HOMEWORK 2 ANSWERS WRITE UP

Part 1: Keypoint Detection

Question:

- Provide the psuedocode of the corner detector.
- Display the first frame of the sequence overlaid with the detected keypoints. Ensure that they are clearly visible (try `plot(..., 'g.', 'linewidth',3)`).

Answer:

Psuedo Code for the Corner Detector:

```
% im: input image
% tau: threshold


% Smooth Image
gaussian = fspecial('gaussian', 3, 1);
filteredImage = imfilter(im, gaussian, 'replicate');

% Find Image Gradients
[fx, fy] = gradient(filteredImage);

% Calculate Second Moment Matrix Values
IxSquare = fx .* fx;
IySquare = fy .* fy;
IxIy = fx .* fy;

% Filter again to get Smoothed Derivatives
IxSquare = imfilter(IxSquare, gaussian, 'replicate');
IySquare = imfilter(IySquare, gaussian, 'replicate');
IxIy = imfilter(IxIy, gaussian, 'replicate');

% Compute Harris Corner Score using Determinant and
Trace Values
momentDeterminant = (IxSquare .* IySquare) -
(IxIy.^2);
momentTrace = (IxSquare + IySquare).^2;
rScore = momentDeterminant - (tau .* momentTrace);
% Normalize the feature score rScore
```

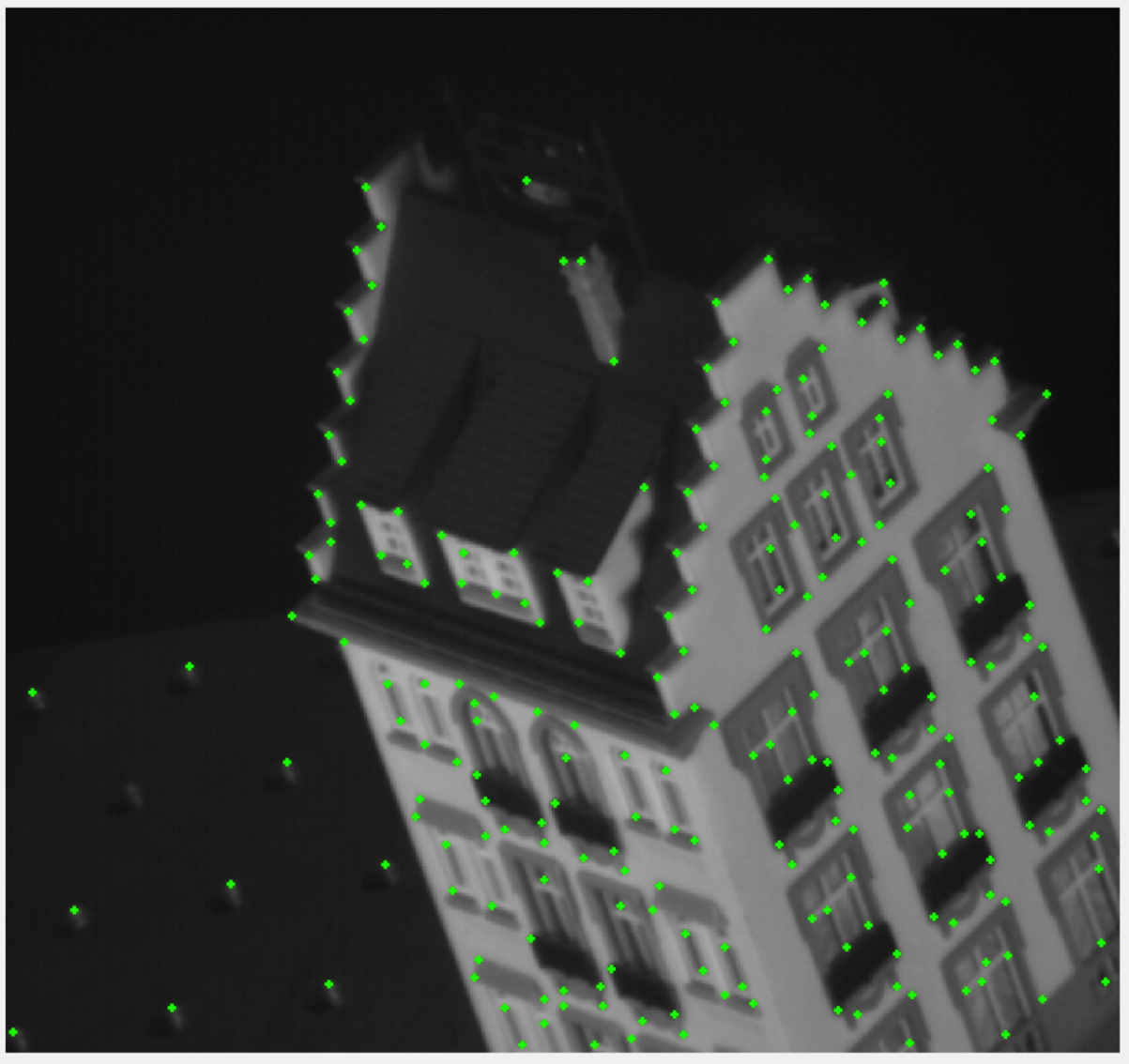
```
maxRScore = max(max(rScore));
rScore = rScore ./ maxRScore;

% Perform Non Maximum Suppresion
sze = 13;
nonMaxRScore = ordfilt2(rScore, sze^2, ones(sze));

% Apply Thresholding
thresholdedImage = (rScore == nonMaxRScore) & (rScore
> max(max(rScore)) * 0.2);
[r, c] = find(thresholdedImage);

% Return output
keyXs = r;
keyYs = c;
```

First frame of the sequence overlaid with the detected keypoints:



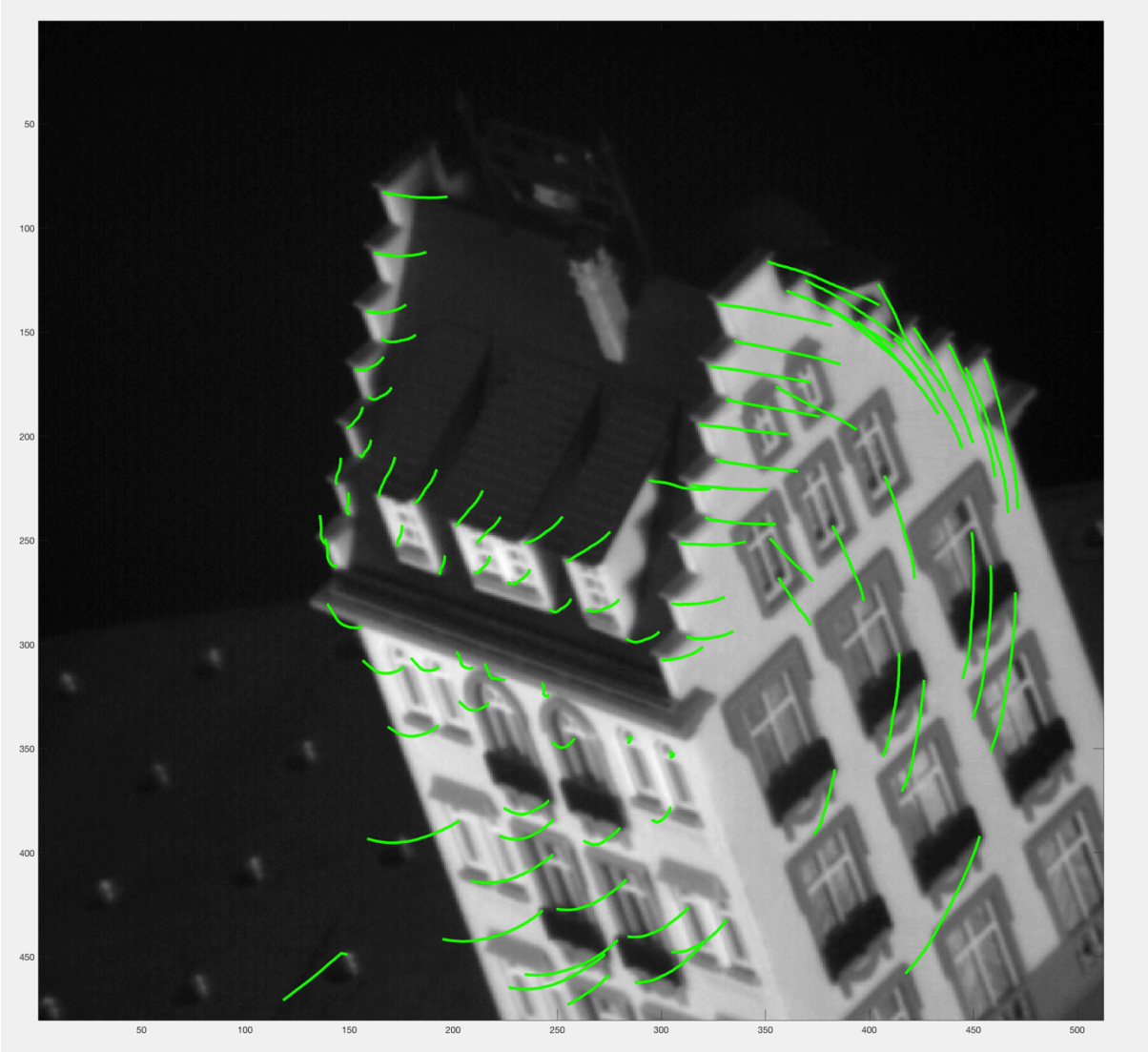
Question:

- For all the keypoints, display (1) the keypoints at the first frame (as green) and (2) the tracked keypoints at the second frame (as red) on the first frame of the sequence.
- For 20 random keypoints, draw the 2D path over the sequence of frames. That is, plot the progression of image coordinates for each of the 20 keypoints. Plot each of the paths on the same figure, overlaid on the first frame of the sequence.
- On top of the first frame, plot the points which have moved out of frame at some point along the sequence.

Answer:

Keypoints in the 1st frame in GREEN and tracked keypoints at 2nd frame in RED:

Draw the 2D path over the sequence of frames. That is, plot the progression of image coordinates for each of the 20 keypoints. Plot each of the paths on the same figure, overlaid on the first frame of the sequence:



NOTE: Points are more than 20 in number, to show better flow tracking.

On top of the first frame, plot the points which have moved out of frame at some point along the sequence:

Points which have moved out of the frame at some point are shown in GREEN.

Part 2: Shape Alignment

Question:

- Explain your algorithm, initialization, and model of the transformation
- Display the aligned results
- Report averaged alignment error
- Report runtime

Answer:

The algorithm used is the Iterative closest points algorithm and shape alignment done with the help of least squares method. Following is the algorithm implemented:

1. Compute translation scaling using variance and mean of points from both images to obtain an initial transformation matrix, Tfm, of size 3 x 3.
2. Shift points in image 1 using Tfm.
3. Use L2 norm to find closest points in image 1 and image 2.
4. Using the closest neighbouring points find the affine transform by solving linear equations given below:

$$\begin{bmatrix} x_i' \\ y_i' \end{bmatrix} = \begin{bmatrix} m_1 & m_2 \\ m_3 & m_4 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \end{bmatrix}$$

5. Find the translation parameters m1, m2, m3, m4, t1 and t2.
6. Obtain new Tfm by multiplying current Tfm by the 3 x 3 matrix of m1, m2, m3, m4, t1 and t2.
7. Repeat steps 3 to 6 until convergence.

Initialization:

1. The mean of the points give the center of mass and translation.
2. The variance of the points gives the scaling.
3. This can be used to obtain the initial transformation by using multiplication of the following matrices:

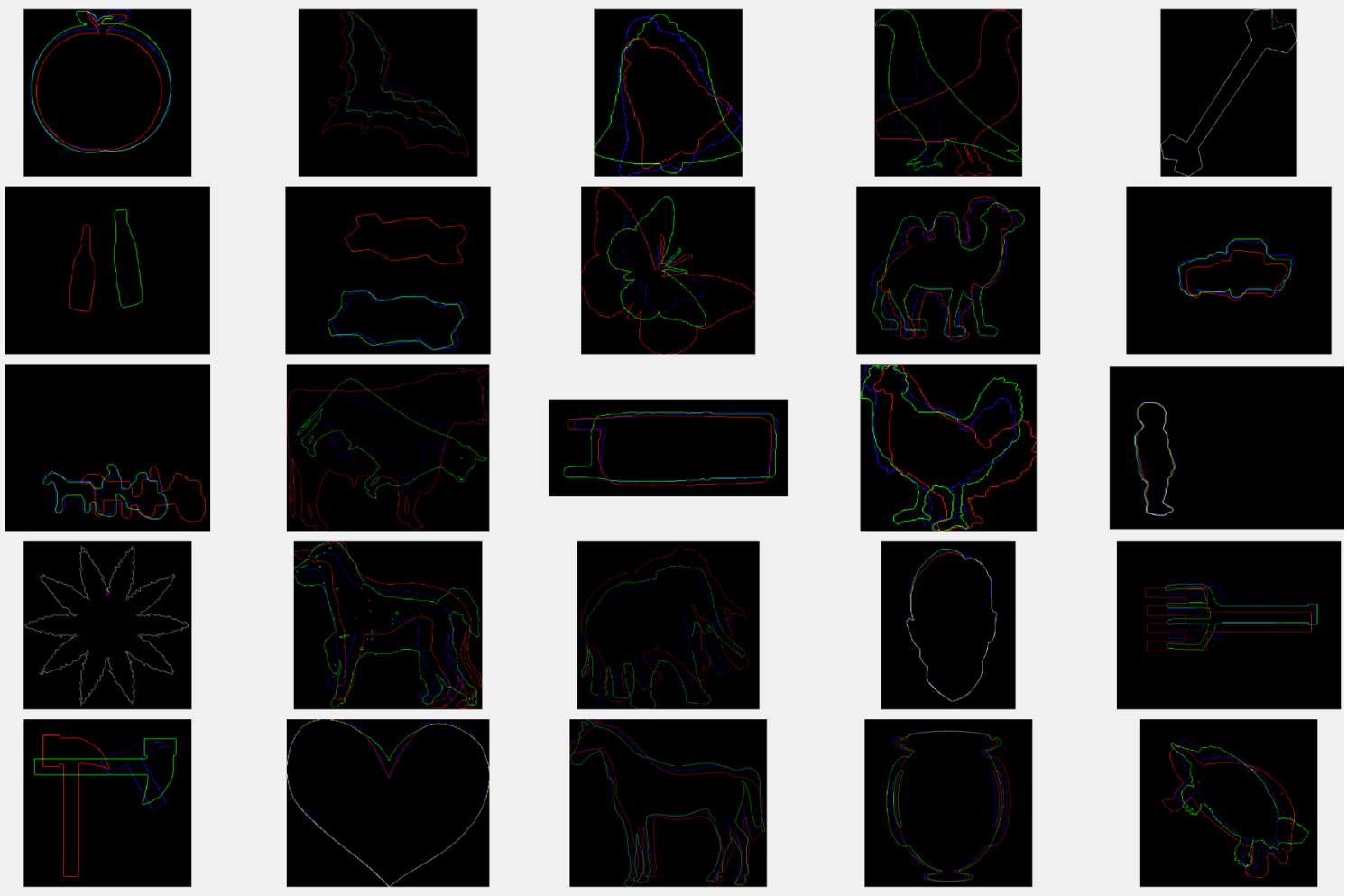
$$\begin{bmatrix} 1 & 0 & \overline{X_2} \\ 0 & 1 & \overline{Y_2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \sigma_2/\sigma_1 & 0 & 0 \\ 0 & \sigma_2/\sigma_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -\overline{X_1} \\ 0 & 1 & -\overline{Y_1} \\ 0 & 0 & 1 \end{bmatrix}$$

where subscript 1 is for points of image 1 and subscript 2 is for points of image 2.

Model:

The transform in use is the affine transform, which includes other transforms like scaling, translation and rotation. The Tfm matrix is of dimension 3 x 3 with the bottom row always being [0 0 1].

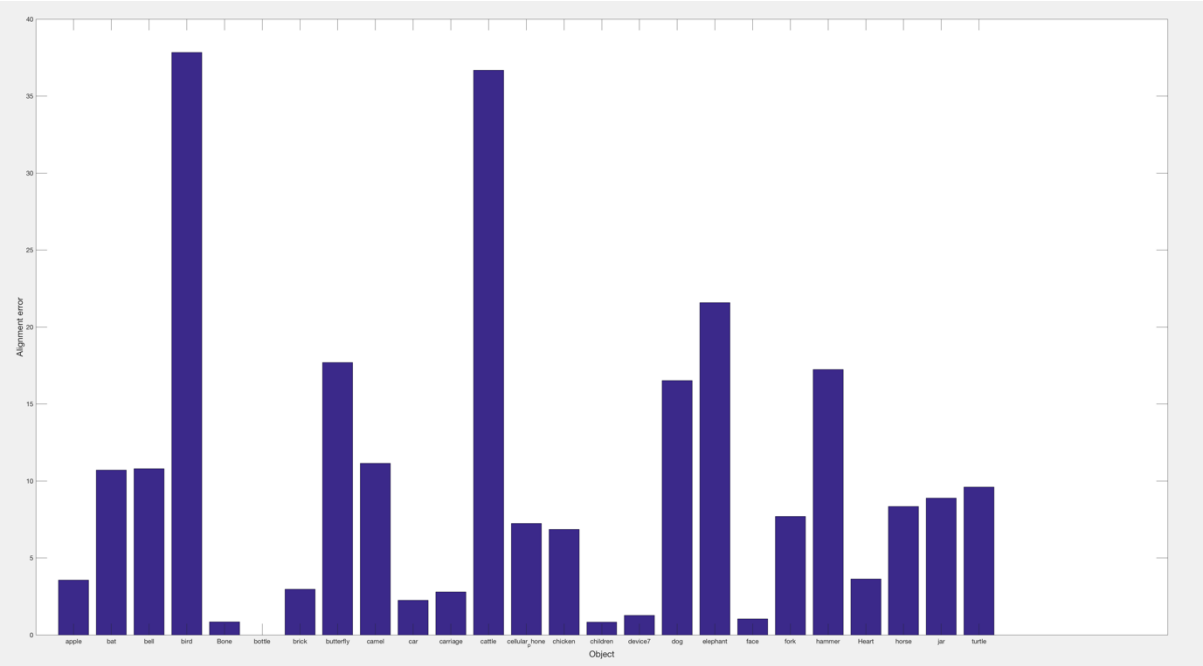
Aligned Results :



The averaged alignment error and runtimes for test cases:

Elapsed time is 4.969212 seconds.
Error for aligning "apple": 3.567185
Elapsed time is 28.808520 seconds.
Error for aligning "bat": 10.709303
Elapsed time is 3.049303 seconds.
Error for aligning "bell": 10.799640
Elapsed time is 11.385433 seconds.
Error for aligning "bird": 37.838135
Elapsed time is 12.811515 seconds.
Error for aligning "Bone": 0.857951
Elapsed time is 1.188295 seconds.
Error for aligning "bottle": NaN
Elapsed time is 2.871024 seconds.
Error for aligning "brick": 2.972271
Elapsed time is 11.193811 seconds.
Error for aligning "butterfly": 17.688843
Elapsed time is 20.922924 seconds.
Error for aligning "camel": 11.155521
Elapsed time is 2.233087 seconds.
Error for aligning "car": 2.251436
Elapsed time is 5.117859 seconds.
Error for aligning "carriage": 2.792652
Elapsed time is 66.505475 seconds.
Error for aligning "cattle": 36.681950
Elapsed time is 8.672469 seconds.
Error for aligning "cellular_phone": 7.248258
Elapsed time is 5.768587 seconds.
Error for aligning "chicken": 6.864122
Elapsed time is 1.534515 seconds.
Error for aligning "children": 0.839553
Elapsed time is 128.310969 seconds.
Error for aligning "device7": 1.277715
Elapsed time is 51.913989 seconds.
Error for aligning "dog": 16.517281
Elapsed time is 101.909054 seconds.
Error for aligning "elephant": 21.592793
Elapsed time is 5.373575 seconds.
Error for aligning "face": 1.041069
Elapsed time is 33.703339 seconds.
Error for aligning "fork": 7.699709
Elapsed time is 3.548940 seconds.

Error for aligning "hammer": 17.236469
Elapsed time is 16.248940 seconds.
Error for aligning "Heart": 3.641494
Elapsed time is 104.211113 seconds.
Error for aligning "horse": 8.340858
Elapsed time is 34.532624 seconds.
Error for aligning "jar": 8.890699
Elapsed time is 9.304210 seconds.
Error for aligning "turtle": 9.600401
Averaged alignment error = 9.718593



Part 3: Object Instance Recognition

Question:

Use plotmatches to display

- (1) the matches by thresholding nearest neighbor distances.
- (2) the matches by thresholding the distance ratio. Describe the differences of (1) and (2).

Answer:

Matches by thresholding nearest neighbor distances:

Matches by thresholding distance ratio:

Threshold = 0.7

Difference in 1 and 2:

The distance ratio gives better results since the points will only be a match if there is a large difference between the closest and the 2nd closest descriptors. The nearest neighbor method gives a lot of false positives which is worse performance than the distance ratio method.

Graduate Credit

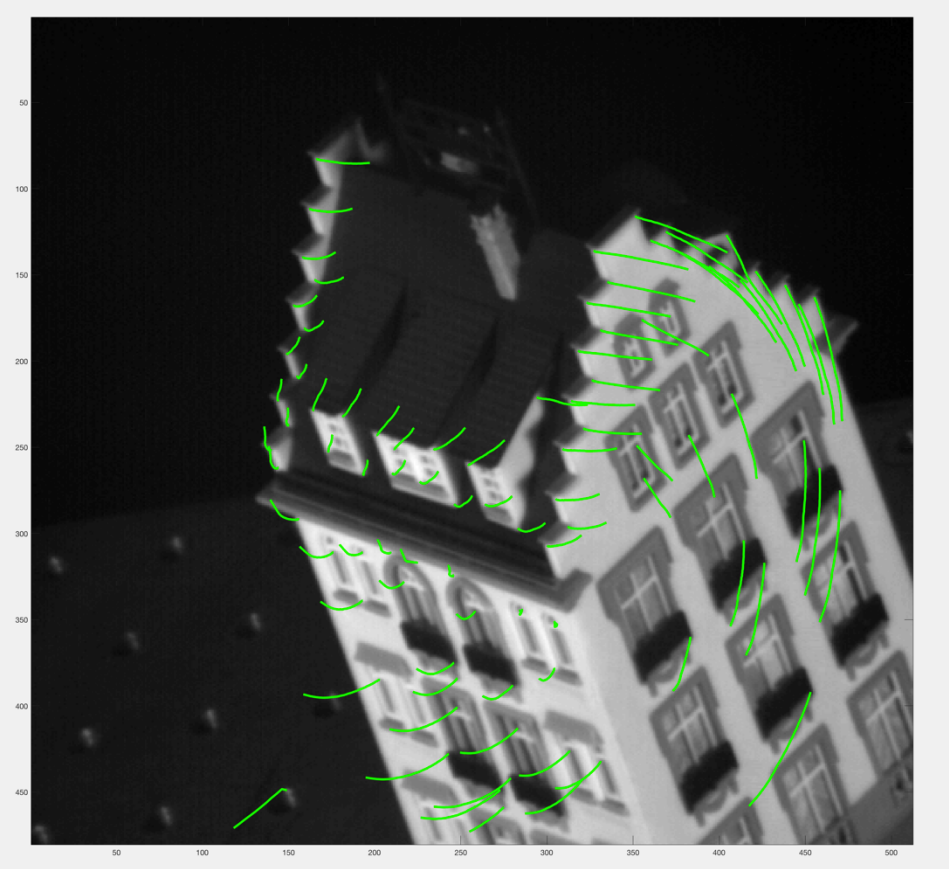
Part 1: Feature Tracking

Question:

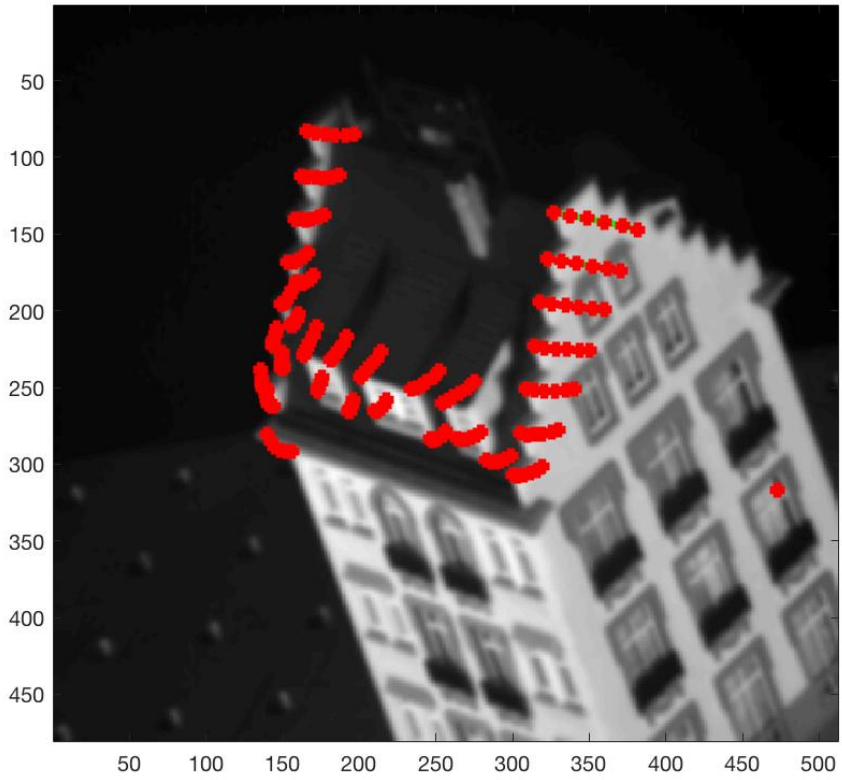
- Implement a coarse-to-fine feature tracker with at least 3 levels. Test your implementation on frames 10; 20; 30; 40; 50. Compare your results with the single-scale tracker. Display the 6 frames and overlay the tracked keypoints.
- Implement a multi-scale optical flow algorithm using the KLT tracker. Display your flow field from frame 0 to frame 10 using the flowToColor function.

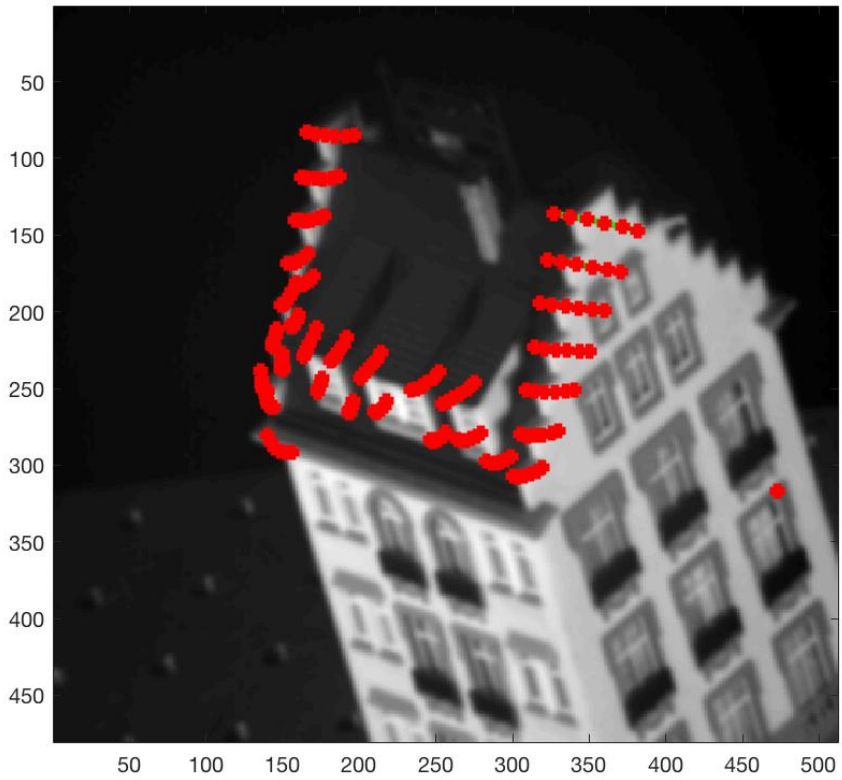
Answer:

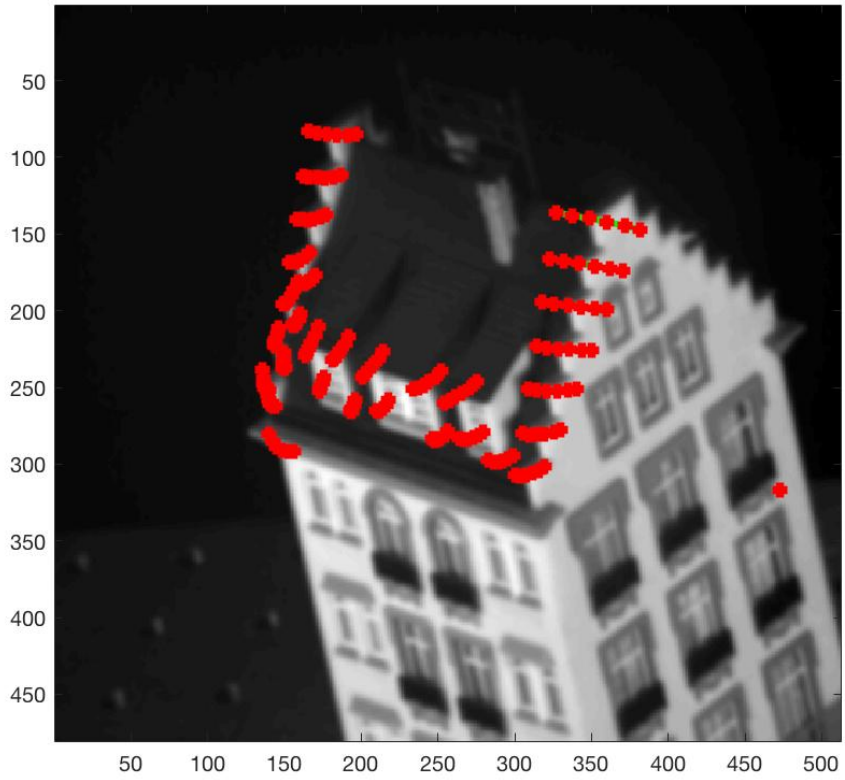
Single Level Tracking:

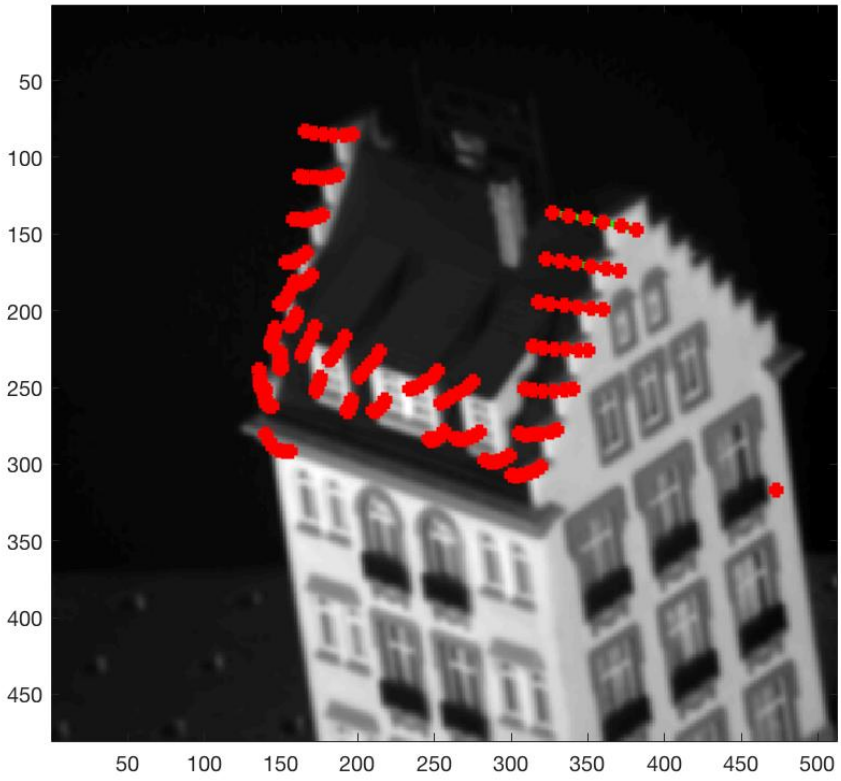


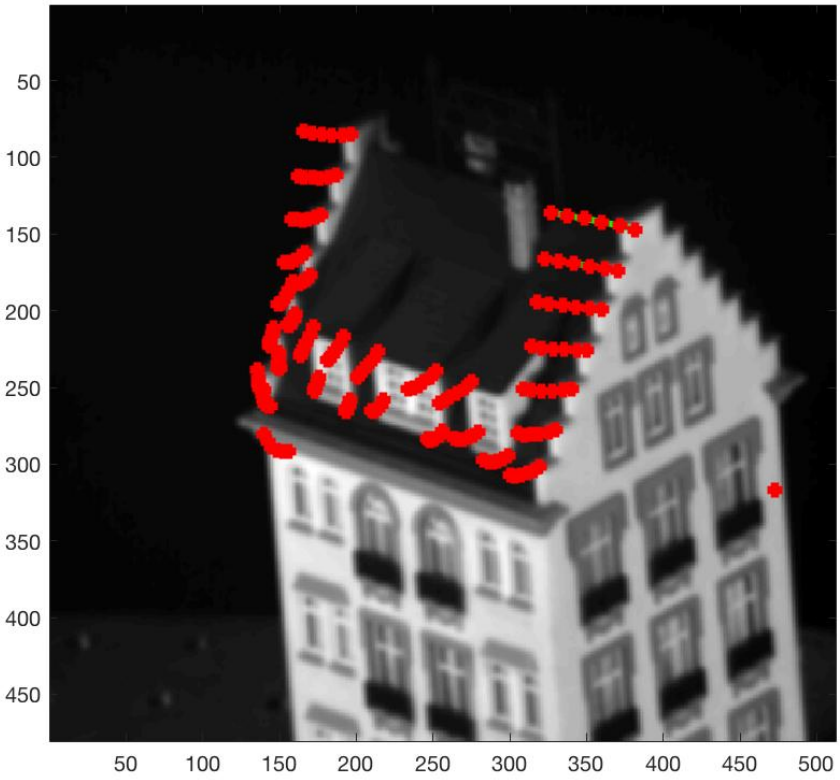
Multi Scale Tracking:

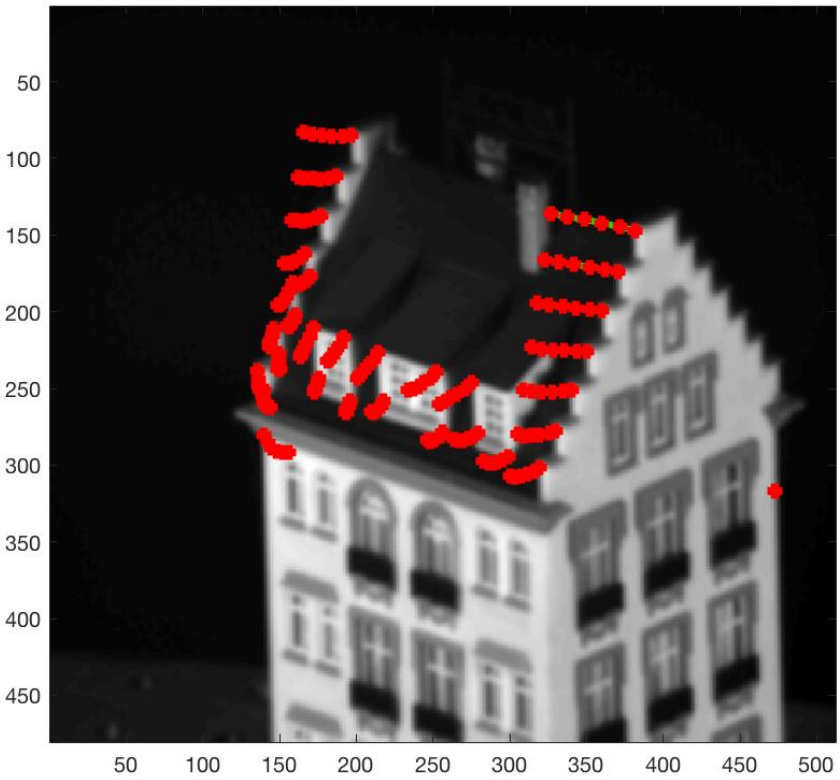

50
100
150
200
250
300
350
400
450
50
100
150
200
250
300
350
400
450
500


50
100
150
200
250
300
350
400
450
50
100
150
200
250
300
350
400
450
500


50
100
150
200
250
300
350
400
450
50
100
150
200
250
300
350
400
450
500


50
100
150
200
250
300
350
400
450
50
100
150
200
250
300
350
400
450
500


50
100
150
200
250
300
350
400
450
50
100
150
200
250
300
350
400
450
500


50
100
150
200
250
300
350
400
450
50
100
150
200
250
300
350
400
450
500

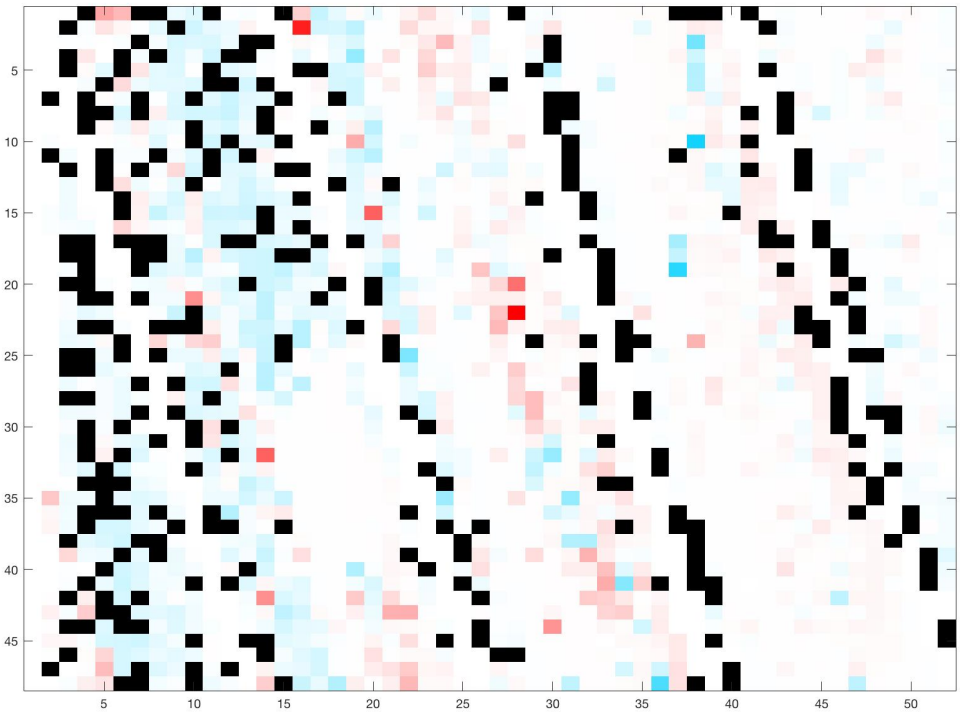
Single level vs Multi scale tracking:

Single level tracking is not so efficient if there are large motions in the frames. Multi Scale tracking is beneficial because the feature size is larger which helps in tracking large motions. Multi scale tracking is also faster in runtime.

Optical flow Algorithm:

1. Subsampled the image by creating arrays of indices with a spacing of 10 pixels in 2 – D.
2. Passed the indices array to trackPoints function to detect motion flow from frame 1 to 10.
3. Saved the tracked points in trackXFlow and trackYFlow arrays to give motion.
4. Created the H x W x 2 tensor to fill gradients in X and Y directions.
5. Passed this tensor to flowToColor function and displayed the response showing optical flow.

Results: Flow Field:


5
10
15
20
25
30
35
40
45
5
10
15
20
25
30
35
40
45
50

Part 2: Shape Alignment

Question:

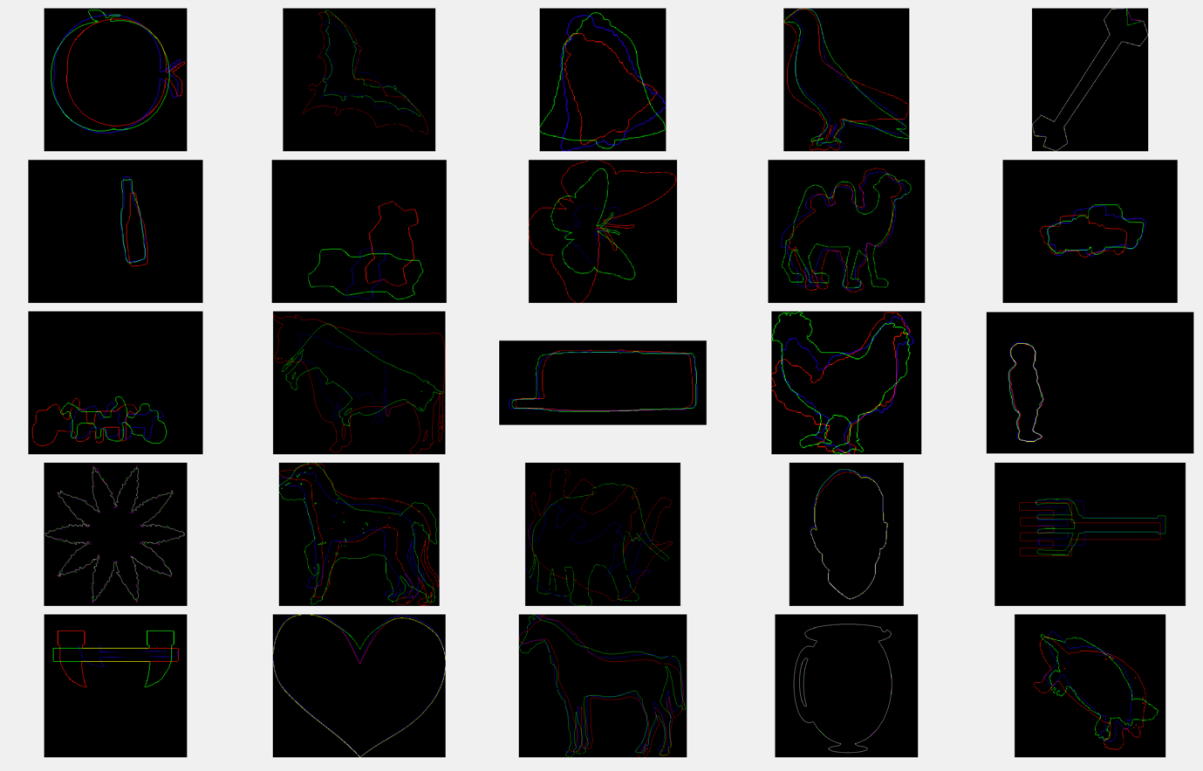
Improve your shape alignment results. For example, you can try using multiple initialization trials (e.g. horizontal flips) and pick the best one. Explain your approach and report improvement.

Answer:

I used the correlation function called corr2, to find out the similarity between the two images after applying the following transforms on the first image. The similarity function gives a closeness score for an image to another reference image. The getBestOrientation function returns the transformed image with the highest correlation score. These were the transforms applied:

1. Horizontal Flip
2. Vertical Flip
3. Both Horizontal and Vertical Flip
4. Rotation in 90 degrees
5. Rotation in 180 degrees
6. Rotation in 270 degrees

Results:



Improvement:

The averaged alignment error reduced to 6.41786 from 9.7

Part 3: Object Instance Recognition

Question:

Use a space partitioning data structure like a kd-tree or some third party approximate nearest neighbor package to accelerate matching. Report the before/after runtime.

Answer:

I used the inbuilt createns and knnsearch function to form a kdtree data structure from descriptor 2 and find nearest neighbours by the knn nearest neighbours algorithm.

```
kdVar = createns(desb2', 'NSMethod',
'kdtree');
neigh = knnsearch(kdVar, desb1', 'K', 2);
```

Runtime with calculation of individual Euclidean distances for each descriptor in image 1:

Elapsed time is 3.989690 seconds. (Run time large due to use of for loops.)

Runtime with finding nearest neighbours using kdtree and knnsearch for each descriptor in image 1:

Elapsed time is 0.146675 seconds.

Question:

Suppose that you have matched a keypoint in the object region to a keypoint in a second image (see the figure below). Given the object bounding box center x-y, width, and height (x1, y1, w1, h1) and the position, scale, and orientation of each keypoint (u1,v1,s1, theta1; u2,v2,s2,theta2), show how to compute the predicted center position, width, height, and relative orientation of the object in image 2.

Answer:

The procedure to find out predicted center position, width, height, and relative orientation of the object in image 2 is given below:

object 1 in Image 1 = (x_1, y_1, w_1, h_1)

point of object$_1$ in Image 1 = $(u_1, v_1, s_1, \theta_1)$

point of object$_2$ in Image 2 = $(u_2, v_2, s_2, \theta_2)$

w_2 and h_2 can be found out using scaling :

$$w_2 = \frac{s_2}{s_1} w_1$$

$$h_2 = \frac{s_2}{s_1} h_1$$

The relative rotation angle =

$$\theta = \theta_2 - \theta_1.$$

x_2 and y_2 can be obtained by translation, scaling and rotation ;

$$\text{Rotation} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

$$[x_2, y_2] = [u_2, v_2] + \frac{s_2}{s_1} \times \text{Rotation} \times [x_1 - u_1, y_1 - v_1]$$