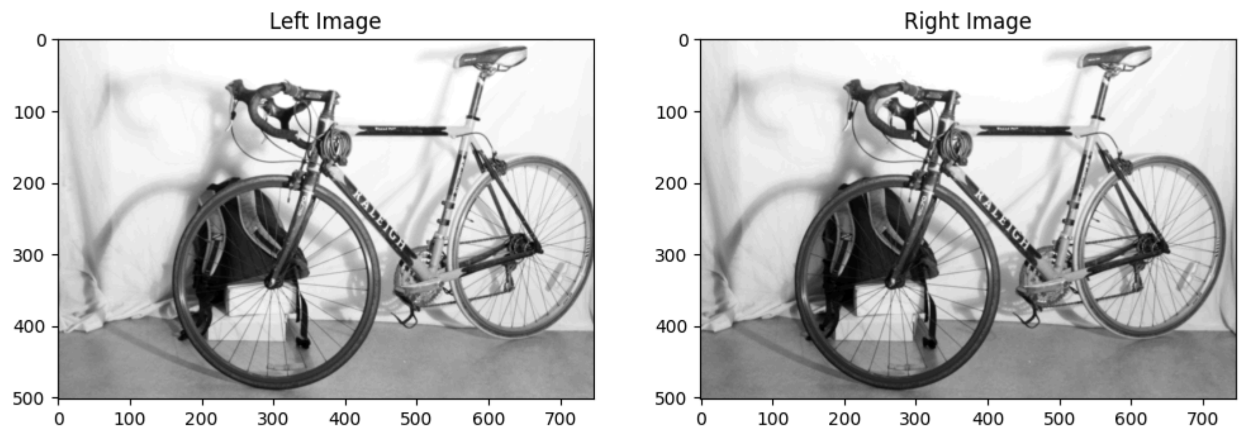
OpenCV uses 2D sobel filters for partial derivative calculations whereas we use 1D Sobel derivative filters. This only smoothes images and therefore gradients, which removes false corners from noise but also dilates the original corner over more pixels.

Another difference is that openCV uses gaussian filter instead of sum kernel for computing Sxx, Syy, Sxy. This only affects the weighted sum.

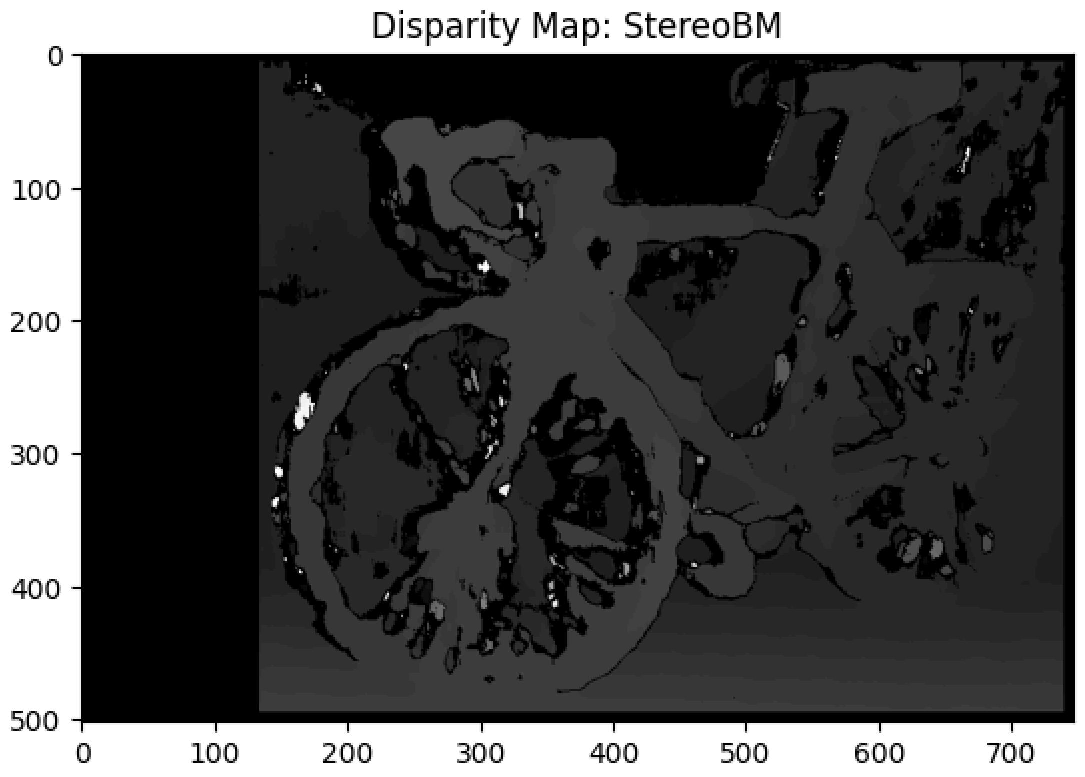
Effectively, both methods perform same operations. Only difference lies in application of gaussian smoothing at places.

Question 2: Simple Stereo 3D reconstruction

For simple stereo problem, we load the left and right images and then convert them to grayscale for disparity matrix calculation.

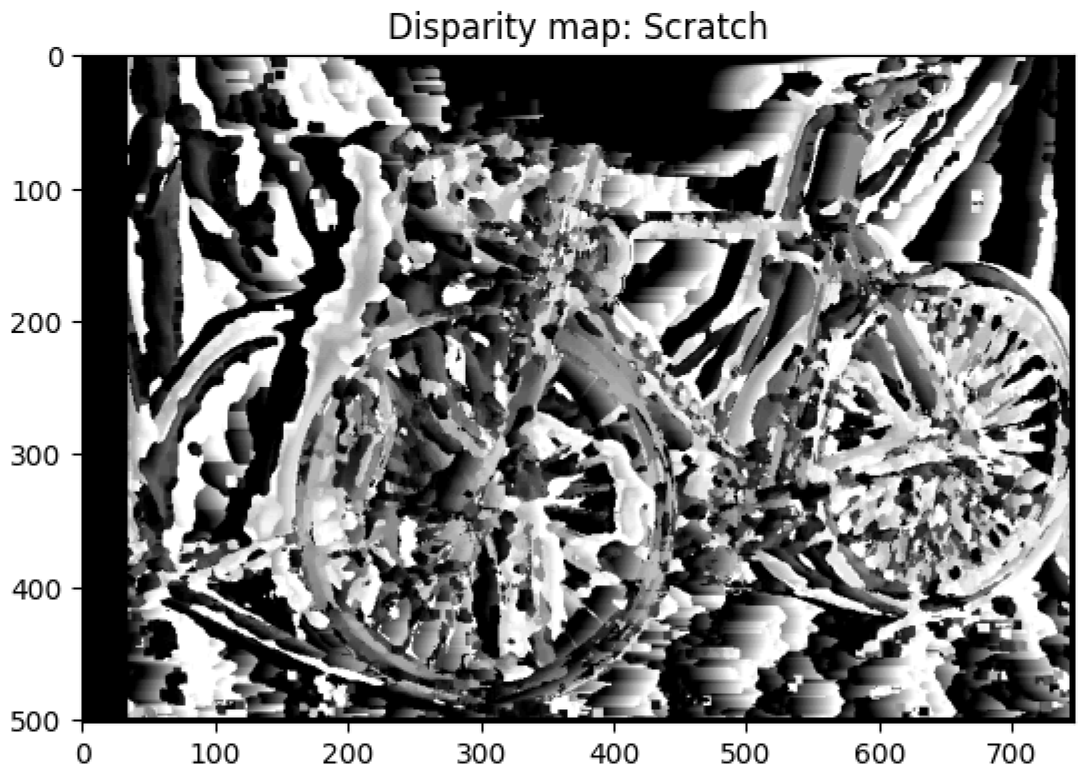


Using openCV's stereobm function to find disparity using block matching approach, we use numdisparities = 16*3 and blocksize=15, this gives us:



For computing from scratch, we need to scan the horizontal line for every pixel and match the pixel patch that has least sum of absolute difference as corresponding pixel in right image. Since, image dimensions are ~(2000, 3000). The number of pixels are exponential 6M and scanning over atleast 50 pixels disparity gives 10^7 computations which is veery large for a patch of any size. Therefore we downsample the image twice to the shape (500, 750). This makes scratch computations easier.

For same parameters, the scratch implementation results:



For 3D reconstruction point cloud, we are given Kl and Kr (calibration matrices), According to OpenCV, matrix Q (disparity to depth projective matrix) is formulated as:

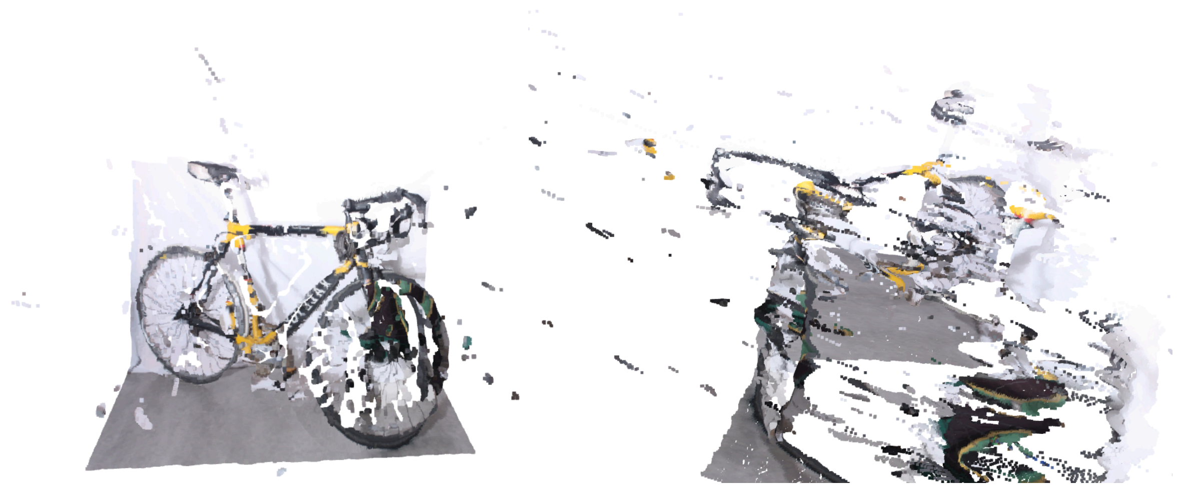
$$\mathbf{P1} = \begin{bmatrix} f & 0 & cx_1 & 0 \\ 0 & f & cy & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

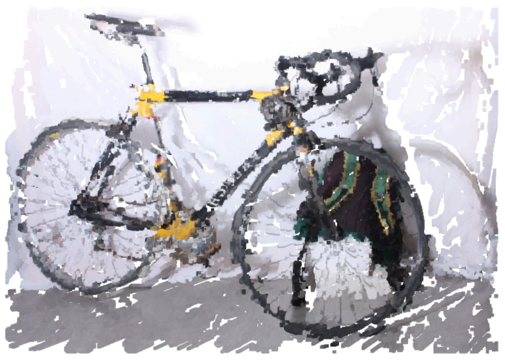
$$\mathbf{P2} = \begin{bmatrix} f & 0 & cx_2 & T_x \cdot f \\ 0 & f & cy & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix},$$

$$\mathtt{Q} = \begin{bmatrix} 1 & 0 & 0 & -cx_1 \\ 0 & 1 & 0 & -cy \\ 0 & 0 & 0 & f \\ 0 & 0 & -\frac{1}{T_x} & \frac{cx_1 - cx_2}{T_x} \end{bmatrix}$$

Where P1 = Kl, P2 = Kr, therefore we can easily compute projective matrix. We project each disparity over Q to get depth values corresponding to every pixel. Filtering the non-zero disparity values, we get the x, y, z and r, g, b values for each pixel and this is the 3d point cloud of the image.

Result from stereobm:





: result from scratch

Observation

As, we can observe that the cycle’s 3d depth perspective is very clear and intuitive which shows the smooth gradient of the disparity maps from our calculations.

Question 3: Epipolar geometry

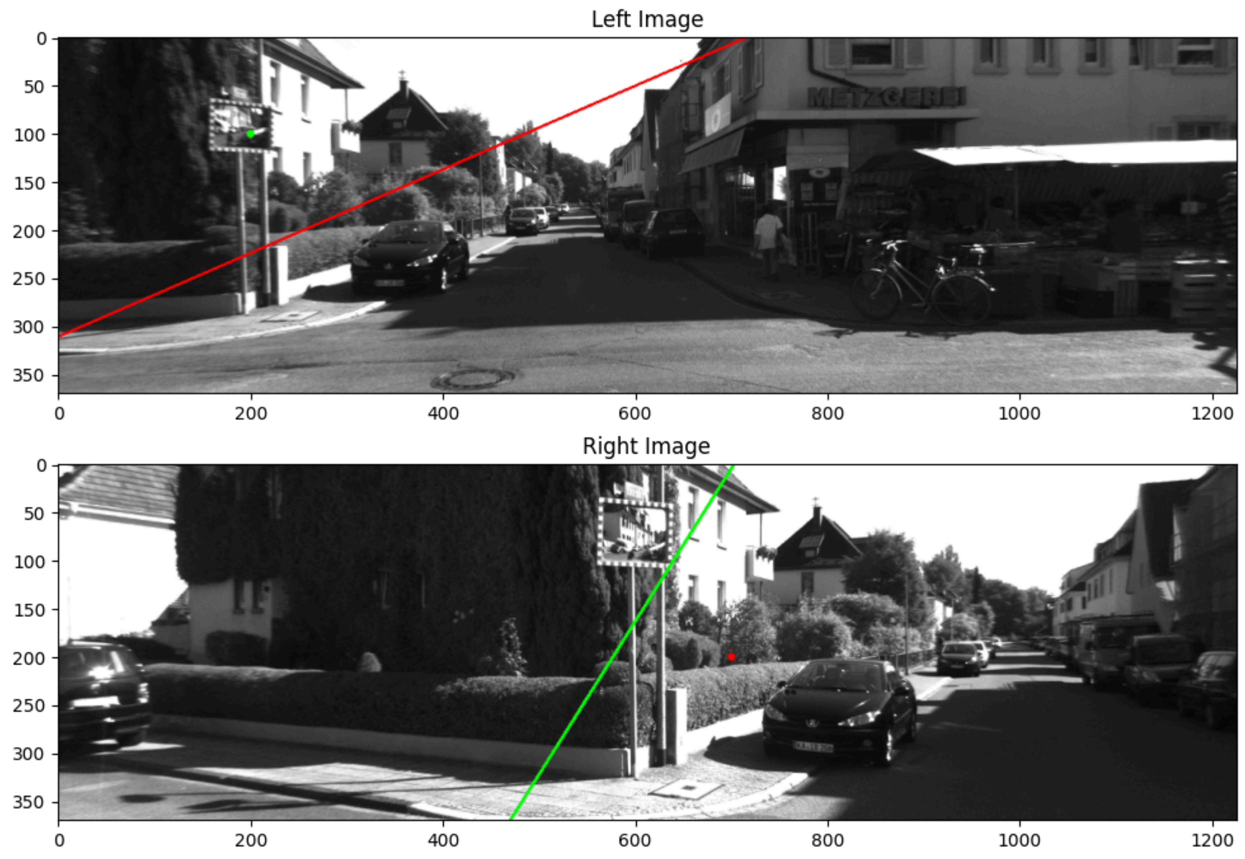
1. 1st part concerns with calculation of epipolar line from a left image pixel and a right image pixel.

Given the epipolar constraints as : ur.T * F * ul = 0

We can easily deduce that if we have right image pixel, we need to compute ur.T * F

And if we have left image pixel, we need to compute F * ul.

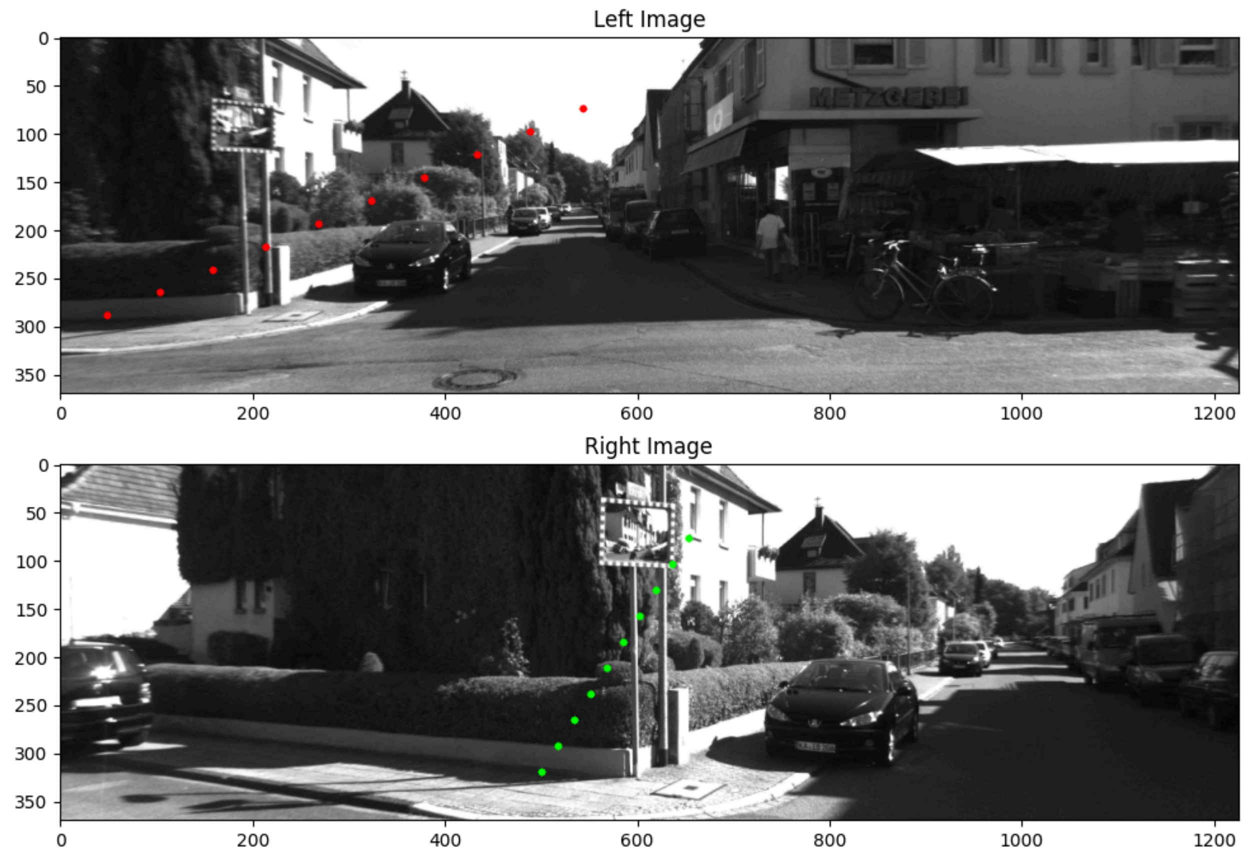
This will give us a,b,c constants for epipolar line ax+by+c=0 in respective cases.



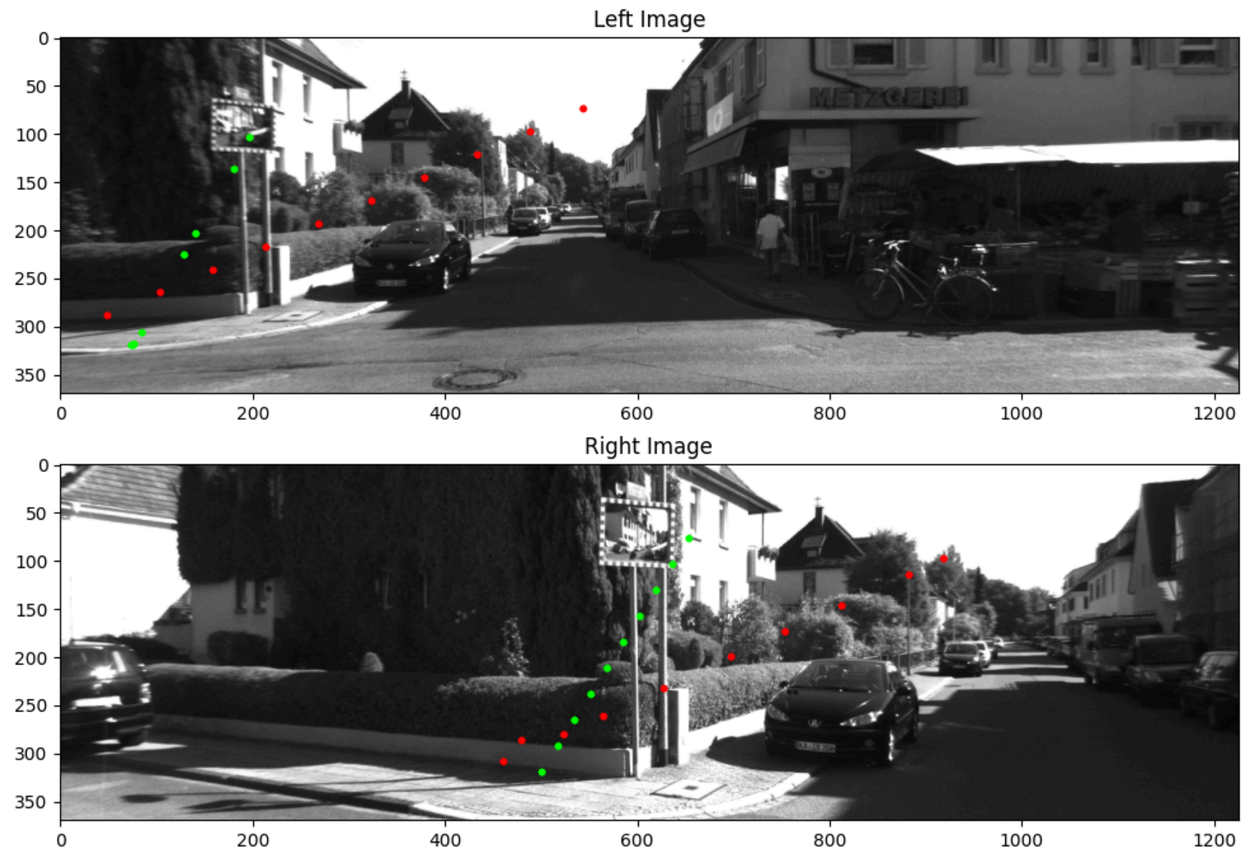
Green line in right image is epipolar line corresponding to green point in left image.

Similarly red line in left image is epipolar line corresponding to red point in right image.

2. Second part concerns us with sampling 10 uniformly distributed points on green and red lines respectively.
    a. For this, we calculate line intersection with image boundaries.
    b. Then, we calculate all pixels between these two line boundaries.
    c. Then based on the patch size of next part (block matching), we filter out the pixels whose patch boundary lies outside image boundaries.
    d. And then we take 10 equally spaced pixels from these filtered pixels.



3. Last comes the correspondence search using block matching algorithm,
    a. Given any point in image 1, we find its corresponding epipolar line in image 2.
    b. Then we find all pixel patches which lie on that epipolar line, because these are feasible for matching with our target pixel.
    c. Then we iterate over every such patch and return the patch with least sum of absolute difference values of intensity.
    d. We do this for every point on red line and green line and below are the correspondences found.



Green points on left image are best matches of 10 uniformly distributed point on green epipolar line in right image.

Similarly, red points on right image are best matches of 10 uniformly distributed points on red epipolar line in left image.

Observation

As we can see that the points obtained lie on the lie therefore, and this is due to the fact that the original pixels also lied on a epipolar line. Therefore, we can conclude that our computations and estimates were fairly accurate.