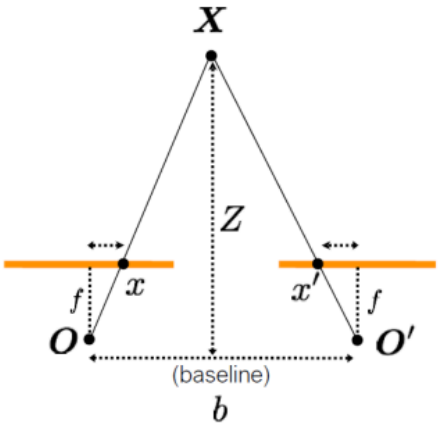
Part 1:

$$\frac{X}{Z} = \frac{x}{f}$$



$$\frac{b - X}{Z} = \frac{x'}{f}$$

Disparity

$$d = x - x' \quad \text{(wrt to camera origin of image plane)}$$

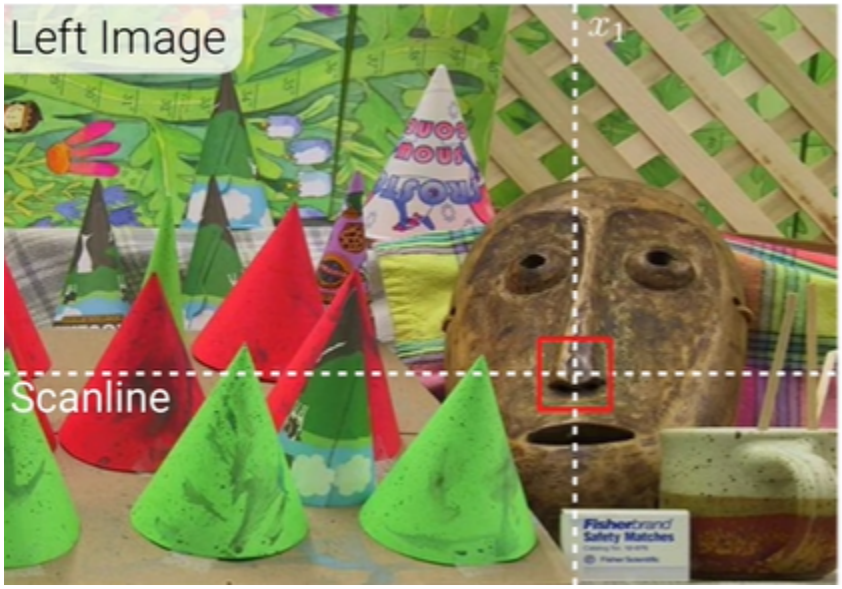
$$= \frac{bf}{Z}$$

``Stereo vision is the process of extracting 3D information from multiple 2D views of a scene. If we have two aligned cameras with a displacement between them in the x-axis. We can extract information about the depth of the objects from the disparity between the two images``

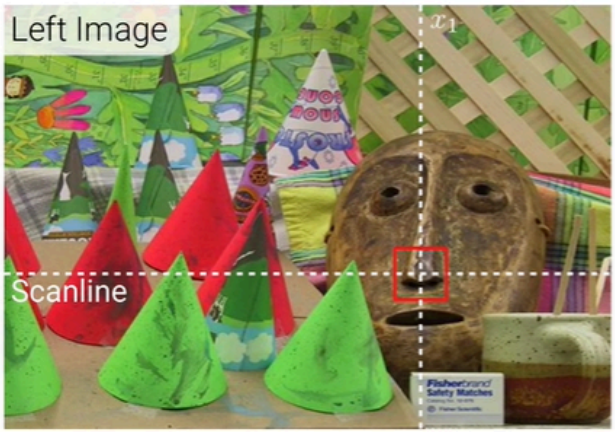
The first step we need to estimate the disparity therefore estimate the depth of the object is matching each object location to its corresponding location in the other image.

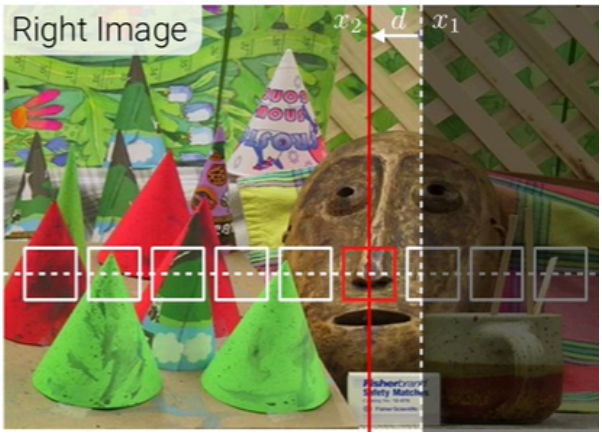
Block Matching:

1. Iterate through each pixel in the image, excluding a border region of size half_window_size.
2. For each pixel, define a window (block) around it in the left image.



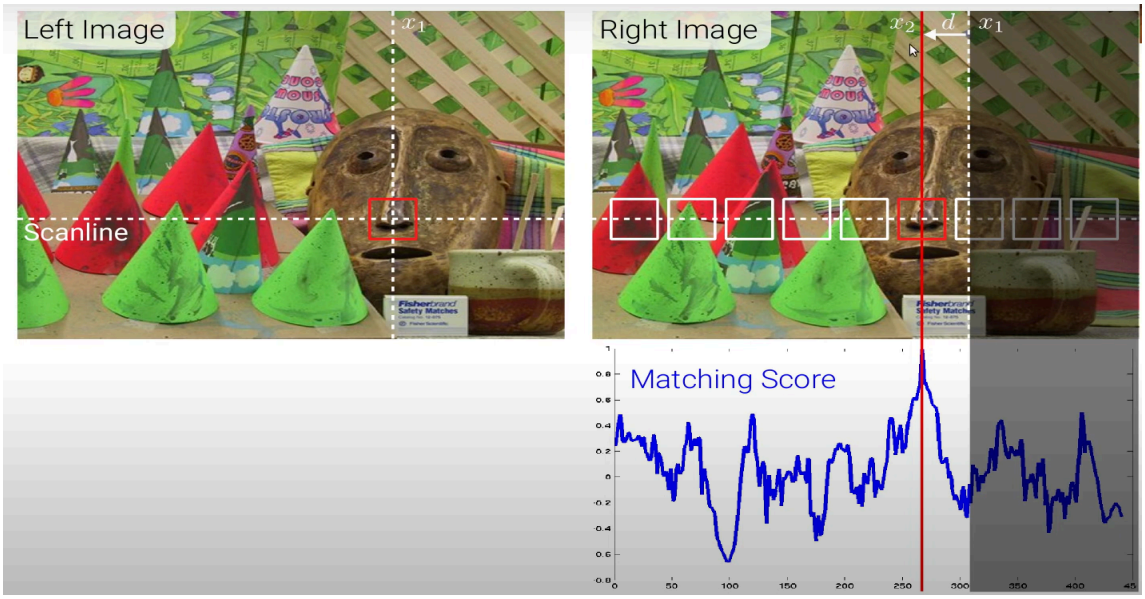
3. Iterate through rows in the right image within a certain range.
4. For each row, define a window in the right image.





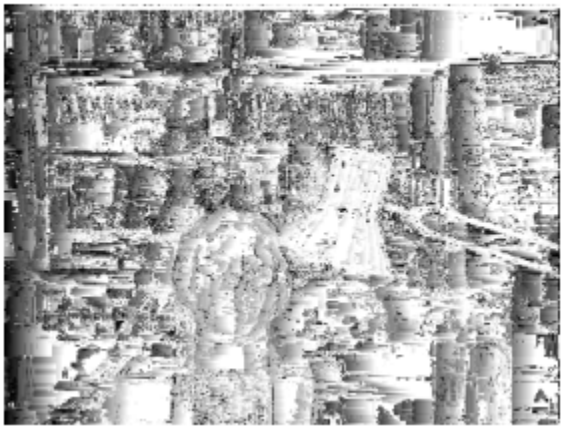
5. Compute the cost between the left and right windows based on the specified cost function ("SSD" or SAD).

6. Update the best match if the computed cost is lower than the current best match.
7. Store the disparity value (difference in column indices) corresponding to the best match.

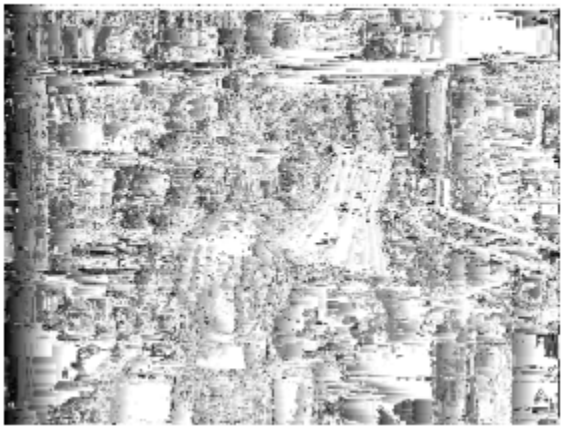


Results

SAD (Window Size 1)

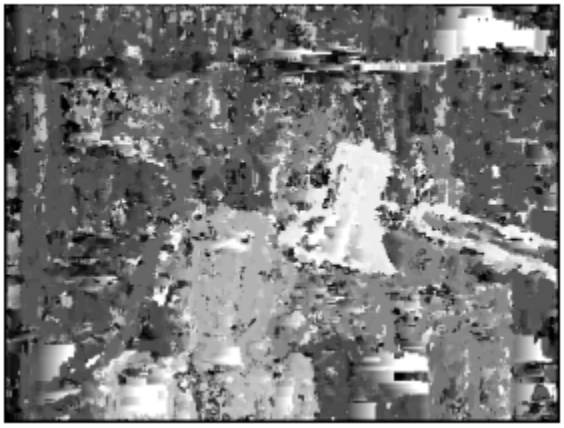


SSD (Window Size 1)

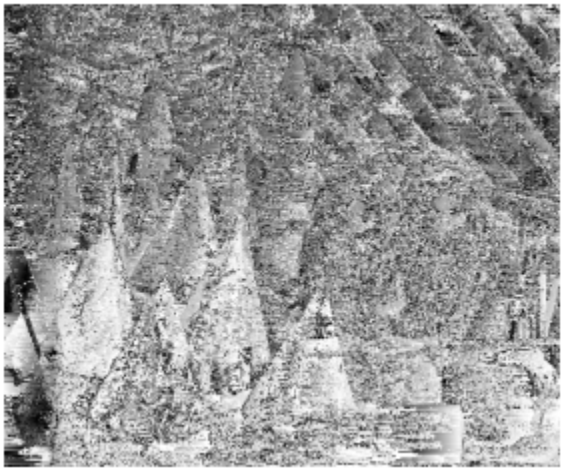


SAD (Window Size 5)

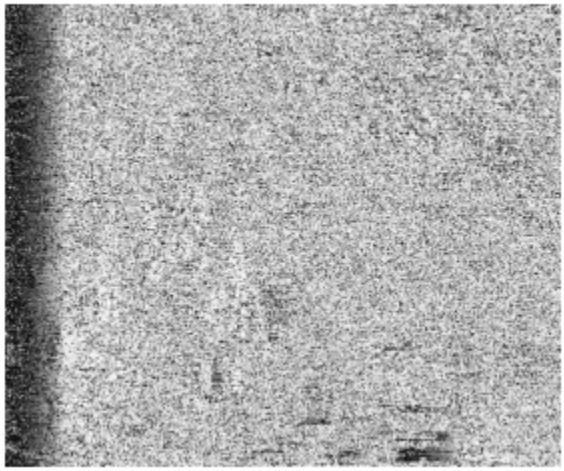
SSD (Window Size 5)



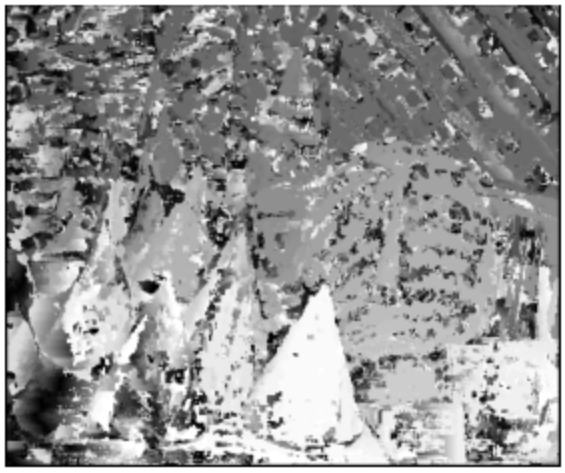
SAD (Window Size 1)



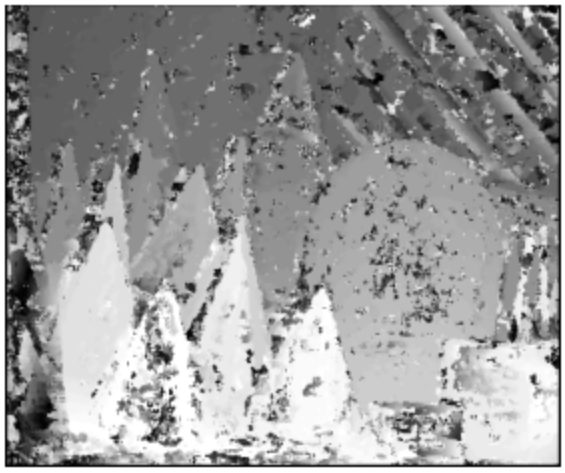
SSD (Window Size 1)



SAD (Window Size 5)

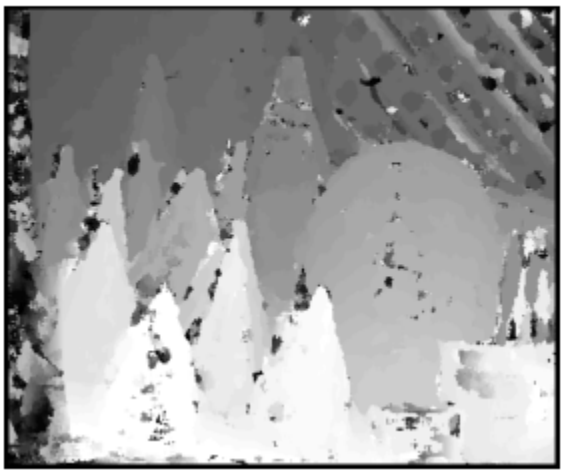


SSD (Window Size 5)

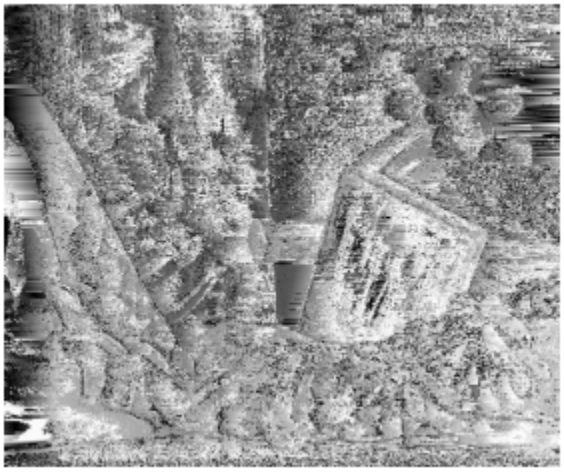


SAD (Window Size 9)

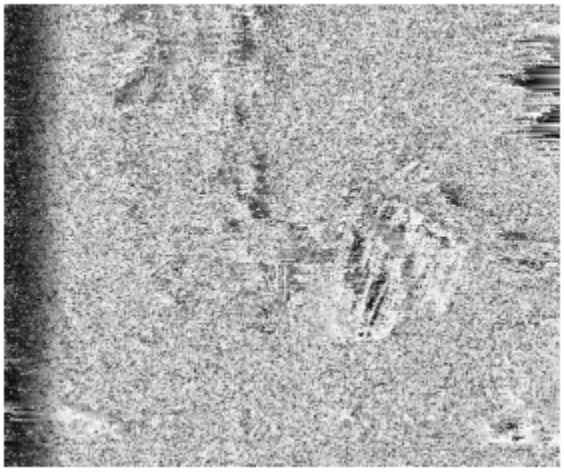
SSD (Window Size 9)

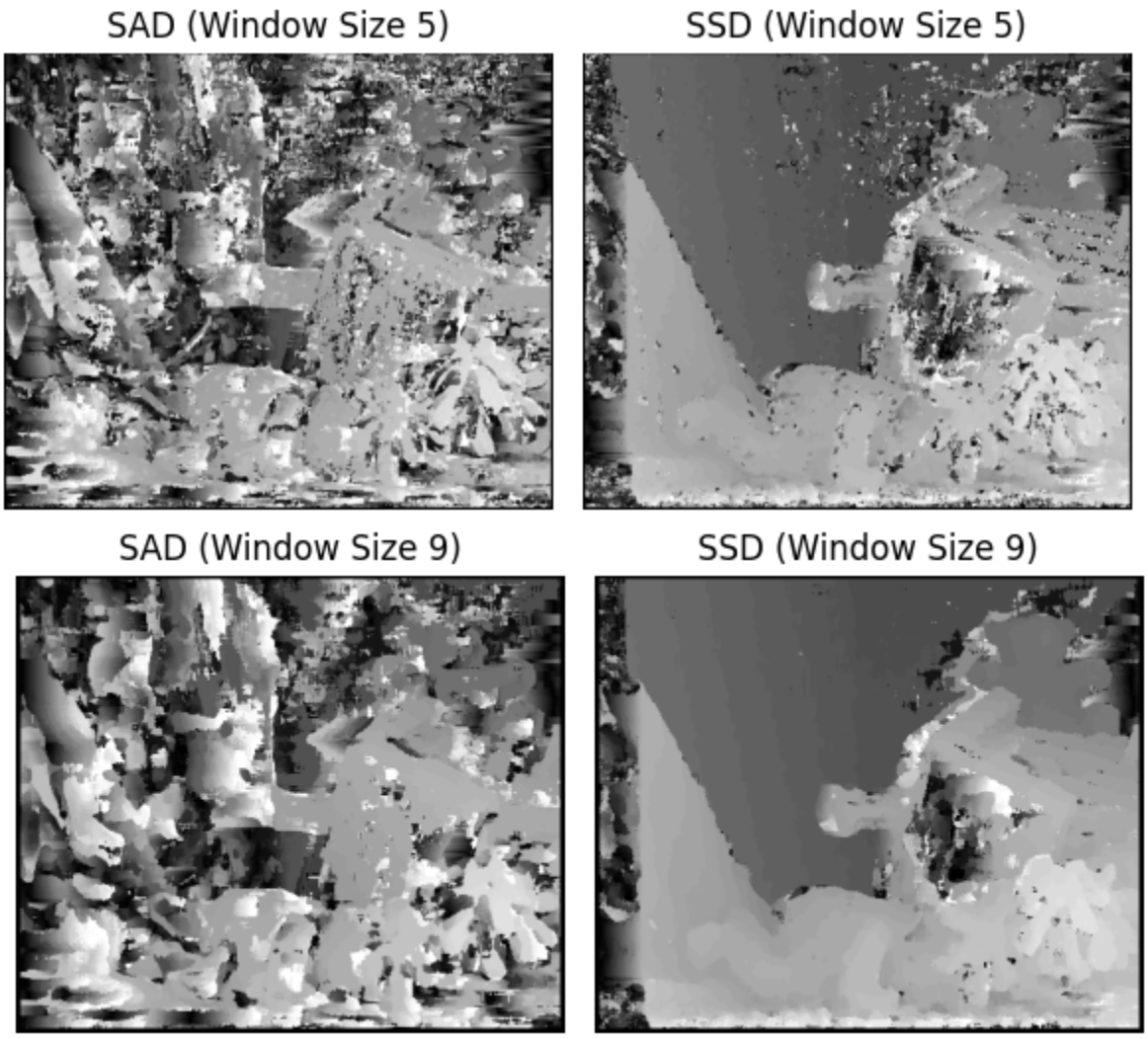


SAD (Window Size 1)



SSD (Window Size 1)





Part 2: ***Dynamic Programming:***

The essential idea in this technique is rather than taking blocks and comparing them we take 2 scanlines from each image and we work on each pixel individually, this gives out more accurate results and better outputs/ disparities. To do that we take the row of pixels of the left image as vertical and the row of pixels from the second image as horizontal and fill the cost matrix between pixels.

The Algorithm:

- The cost for the first element (`dp[i][0][0]`) is the squared difference between the pixel intensities of the corresponding pixels in `img1` and `img2`.

- The first column (`dp[i][j][0]`) and the first row (`dp[i][0][j]`) are initialized with increasing values of a constant `c0`.

- The nested loops perform bottom-up dynamic programming to fill in the rest of the dynamic programming matrix.

- For each pixel `(i, l)` in the input images, it computes the cost as the squared difference between the pixel intensities of `img1` and `img2` using this function

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

- The dynamic programming recurrence relation is used to calculate the cost for each entry in `dp[i][l][r]` by considering three possibilities: a diagonal move (`dp[i][l-1][r-1] + cost`), an upward move (`dp[i][l-1][r] + c0`), and a leftward move (`dp[i][l][r-1] + c0`) using this formula:

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

- The minimum cost is selected, and the value is stored in the dynamic programming matrix.

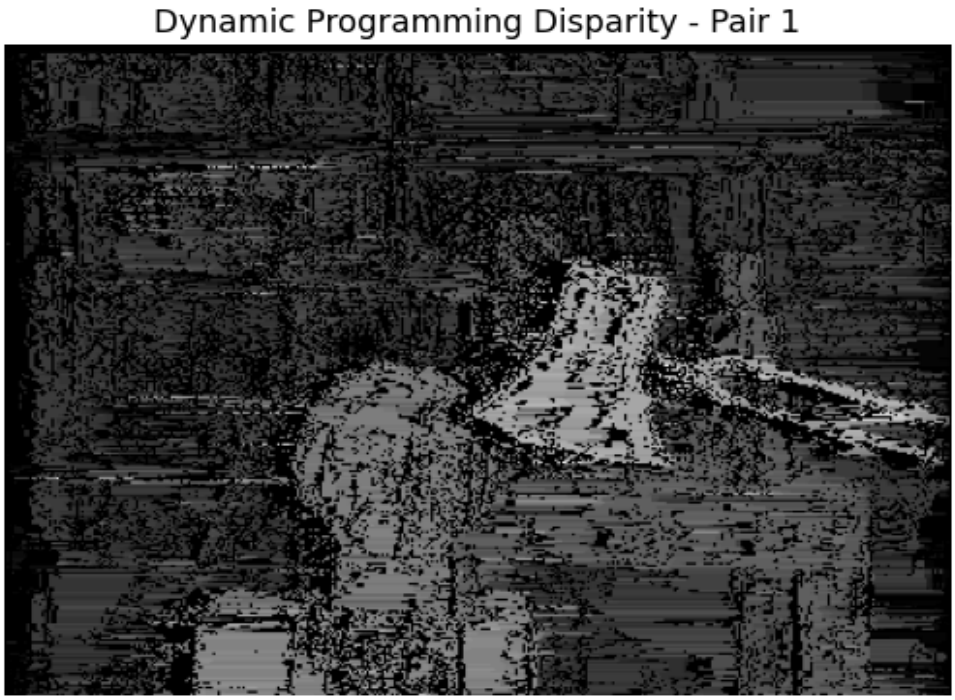
- After filling the dynamic programming matrix, the algorithm backtracks to find the optimal path that minimizes the total cost.

- Starting from the bottom-right corner of the matrix, it moves towards the top-left corner based on the minimum cost decisions made during the dynamic programming phase.

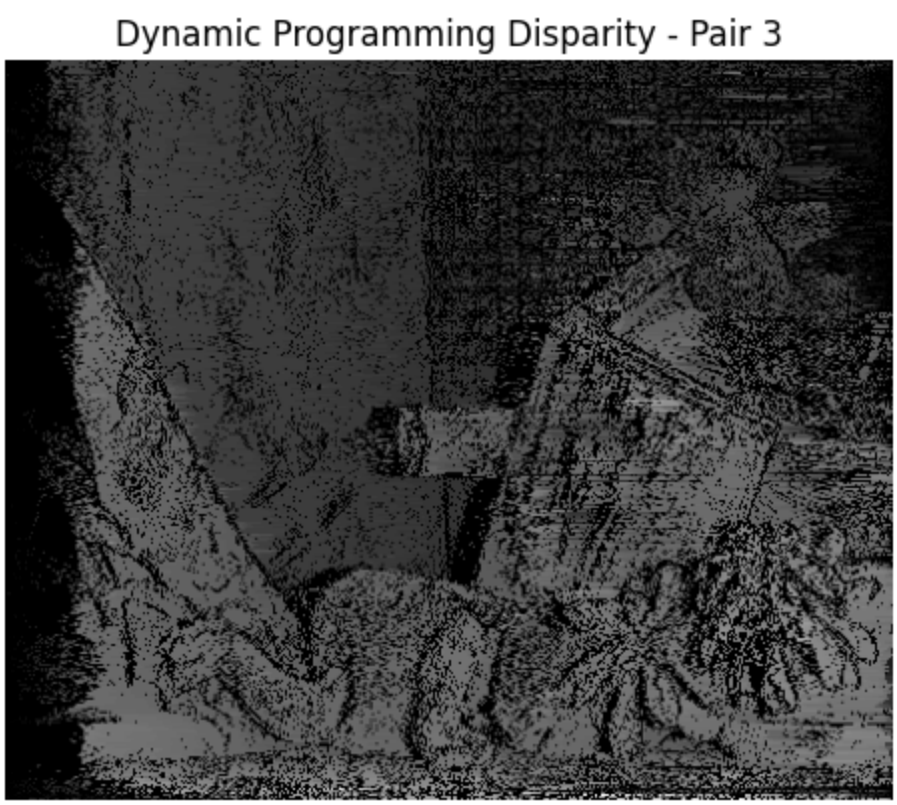
- The disparity map (`output`) is updated with the absolute difference between the column indices of the corresponding pixels in `img1` and `img2`.

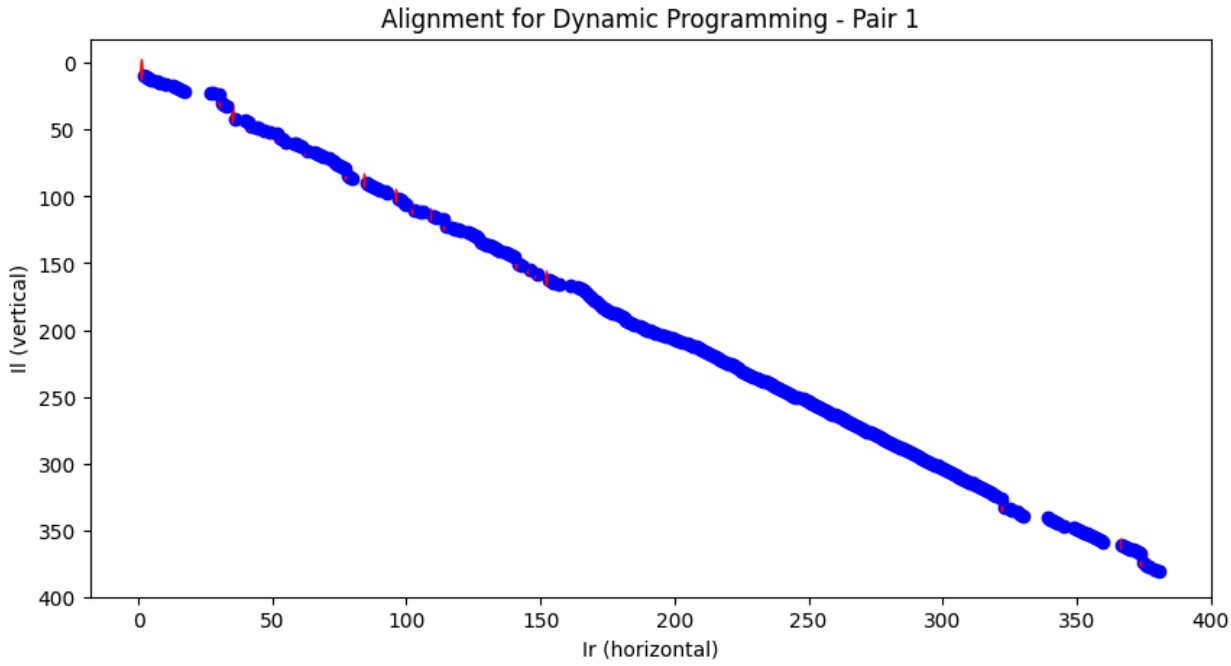
- The final disparity map (`output`) is normalized by multiplying it by `255 / np.max(output)` to bring the values within the range [0, 255].

Results:



Dynamic Programming Disparity - Pair 2


Dynamic Programming Disparity - Pair 3


Alignment for Dynamic Programming - Pair 1
Il (vertical)
Ir (horizontal)
0
50
100
150
200
250
300
350
400