CV Assignment 2

Name 1: Abdelrahman Salem Mohamed
ID 1: 6309
Name 2: Reem Abdelhaleem
ID 2: 6114

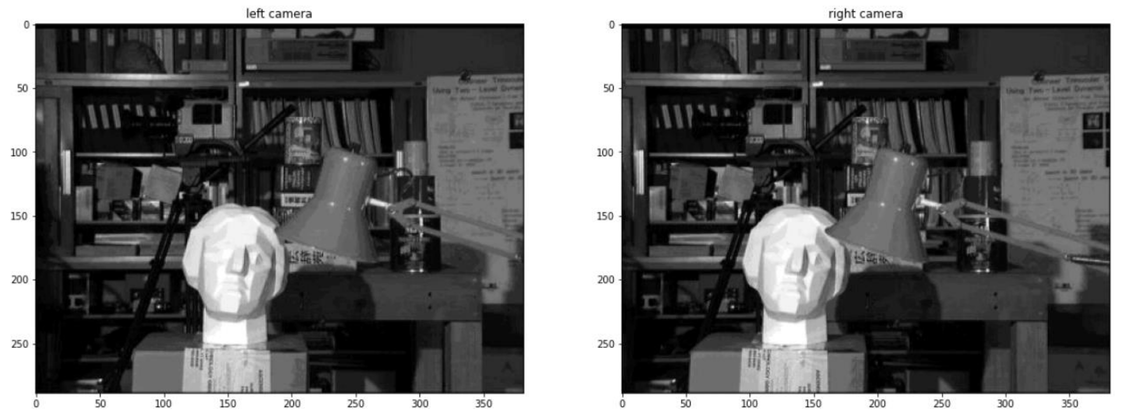
1 - Stereo Matching

In this part of the assignment, you will implement and test some simple stereo algorithms. In each case you will take two images Il and Ir (a left and a right image) and compute the horizontal disparity (i.e.., shift) of pixels along each scanline. This is the so-called baseline stereo case, where the images are taken with a forward-facing camera, and the translation between cameras are along the horizontal axis. We will calculate the disparity using two ways.

1.1 Block Matching

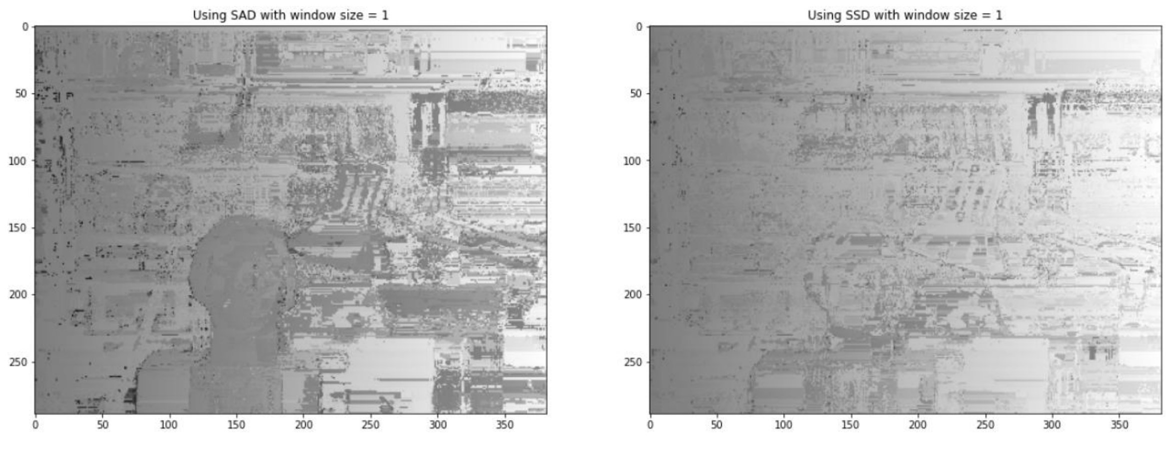
The first way to get the disparity is by matching each pixel in the left image to a pixel in the right image. Since there is no rectification needed, you will only need to match the row in the left image with its equivalent in the right image. you will calculate the disparity in two ways, once using the cost as the Sum of Absolute Differences (SAD) and another time using Sum of Squared Differences. Do this for windows of size w where w = 1, 5 and 9. You will produce 6 maps: 2 maps for each window size, once using SAD and the other using SSD.

First Image:

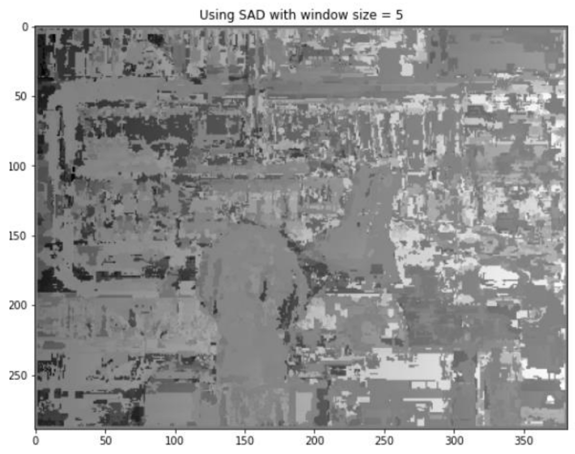


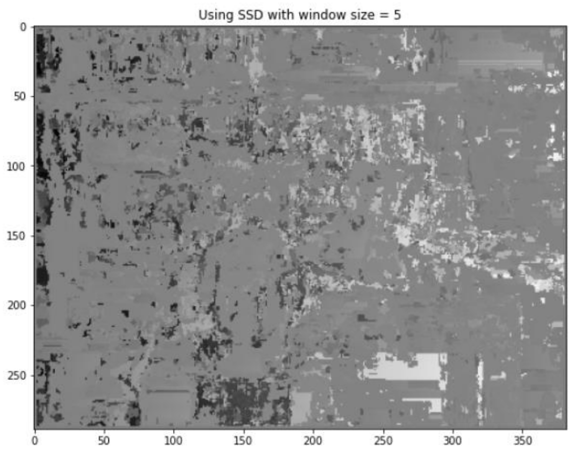
We assuming that the two scenes taken by the same camera (having same intrinsic matrix) And the shift is parallel to the line (Baseline)

A) k=1

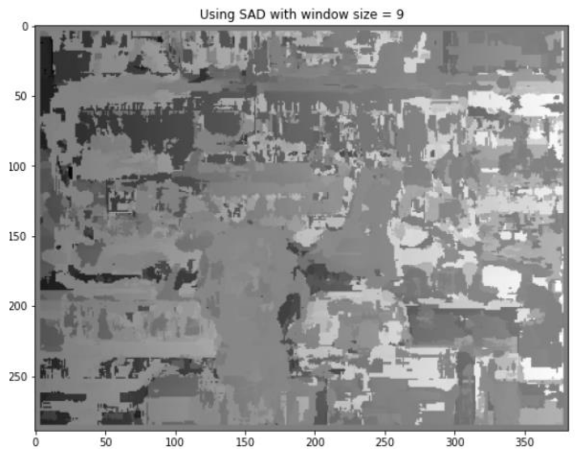


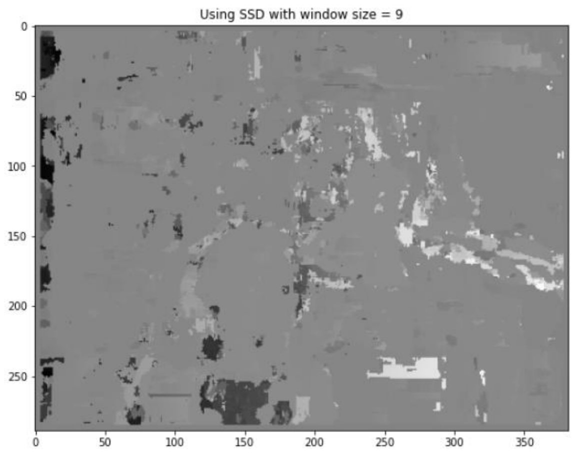
B) k=5



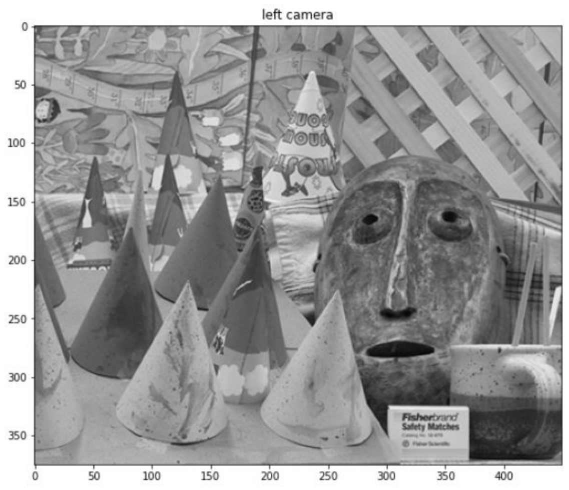


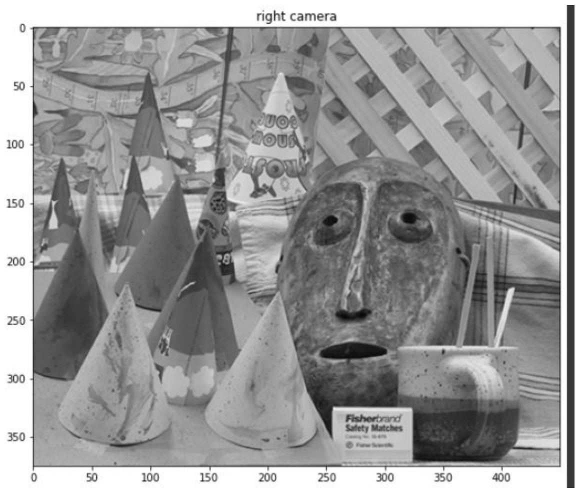
C) k=9



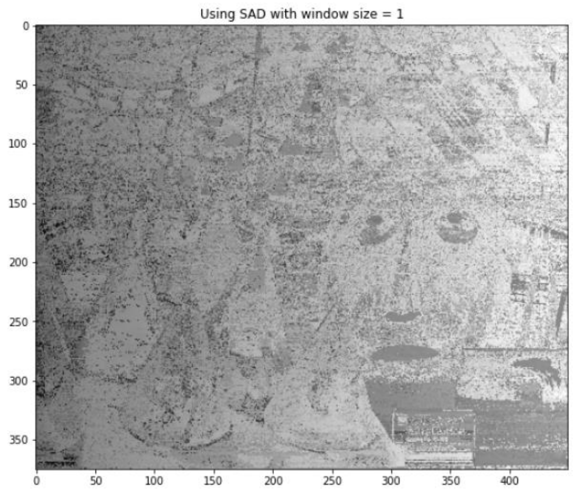


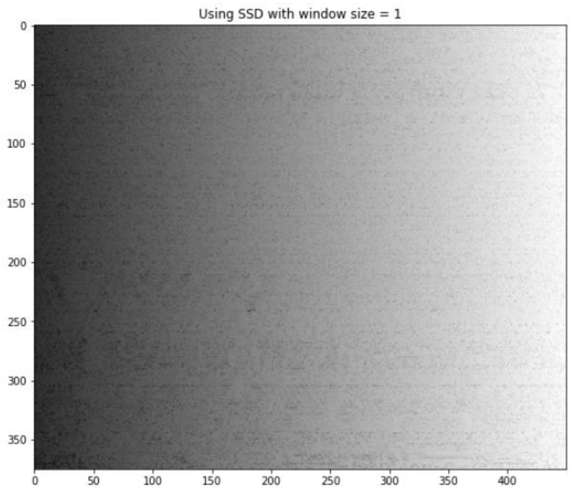
Second Image:



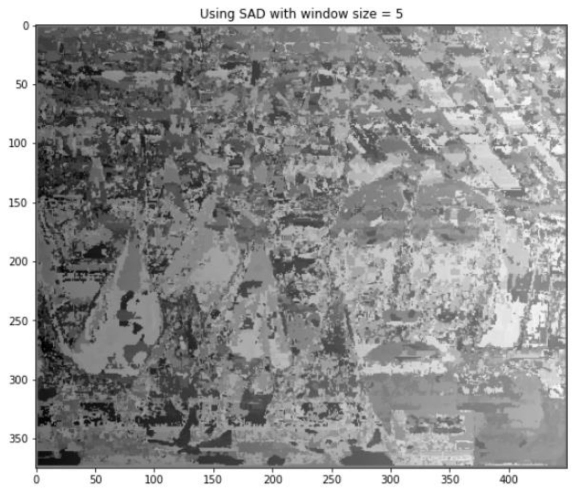


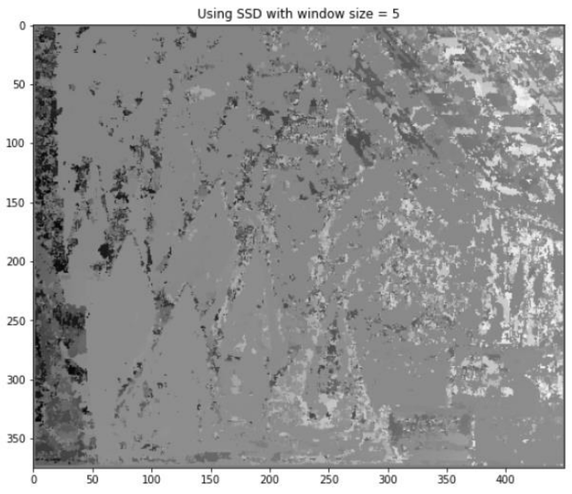
D) k=1


Using SAD with window size = 1

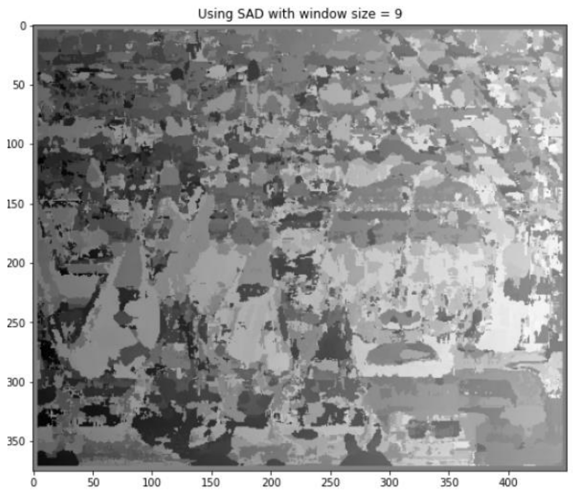

Using SSD with window size = 1

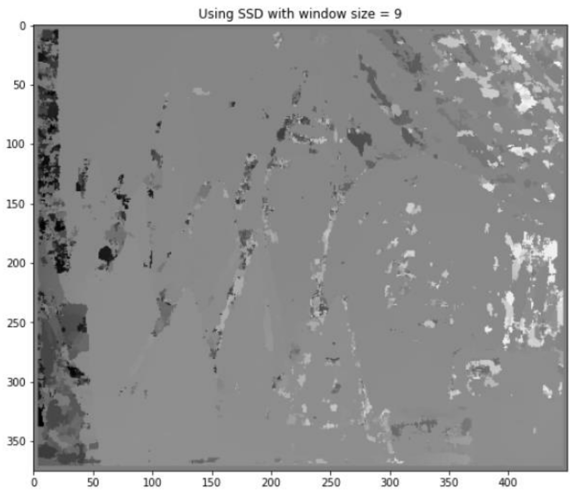
E) k=5


Using SAD with window size = 5

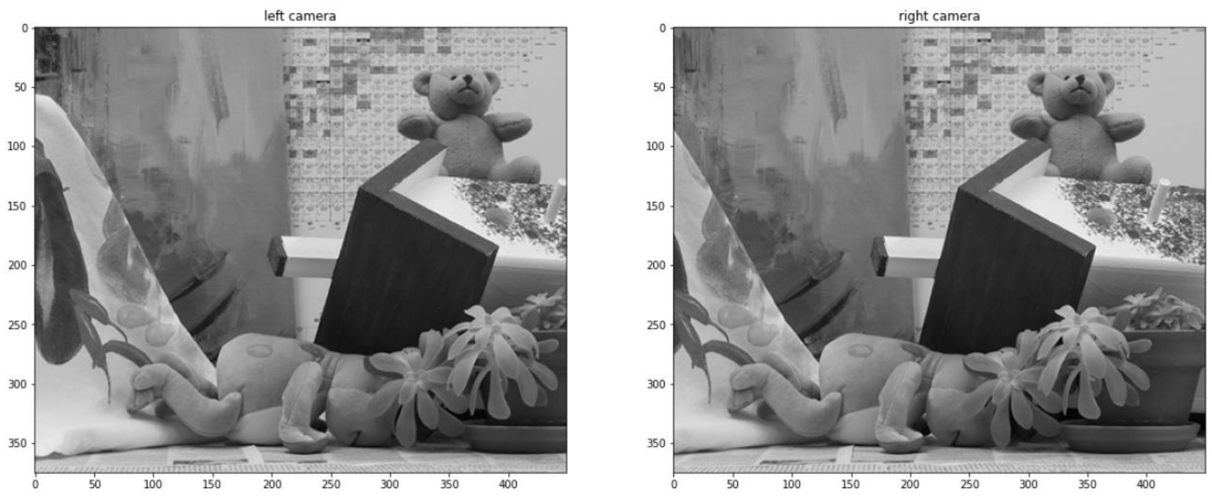

Using SSD with window size = 5

F) k=9

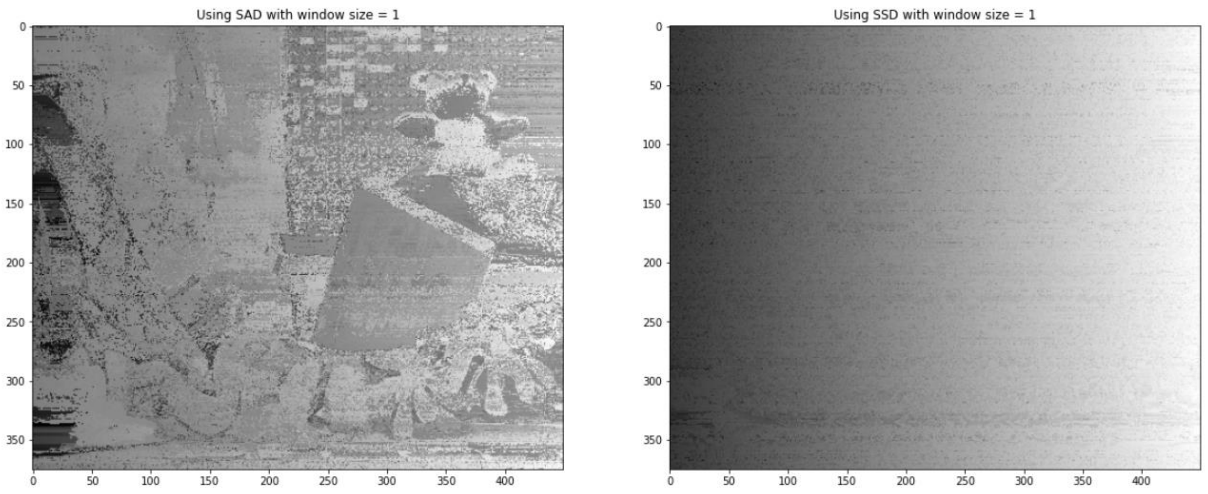

Using SAD with window size = 9


Using SSD with window size = 9

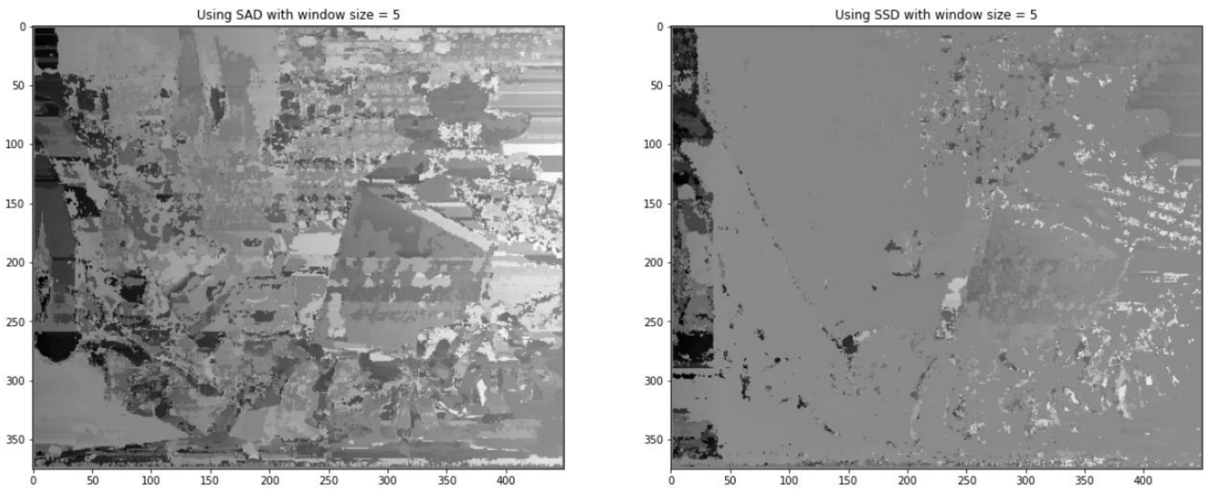
Third Image:



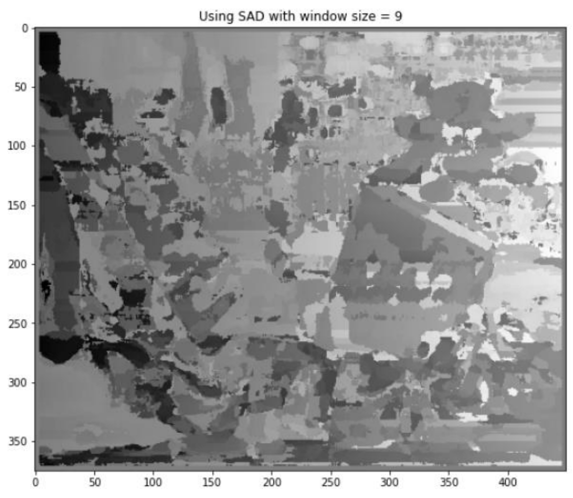
G) k=1

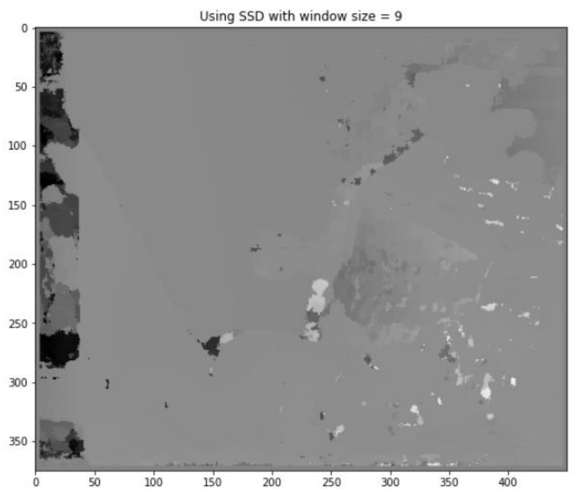


H) k=5



I) k=9





Note:

- It was observed that when the window size is small the disparity map is subjected to become more noisy but some details are clear. On the other hand, larger window size makes the disparity map immune to the noise but we lost some details instead especially at the edges.
- We were expecting that the Sum of Square difference will get better results, but the Sum of absolute difference produces better results. (Maybe not in our problem)
- As the relation between the disparity and the depth is inversely proportion, then the closer the objects are (depth is low) the more disparity.

Notebook

https://colab.research.google.com/drive/1pZDXPgZEDBxihHIZWP5_2iqrKxNzEX4B?usp=sharing

1.2 Dynamic programming

Consider two scanlines Il(i) and Ir(j). Pixels in each scanline may be matched or skipped (considered to be occluded in either the left or right image). Let dij be the cost associated with matching pixel Il(i) with pixel Ir(j). Here we consider a squared error measure between pixels given by:

$$d_{ij} = \frac{(I_l(i) - I_r(j))^2}{\sigma^2}$$

where σ is some measure of pixel noise. The cost of skipping a pixel (in either scanline) is given by a constant c0. For the experiments here we will use σ = 2 and c0 = 1. Given these costs, we can compute the optimal (minimal cost) alignment of two scanlines recursively as follows:

1. $D(1,1) = d_{11}$
2. $D(i,j) = min(D(i-1,j-1) + d_{ij}, D(i-1,j) + c_0, D(i,j-1) + c_0)$

The intermediate values are stored in an N-by-N matrix, D. The total cost of matching two scanlines is D(N, N). Note that this assumes the lines are matched at both ends (and hence have zero disparity there). This is a reasonable approximation provided the images are large relative to the disparity shift. Given D we find the optimal alignment by backtracking. In particular, starting at (i, j) = (N, N), we choose the minimum value of D from (i-1, j -1),(i - 1, j),(i, j - 1). Selecting (i - 1, j) corresponds to skipping a pixel in Il (a unit increase in disparity), while selecting (i, j - 1) corresponds to skipping a pixel in Ir (a unit decrease in disparity). Selecting (i - 1, j - 1) matches pixels (i, j), and therefore leaves disparity unchanged. Beginning with zero disparity, we can work backwards from (N, N), tallying the disparity until we reach (1, 1).
A good way to interpret your solution is to plot the alignment found for single scan line. To display the alignment plot a graph of Il (horizontal) vs Ir (vertical). Begin at D(N, N) and work backwards to find the best path. If a pixel in Il is skipped, draw a horizontal line. If a pixel in Ir is skipped, draw a vertical line. Otherwise, the pixels are matched, and you draw a diagonal line. The plot should end at (1, 1).
We repeat the process explained above for each row of the image and compute the disparity maps for image pairs.

Functions used:

```
def dynamic_programming(left_img, right_img, c0=1, sigma=2):
  rows = left_img.shape[0]
  N = left_img.shape[1]

  left_disparity_map = np.zeros(left_img.shape)
  right_disparity_map = np.zeros(left_img.shape)

  D_total = []
  for row in range(rows):
    D = np.zeros((N,N))

    for i in range(N):
      for j in range(N):
        if i==0 and j==0:
          D[i][j] = ((left_img[row][0]-
right_img[row][0])**2)/(sigma**2)
        elif i==0 and j!=0:
          D[0][j] = D[0][0] + j*c0
        elif j==0 and i!=0:
          D[i][0] = D[0][0] + i*c0
        else:
          dij = ((left_img[row][i]-right_img[row][j])**2)/(sigma**2)
          D[i][j] = min(D[i-1][j-1]+dij, min(D[i-1][j]+c0, D[i][j-
1]+c0))

    k = N-1
    l = N-1
    right_disparity_map[row][j] = abs(k-l)
    left_disparity_map[row][j] = abs(k-l)
    while k!=0 and l!=0:
      if k-1>0 and l-1>0:
          dij_diagonal = D[k-1][l-1]
      if k-1>0:
          dij_above = D[k-1][l]
      if l-1>0:
          dij_before = D[k][l-1]

      dij_min = min(dij_diagonal,min(dij_above,dij_before))

      if dij_min == dij_diagonal:
        l = l-1
        k = k-1
        right_disparity_map[row][l] = abs(k-l)
        left_disparity_map[row][k] = abs(k-l)
      elif dij_min == dij_above:
        k = k-1
      else:
        l = l-1
    D_total.append(D)
  return D_total, left_disparity_map, right_disparity_map

def best_path_cal(D):
  N = D.shape[0]
  k = N-1
  l = N-1
  min_path_track = []
  min_path_track.append([k, l])
  while k!=0 and l!=0:
    if k-1>0 and l-1>0:
        dij_diagonal = D[k-1][l-1]
    if k-1>0:
        dij_above = D[k-1][l]
    if l-1>0:
        dij_before = D[k][l-1]

    temp = min(dij_diagonal,min(dij_above,dij_before))

    if temp == dij_diagonal:
      l = l-1
      k = k-1
      min_path_track.append([k, l])
    elif temp ==dij_above:
      k = k-1
      min_path_track.append([k, l])
    else:
      l = l-1
      min_path_track.append([k, l])

  best_path = np.array(min_path_track)
  return best_path
```

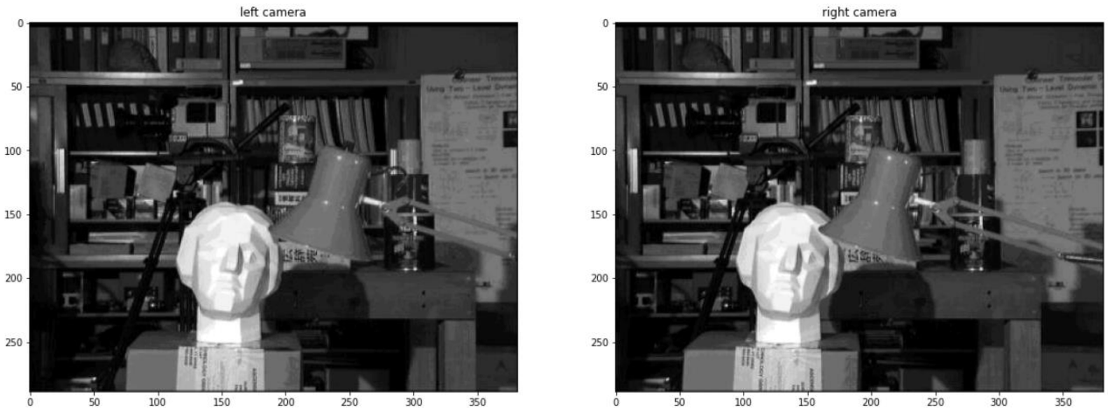
Explanation:

Dynamic programming function get 2 corresponding images, calculates D matrix (N,N) for each 2 corresponding rows in forward direction then in backward direction calculate left and right disparity matrix for each image.

Best_path_cal function get D matrix of any 2 corresponding rows from the corresponding images, by differentiate how each element calculation, from diagonal or above or before, then deciding which path will save diagonal or horizontal or vertical.

Outputs:

First sample image:



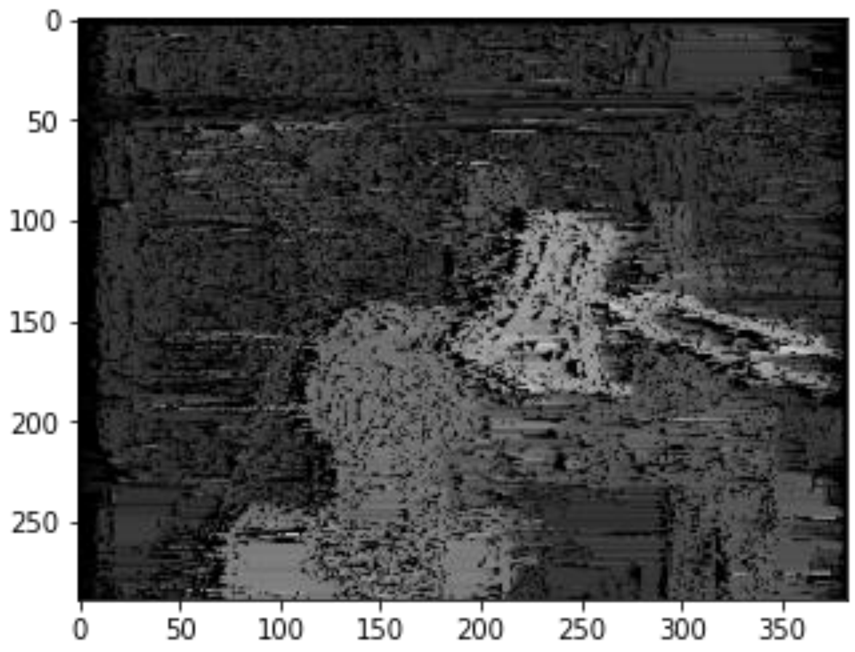


Figure 1left disparity map

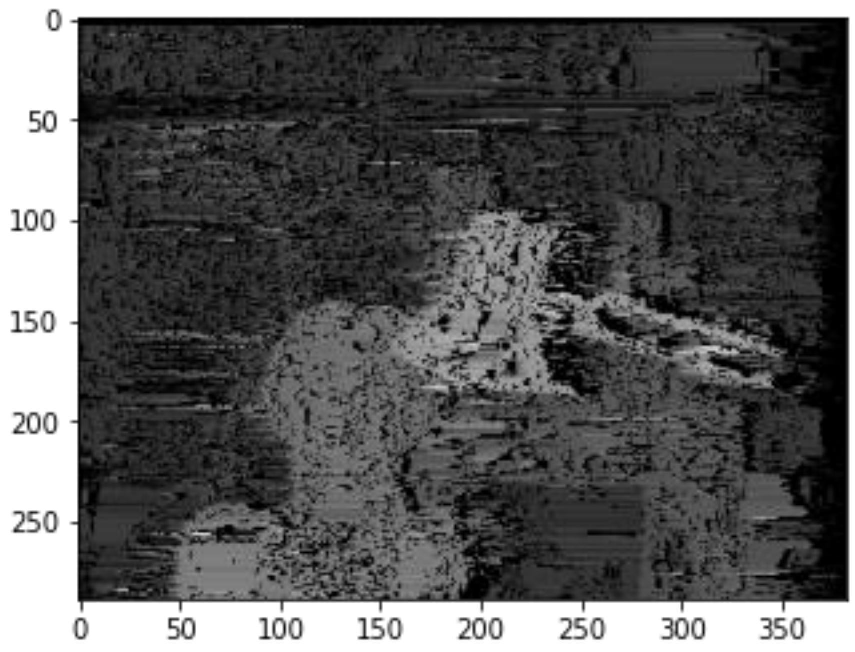
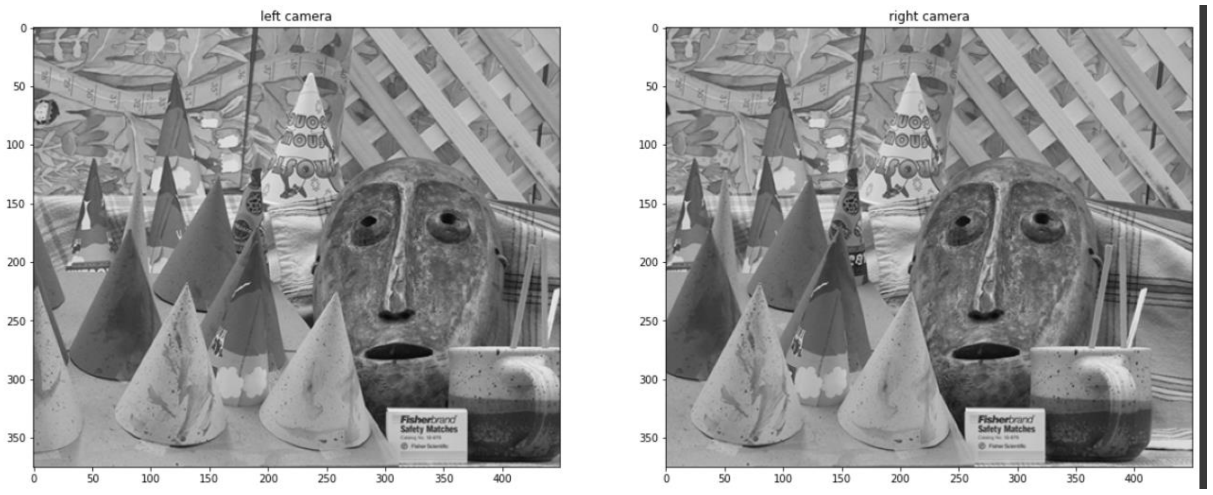
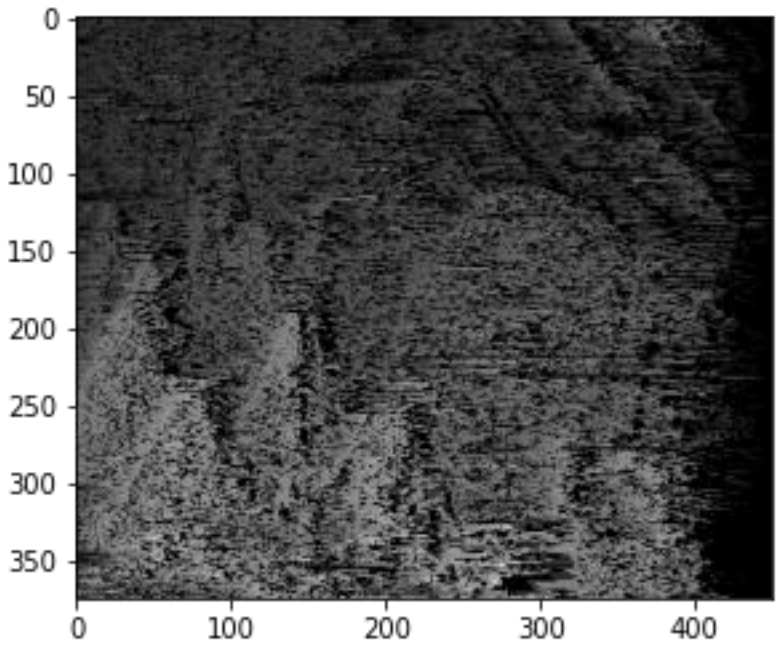


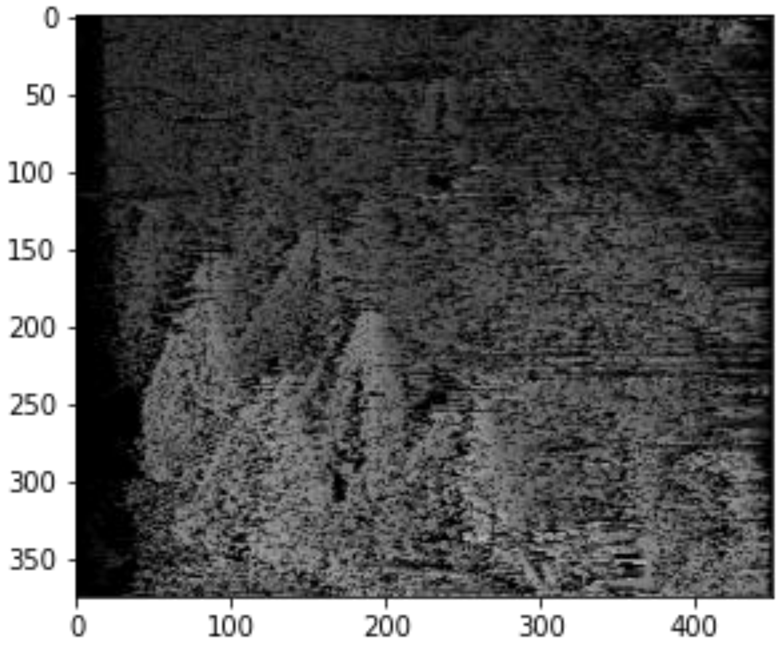
Figure 2right disparity map

Second sample image:



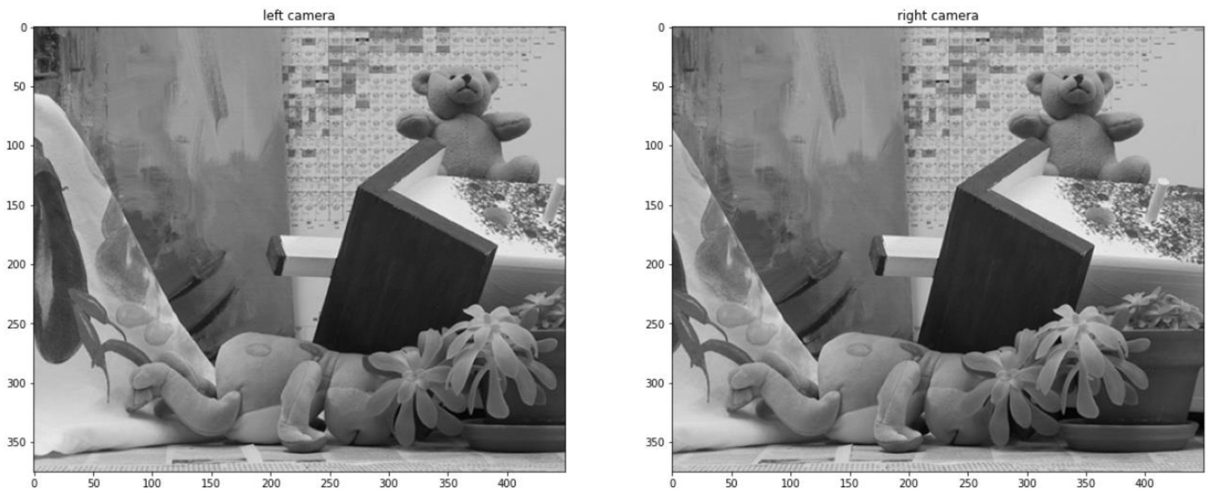


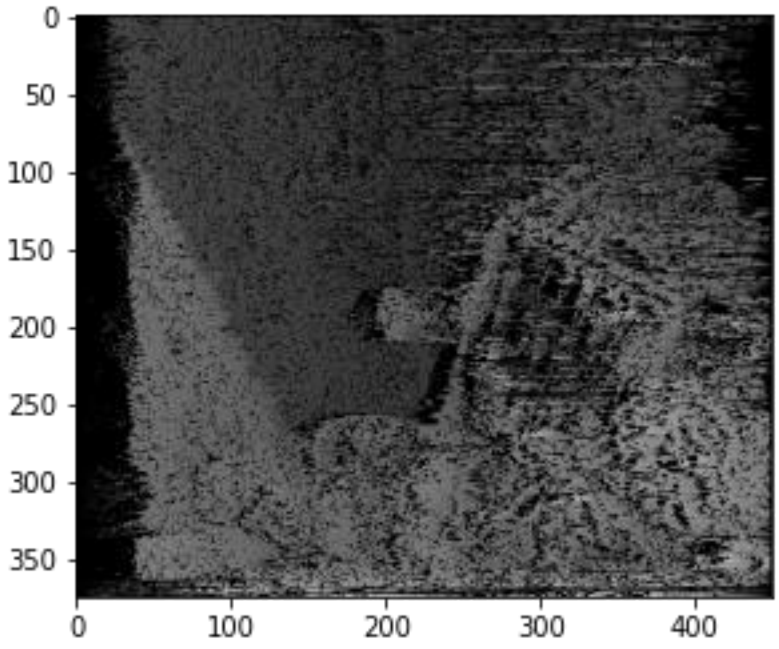
0
50
100
150
200
250
300
350
0
100
200
300
400

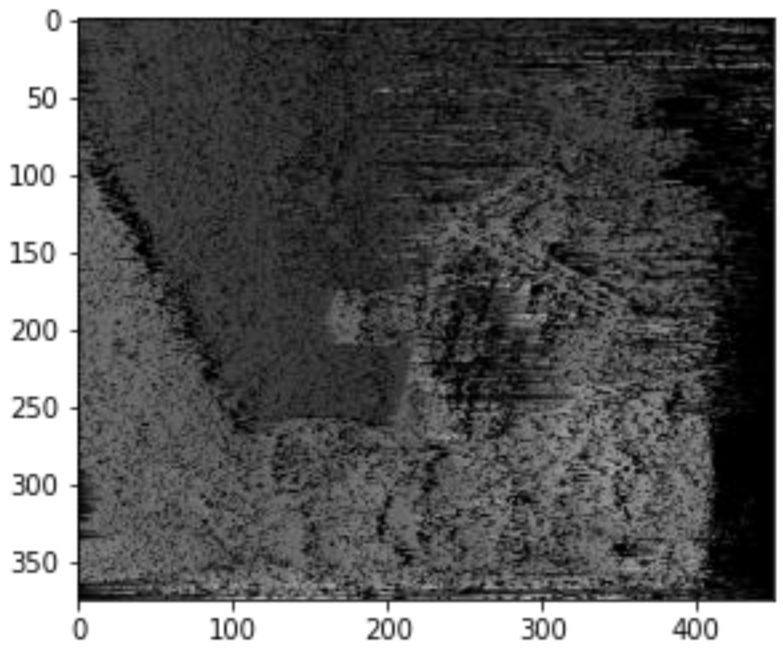


0
50
100
150
200
250
300
350
0
100
200
300
400

Third sample image:







0
50
100
150
200
250
300
350
0
100
200
300
400

1.3 Bonus

There is best path for every D.

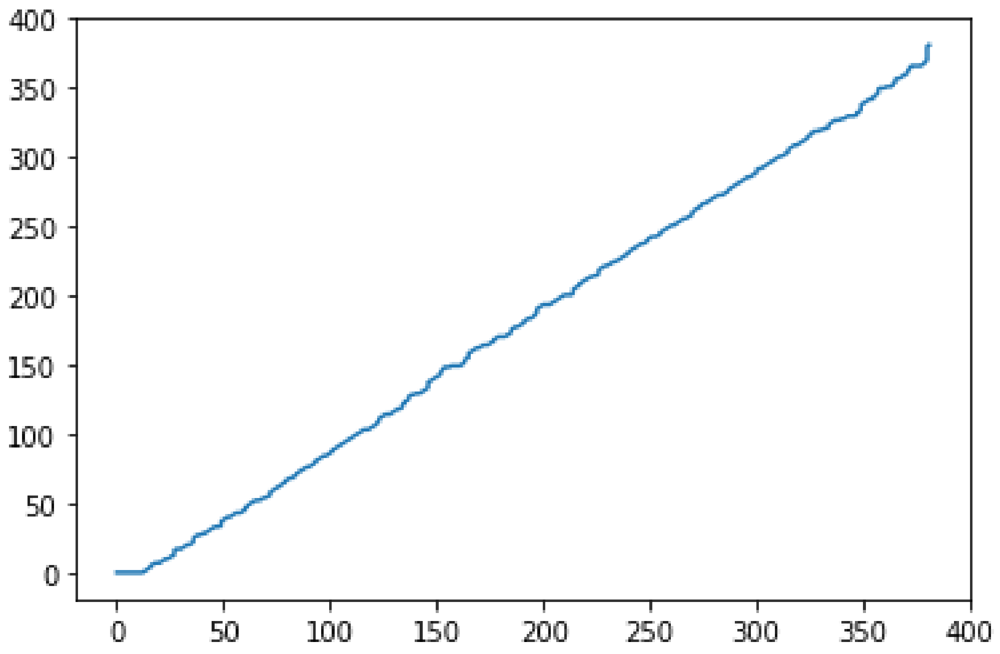
Code:

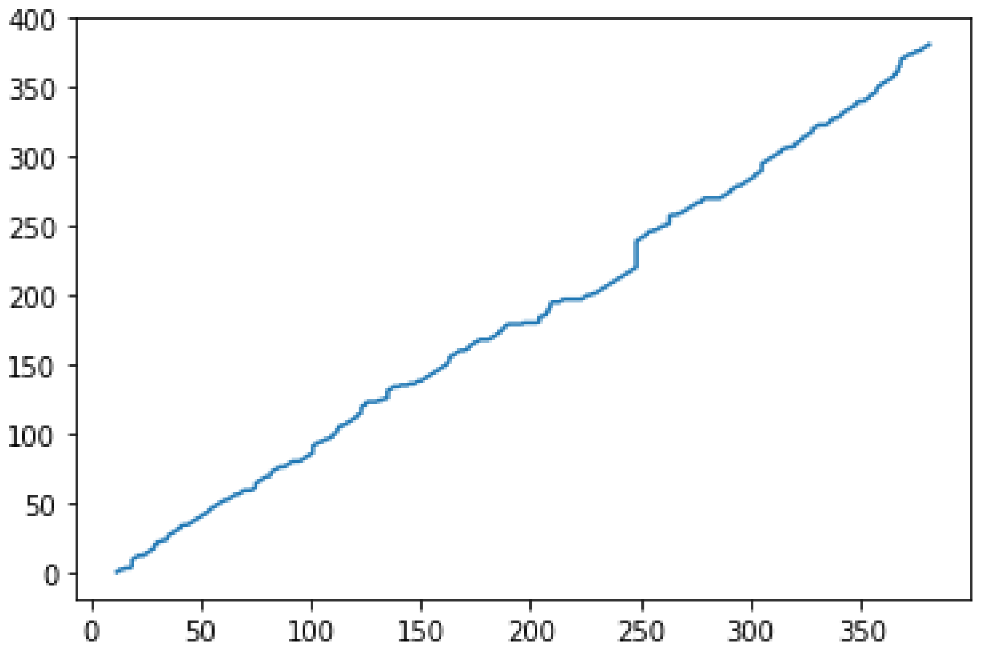
```
def best_path_cal(D):
  N = D.shape[0]
  k = N-1
  l = N-1

  min_path_track = []
  min_path_track.append([k, l])
  while k!=0 and l!=0:
    if k-1>0 and l-1>0:
        dij_diagonal = D[k-1][l-1]
    if k-1>0:
        dij_above = D[k-1][l]
    if l-1>0:
        dij_before = D[k][l-1]

    temp = min(dij_diagonal,min(dij_above,dij_before))

    if temp == dij_diagonal:
      l = l-1
      k = k-1
      min_path_track.append([k, l])
    elif temp ==dij_above:
      k = k-1
      min_path_track.append([k, l])
    else:
      l = l-1
      min_path_track.append([k, l])

  best_path = np.array(min_path_track)
  return best_path
```

Output:

First sample image:

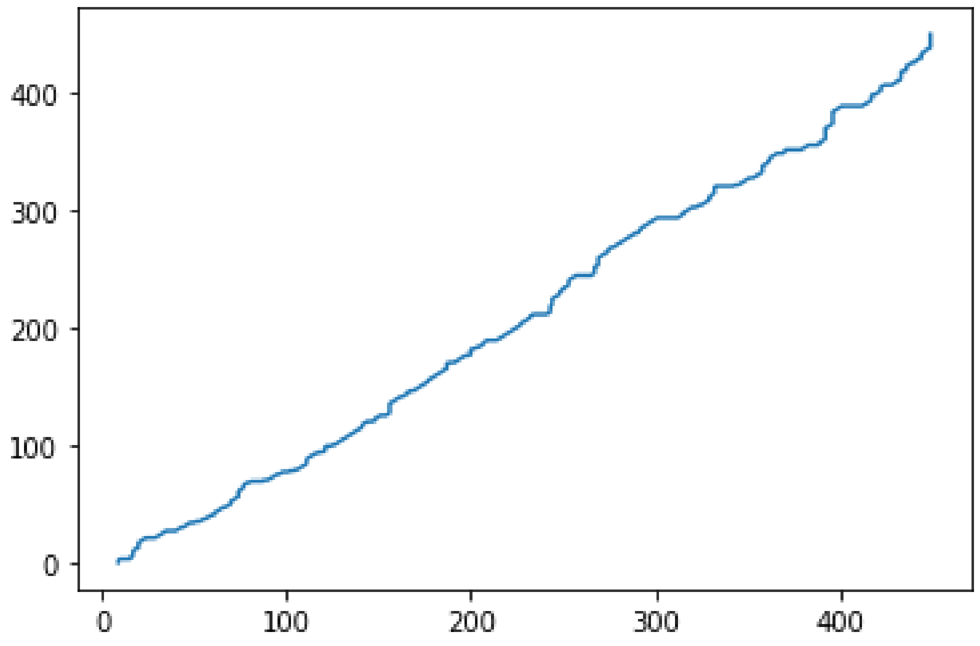
Row 60 and 100.

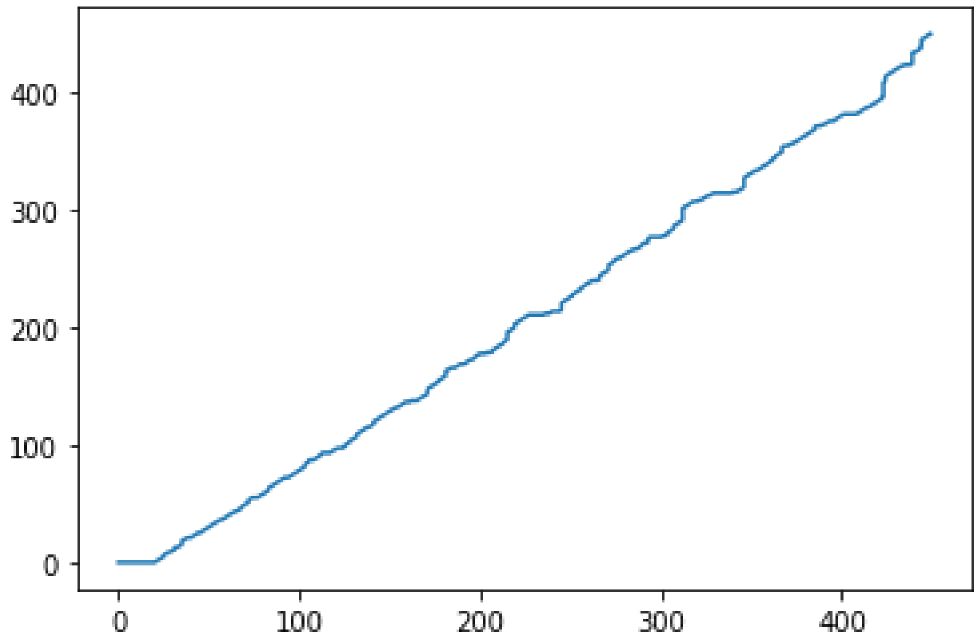




Second sample image:

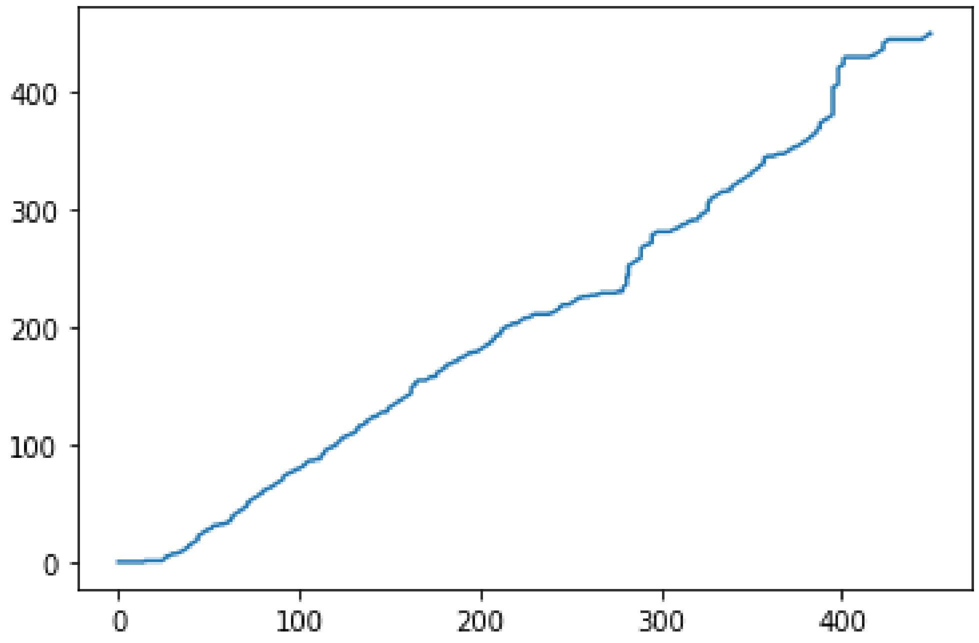
Row 60 and 100.

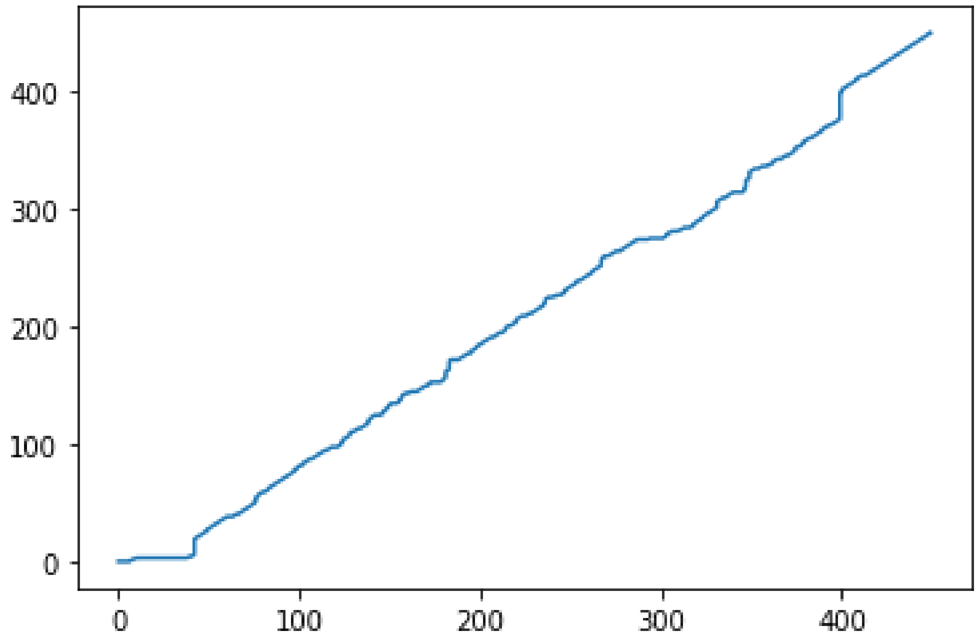




Third sample image:

Row 60 and 100.





Notebook

https://colab.research.google.com/drive/168vPOwEZyZhTcWidRvBWSVGCN12LeZWI?usp=sharing