Computer Vision – HW2

Abhinuv Pitale

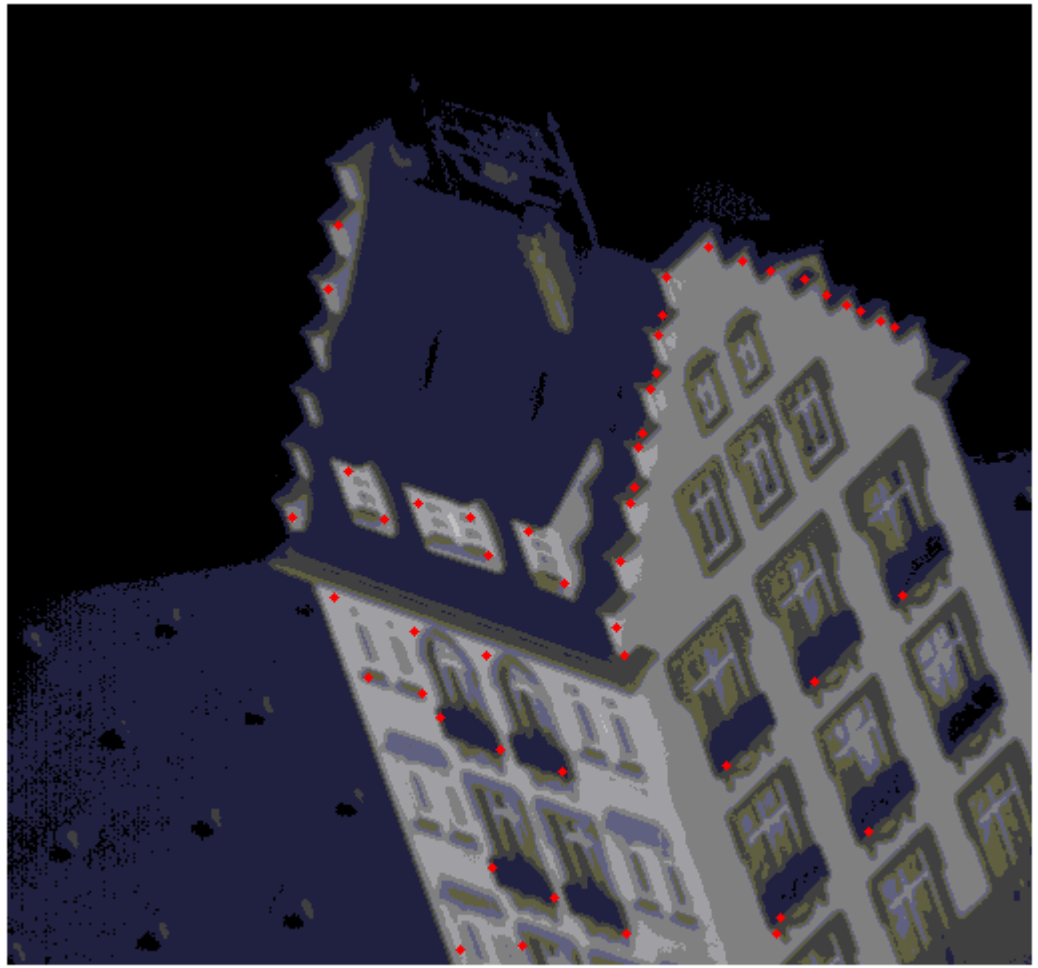
Problem 1: Keypoints and Feature Tracking

- Part A : Keypoint Detection using second moment matrix to detect corners.
- Pseudo Algo
  1. Smoothen the image to clean and remove any noise
  2. Get Image gradients in the x and y directions Ix and Iy
  3. Compute Ix*Ix, Ix*Iy, Iy*Iy
  4. Smoothen Ix*Ix, Ix*Iy, Iy*Iy to get local gradients
  5. Compute Harris function using
  6. Normalize this matrix to 1
  7. Perform non-max suppression
     ->just keep the maxima of a local neighbourhood. (used a 5*5 window)
  8. Threshold the points.

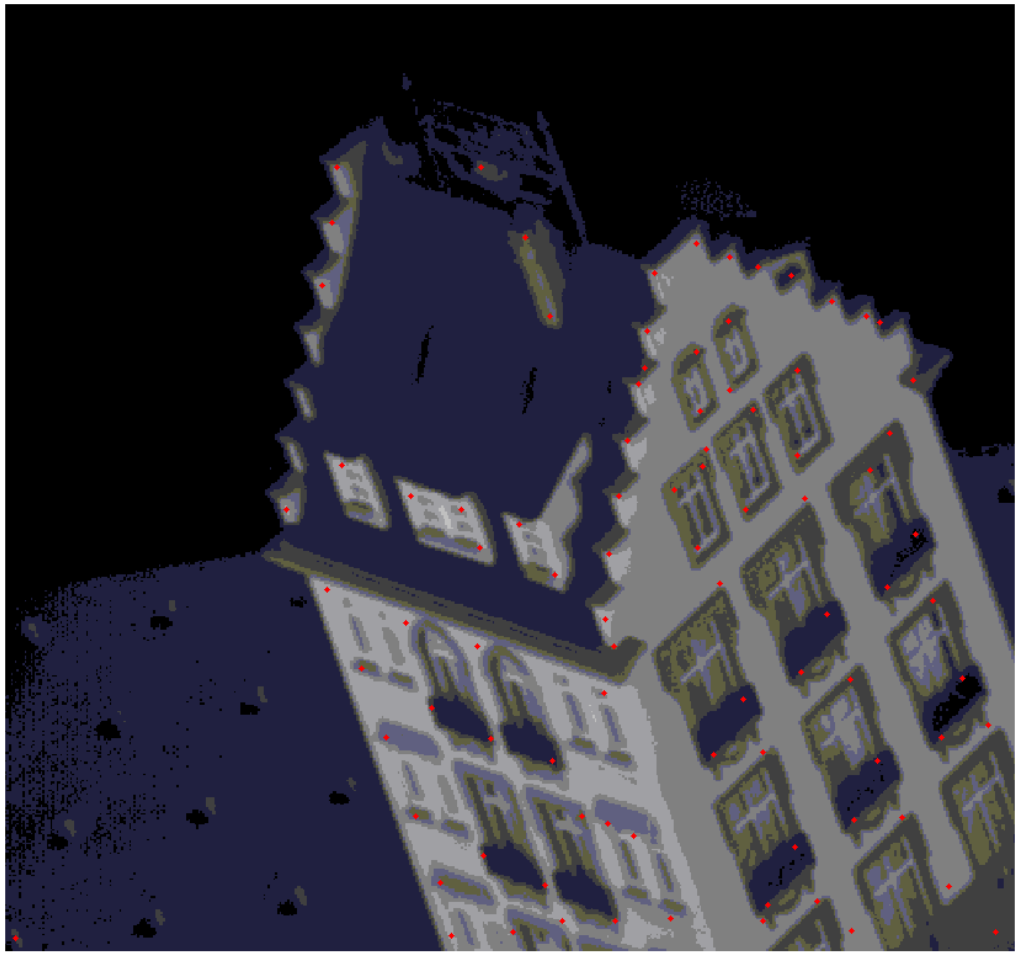
$$\mu(\sigma_I, \sigma_D) = g(\sigma_I) * \begin{bmatrix} I_x^2(\sigma_D) & I_x I_y(\sigma_D) \\ I_x I_y(\sigma_D) & I_y^2(\sigma_D) \end{bmatrix}$$

- Pseudo Code
  1. imGau = GaussianFilter(inputImage,tuningConstants1)
  2. Ix,Iy = gradient(imGau)
  3. Compute Ix*Ix, Ix*Iy, Iy*Iy
  4. A = GaussianFilter(Ix*Ix,tuningConstants2)
     B = GaussianFilter(Ix*Iy,tuningConstants2)
     C = GaussianFilter(Iy*Iy,tuningConstants2)
  5. H = (A.*C - B.*B)./(A+C);
  6. Normalize H
  7. Non Max suppression
     nonMaxNormH(find(normH ~= maximum(normH,25,ones(5)))) = 0;
  8. Return find(nonMaxNormH > tuningConstant3);

Tau = 0.5 with 5*5 window for non max suppression



Tau = 0.6 with 10*10 window for non max supp.

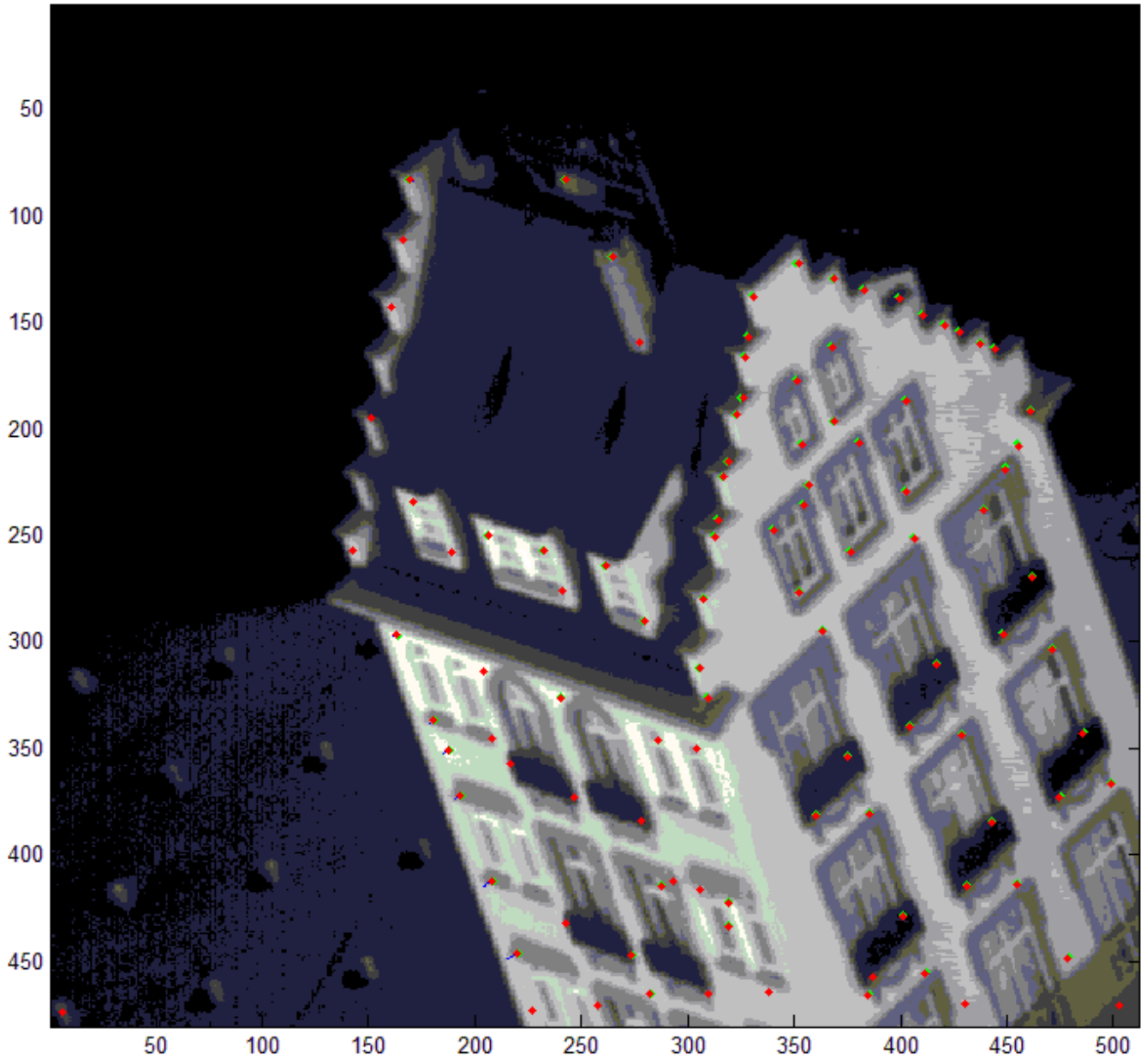


- Part 2: Key Point Tracking

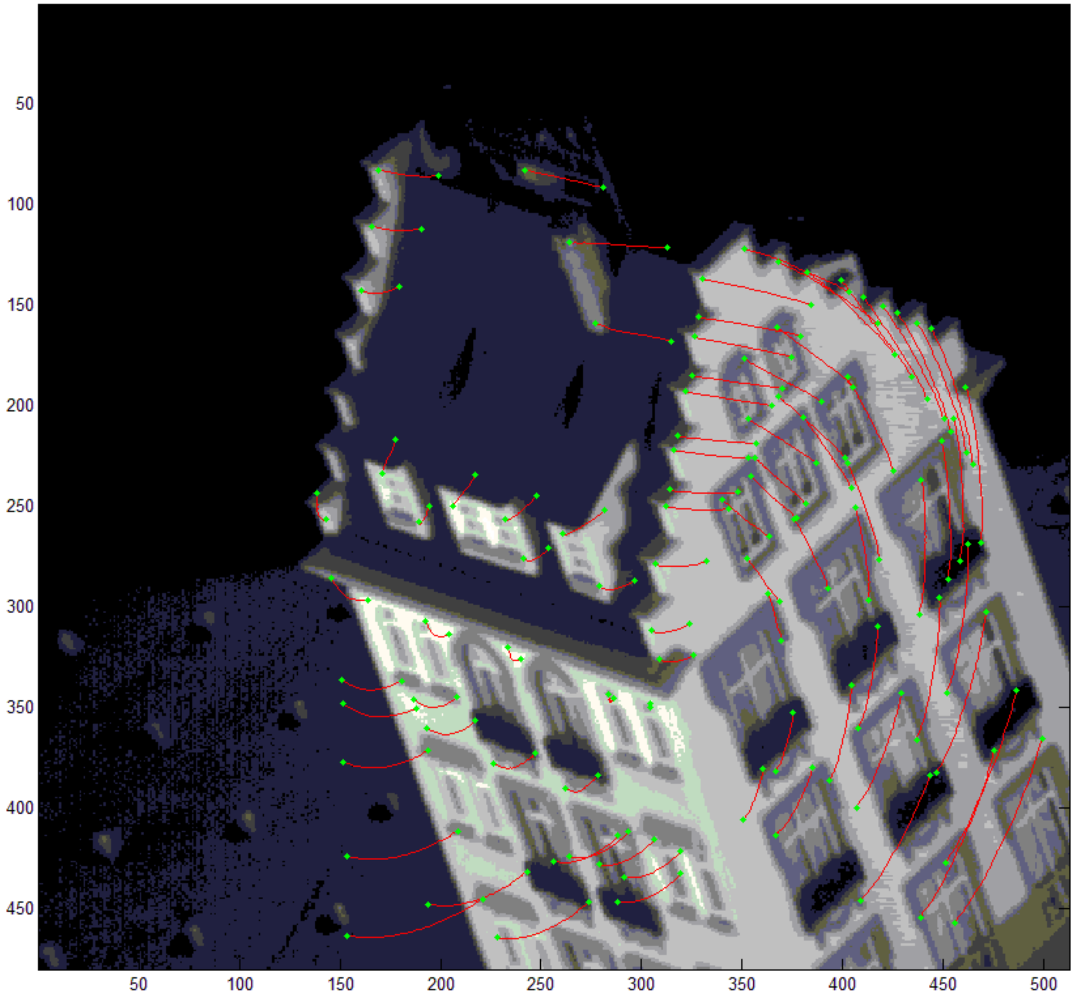
For all the keypoints, display (1) the keypoints at the first frame (as green) and (2) the tracked keypoints at the second frame (as red) on the first frame of the sequence.

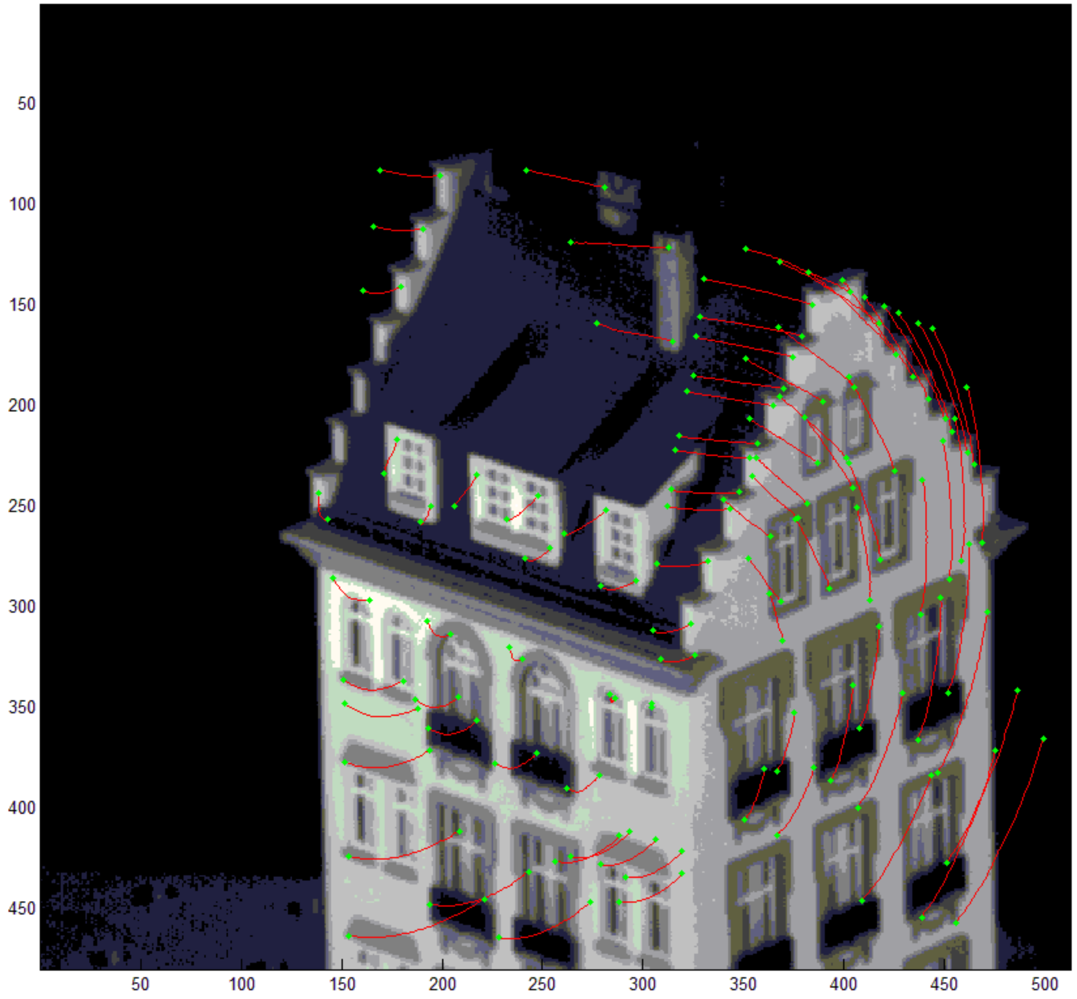
Green Points correspond to the keypoints in the first frame
Red Points correspond to the tracked keypoints in the second frame.

This image is the initial Frame.



Frame 1 -> Frame 50



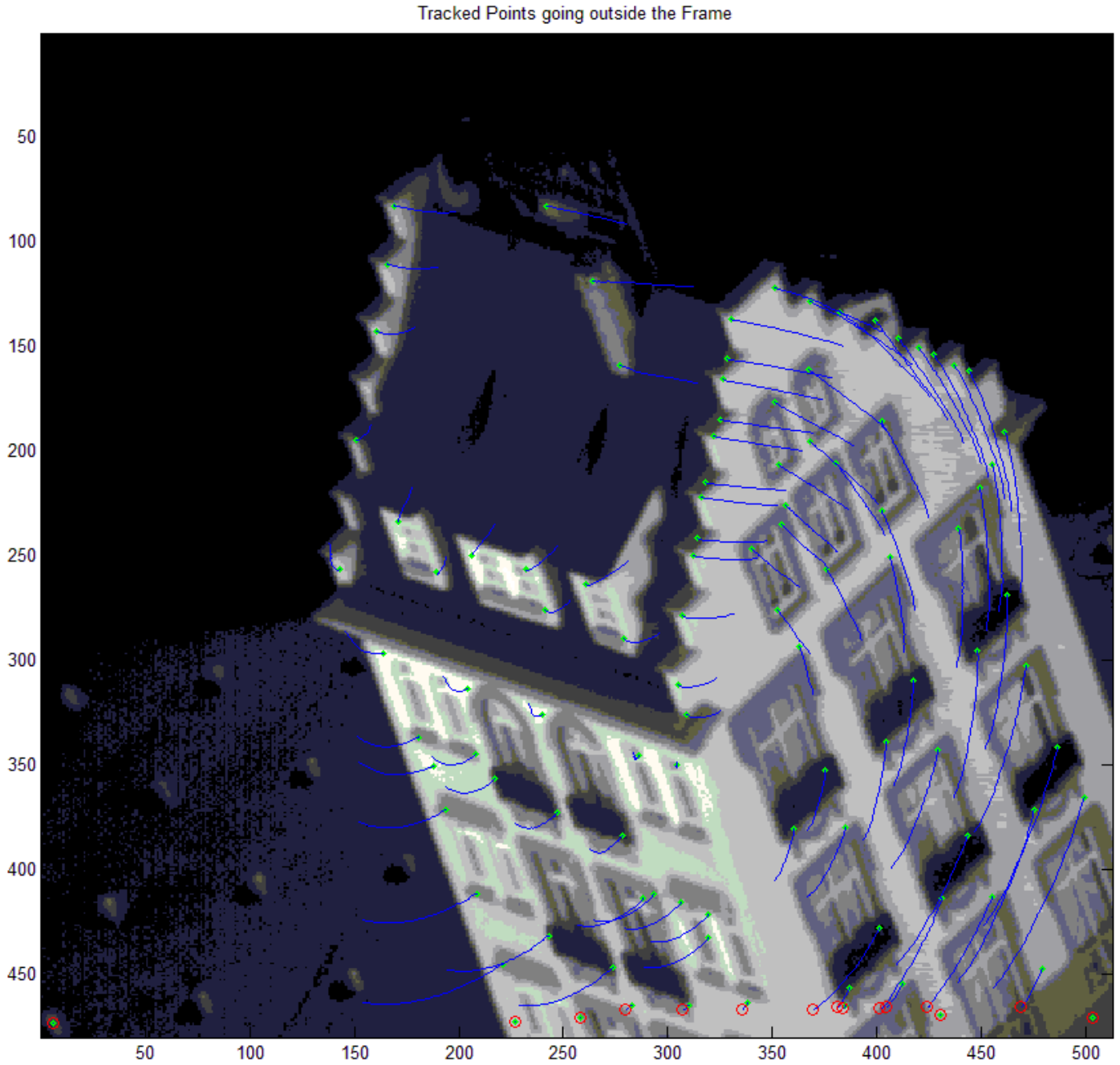


On top of the first frame, plot the points which have moved out of frame at some point along the sequence.

Green -> Initial keyPoints from image0

Blue -> Tracked keyPoints through frame0-50

Red -> Points which have gone out of the frame during tracking.



Problem 2: Shape Alignment

- Algorithm used:

  I have used an ICP algorithm with least squares for shape alignment

1. Compute the translation, scaling and rotation from image 1 to image 2 to create a Transformation Matrix, T which is used as a good initialization measure.
2. Convert image 1 key points using T
3. Using L-2 norm to find the closest points from the transformed image to image 2.
4. Using the corresponding points, find the affine transform by solving the equation

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

5. Find m1,m2,m3,m4,t1,t2
6. NewT = T*[m1 m2 t1;m3 m4 t2;0 0 1] -> to compute the new transform matrix
7. Repeat step 3-6, until maximum convergence. (Least squares method)

- Initialization
  - Use the mean of the image to get the translation
  - Use the standard deviation to get the scaling
  - Use the eigen vectors of the variance to get the rotation
  - Translation ->

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

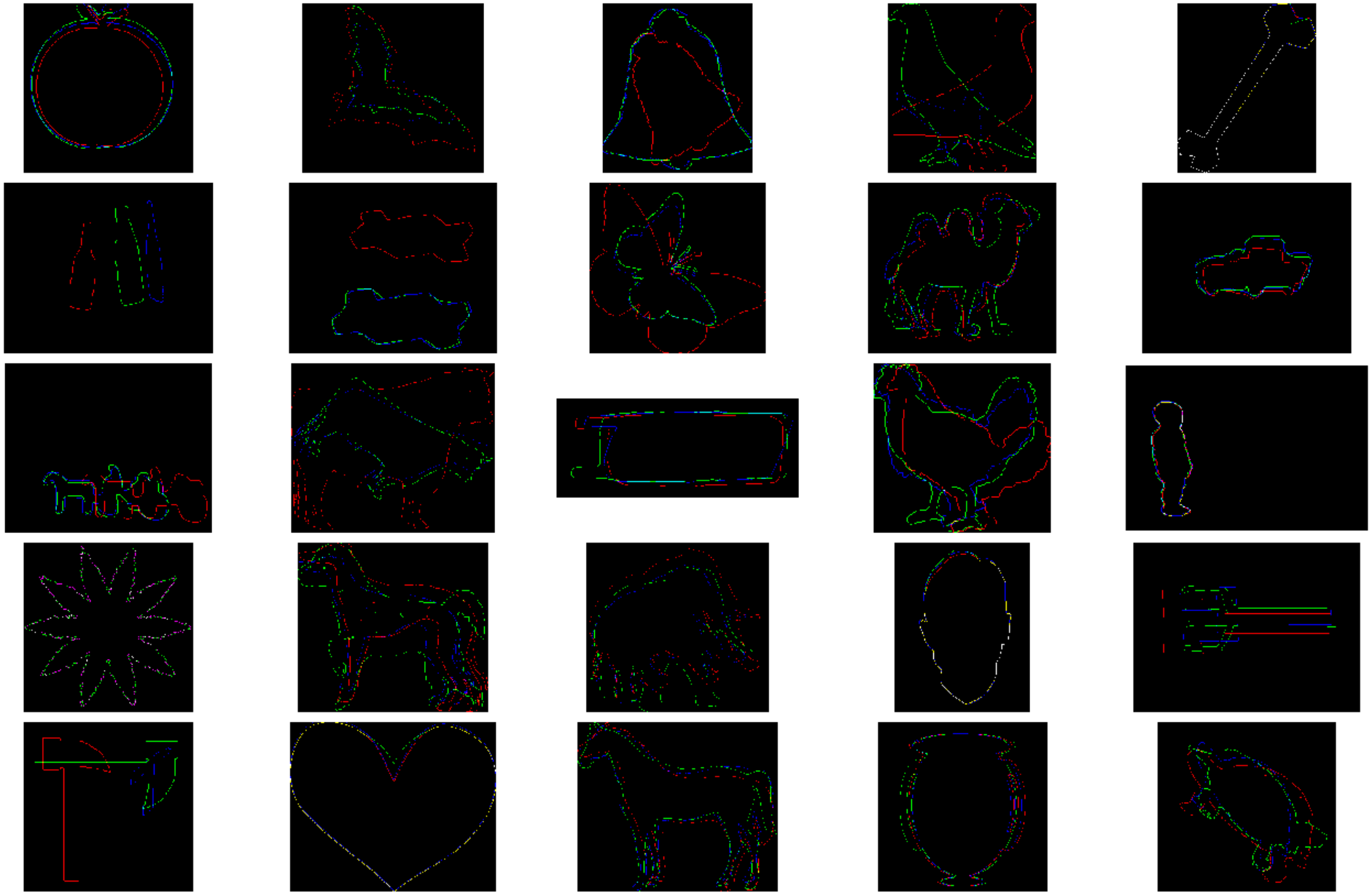
  - Rotation ->

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

  Where theta = theta2-theta1(from image2 and image1's eigen vectors)

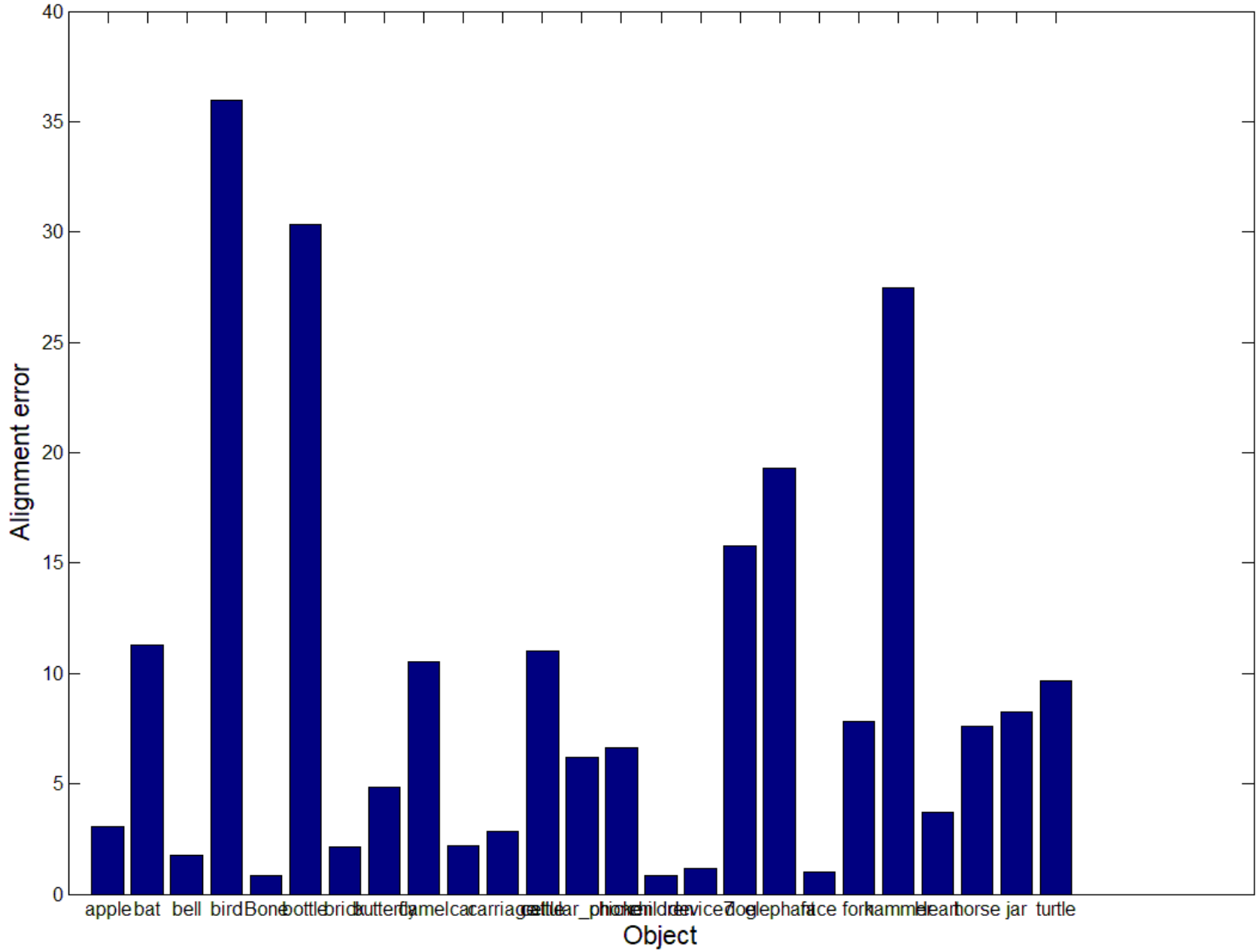
- Model
  - We have used an affine transform, even though the matrix T is a 3x3 dimensioned, the bottom row is always [0 0 1], hence we are not warping.

Red -> Image 1
Blue -> Image 2
Green -> Transform



Good cases -> Apple, Bell, Butterfly, children, face, fork

Bad Cases -> hammer (orientation issues)
Bird (Flipped Image, needs better initialisation)
Bottle (Rank and Condition of the transformation matrix issues, can be solved using lesser number of iterations to get a denser matrix.)


Alignment error
Object
0
5
10
15
20
25
30
35
40
apple
bat
bell
bird
Bone
bottle
bricks
butterfly
camel
car
carriage
cattle
dolphin
fork
hammer
heart
horse
jar
turtle

Elapsed time is 0.158766 seconds.
Error for aligning "apple": 3.043444
Elapsed time is 0.772311 seconds.
Error for aligning "bat": 11.279076
Elapsed time is 0.107247 seconds.
Error for aligning "bell": 1.755130
Elapsed time is 0.310239 seconds.
Error for aligning "bird": 35.966709
Elapsed time is 0.356721 seconds.
Error for aligning "Bone": 0.864518
Elapsed time is 0.027540 seconds.
Error for aligning "bottle": 30.321011
Elapsed time is 0.116633 seconds.
Error for aligning "brick": 2.137177
Elapsed time is 0.314929 seconds.
Error for aligning "butterfly": 4.862310
Elapsed time is 0.527191 seconds.
Error for aligning "camel": 10.521843
Elapsed time is 0.106424 seconds.
Error for aligning "car": 2.183107
Elapsed time is 0.173127 seconds.
Error for aligning "carriage": 2.824559
Elapsed time is 1.718400 seconds.
Error for aligning "cattle": 10.993303
Elapsed time is 0.254183 seconds.
Error for aligning "cellular_phone": 6.220031
Elapsed time is 0.169589 seconds.
Error for aligning "chicken": 6.643210
Elapsed time is 0.081180 seconds.
Error for aligning "children": 0.839553
Elapsed time is 2.978270 seconds.
Error for aligning "device7": 1.175451
Elapsed time is 1.319479 seconds.
Error for aligning "dog": 15.768100
Elapsed time is 2.459379 seconds.
Error for aligning "elephant": 19.307802
Elapsed time is 0.170811 seconds.
Error for aligning "face": 1.007701
Elapsed time is 0.878260 seconds.
Error for aligning "fork": 7.831508
Elapsed time is 0.133916 seconds.
Error for aligning "hammer": 27.452896
Elapsed time is 0.478326 seconds.
Error for aligning "Heart": 3.713124
Elapsed time is 2.589780 seconds.
Error for aligning "horse": 7.603043
Elapsed time is 0.948099 seconds.
Error for aligning "jar": 8.271256
Elapsed time is 0.279771 seconds.
Error for aligning "turtle": 9.640903
Averaged alignment error = 9.289071
Elapsed time is 18.638936 seconds.

Problem 3: Feature Matching

- Plot matches using nearest neighbor

-> It gives us lots of matches with lots of false positives, this is because those points are not reliably matched.

- Plot Matches with distance ratio test

-> Better results as lots of false positives are discarded since we are thresholding those points.

Nearest Neighbor matching with normalized threshold on distances = 0.7

Nearest Neighbor matching with normalized threshold on distances = 0.8

Distance Ratio Test with thresholding for 0.7

Distance Ratio Test with thresholding for 0.5

Graduate Points

- 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.

I am using the k-d tree model provided by MATLAB in createns function for nearest neighbor search.

```
mdl = createns(Descriptor1','NSMethod','kdtree');      // This creates the object which contains the k-d tree
in = knnsearch(mdl,Descriptor2','K',2);                // Uses the tree,to find the smallest distances (&indexes in the desc2)
```

I also used the in-built function pdist2 for finding the nearest neighbor.

```
[D, I] = pdist2(Descriptor2',Descriptor1','euclidean','Smallest',2);   // D-> distances I-> Indexes
```

Elapsed time is 0.637881 seconds. -> This time is reported when individual distances are calculated

Elapsed time is 0.541532 seconds. -> This time is reported when using the k-d tree

Elapsed time is 0.190694 seconds. -> This time is reported using pdist2 function, which is optimized for finding k nearest neighbours.

- 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.

I checked the correlation of the image1 with image2, with image1 having gone through various transforms

1. Horizontal Flip
2. Vertical Flip
3. Horizontal and Vertical Flip
4. Rotation through 90,180,270 degrees

Pros of the technique ->

Averaged alignment error = 5.308495 (Reduced from 9.7)

Cons of the technique ->

Increased the total time of execution!

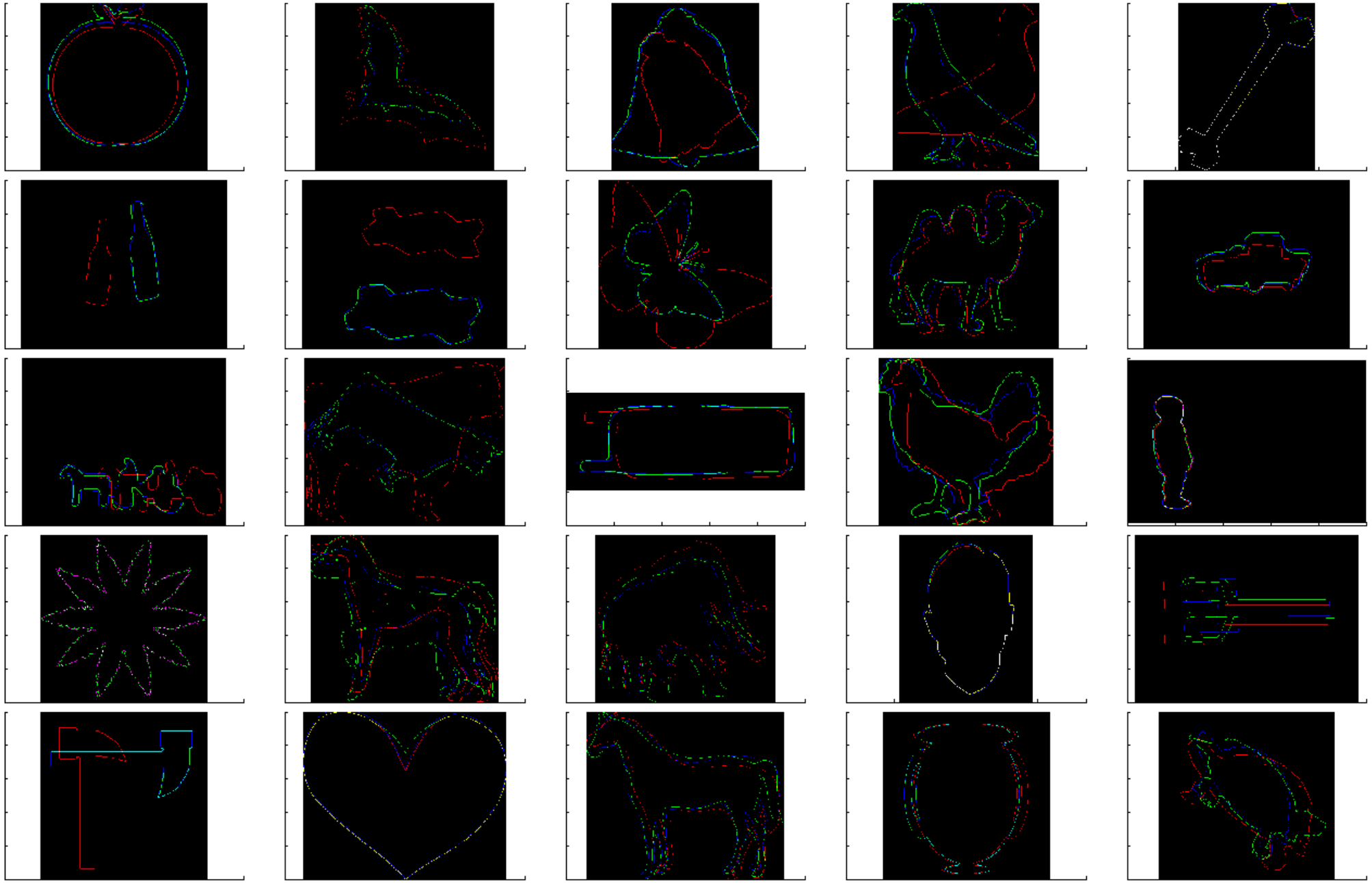
Improved Results for:-

Bottle – Since horizontal flip makes the matching better

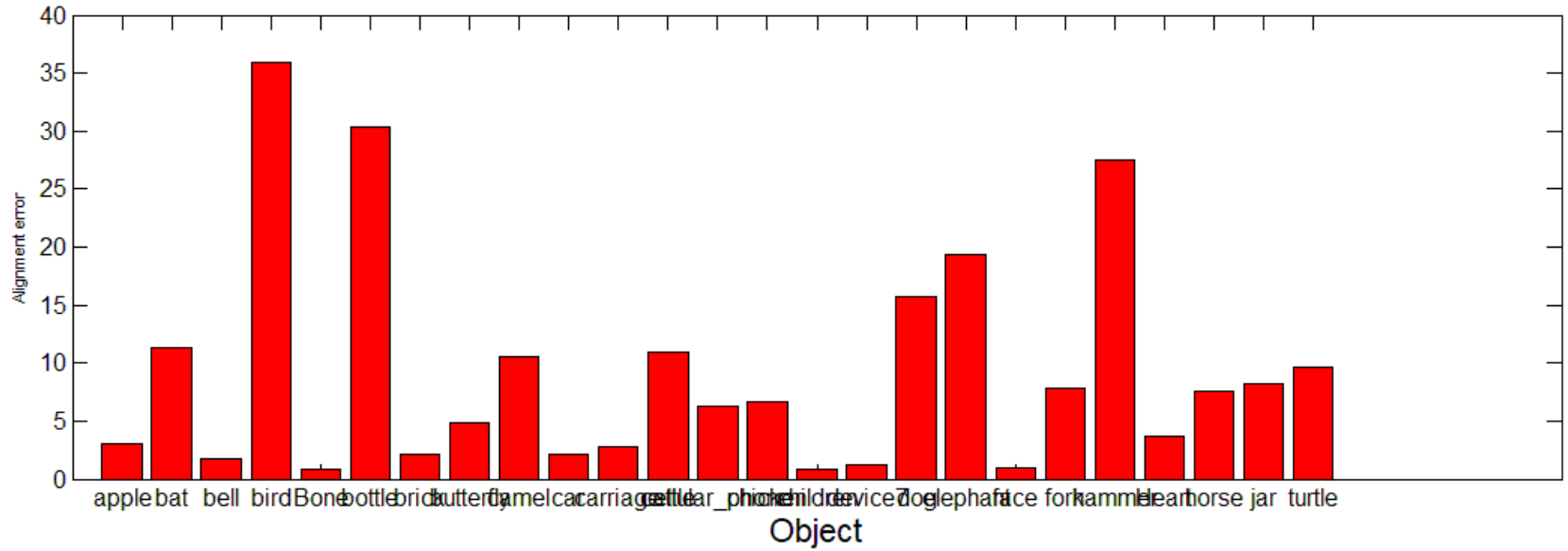
Bird –Horizontal Flip

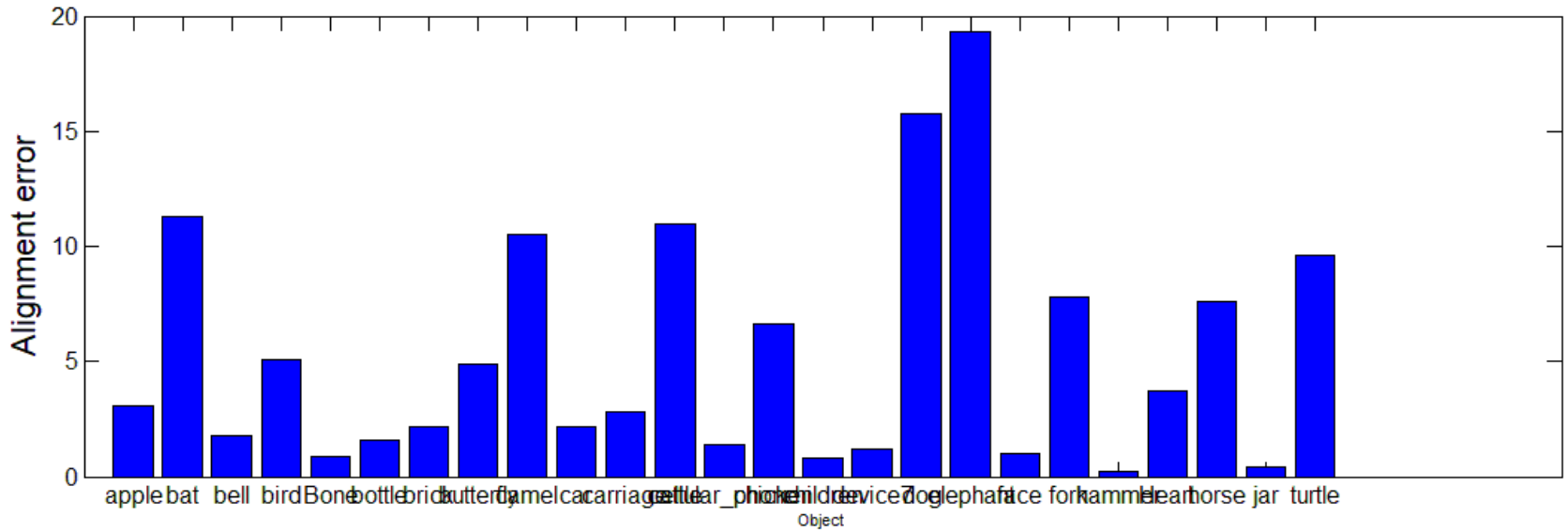
Hammer – rotate by 270

Cellular phone – Vertical fllip



Improved Results!!





- 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.

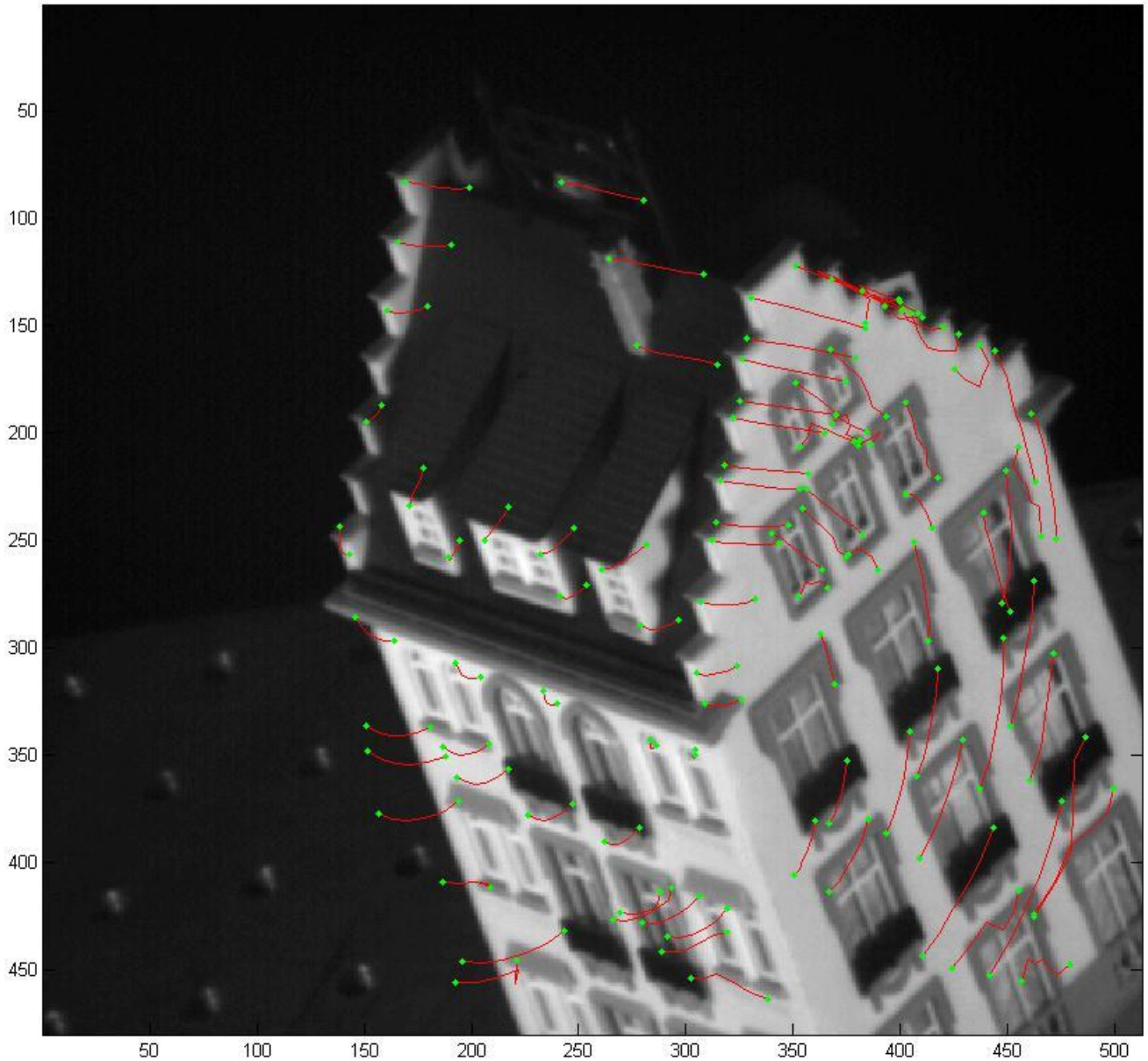
Algorithm

1. Create a gaussian level3 pyramid
2. Generate the KLT tracked points at the coarsest levels
3. Use this as initialization for level 2 and then level1 (essentially you are up sampling them.)
4. Use only 0,10,20,30,40,50 th frames instead of using the entire range of images specified.

Fine Tracking – Single level tracking

Time taken :
Elapsed time is 4.364445 seconds.
Elapsed time is 4.156424 seconds.
Elapsed time is 4.098362 seconds.
Elapsed time is 4.075506 seconds.
Elapsed time is 3.992739 seconds.
Total Time ~ 20 seconds and the tracking is not so accurate in some places.

It is not so good in detecting larger motions.



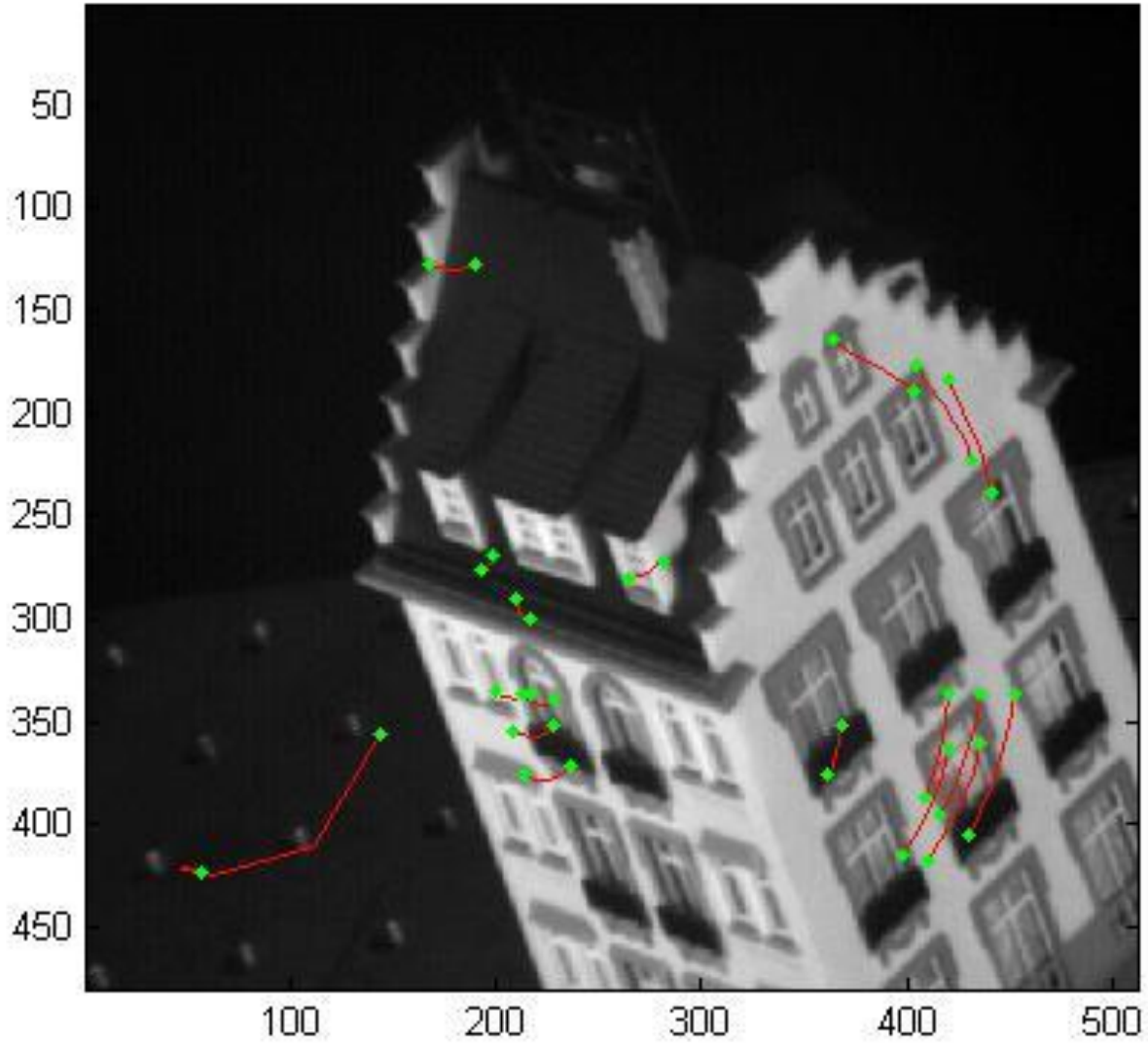
Coarse to Fine Tracking – Multi scale tracking

The time taken for this is:-

Elapsed time is 0.187034 seconds.
Elapsed time is 0.175460 seconds.
Elapsed time is 0.165279 seconds.
Elapsed time is 0.154288 seconds.
Elapsed time is 0.146914 seconds.

Elapsed time is 2.168882 seconds.
Elapsed time is 2.173493 seconds.
Elapsed time is 2.162417 seconds.
Elapsed time is 2.151538 seconds.
Elapsed time is 2.143115 seconds.
Total Time -> 15 seconds

This is a faster and also produced a much better result as both large scale and the small scale motions are tracked.



- 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.

My implementation:

1. I subsampled the image from (1:10:size(im{1},1)) and then created an array of pt_x and pt_y.
2. Passed this array to the trackPoints function and got the resulting changes in x and y directions
3. Saved these changes in another array:-

```
track_x_op = track_x(:,2)-track_x(:,1);
track_y_op = track_y(:,1)-track_y(:,1);
track_x_op = reshape(track_x_op(),size1,[]);
track_y_op = reshape(track_y_op,size1,[]);
```

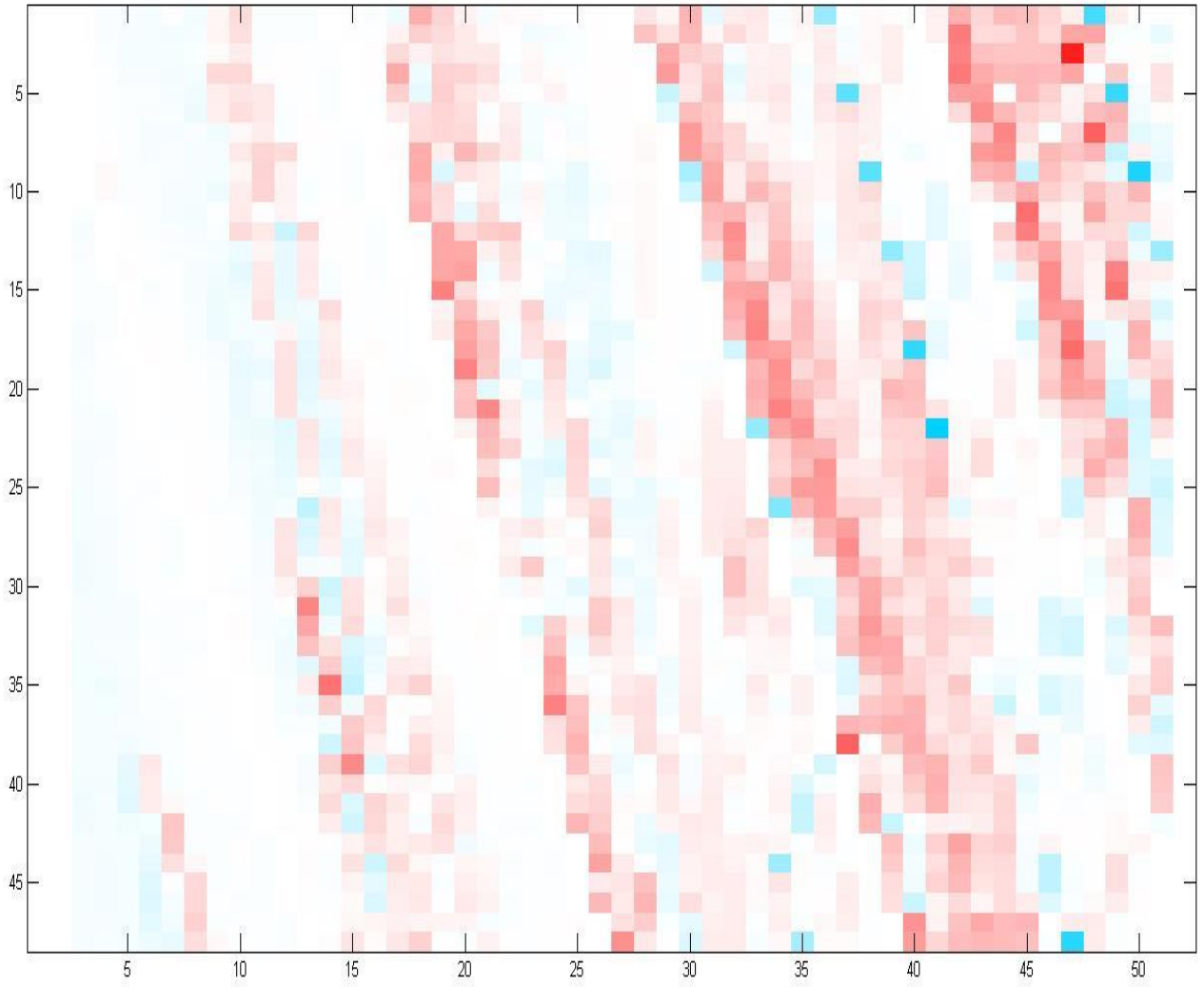
4. Created the H*W*2 array representing the Ix and the Iy gradients:-

```
tracker(:,:,1) = track_x_op;
tracker(:,:,2) = track_y_op;
```

5. Passed this to the flowToColor function and did an imagesc on the response which shows the optical flow.

Results as shown from the flowToColor function.

This comparison is between frame 0 and frame 10.



- 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.

First Image → $(x1, y1, w1, h1)$

Keypoint in image1 → $(u1, v1, s1, theta1)$
Keypoint in image2 → $(u2, v2, s2, theta2)$

w1 & h1 can be determined by the scaling

$$w2 = \frac{s2}{s1} w1$$

$$h2 = \frac{s2}{s1} h1$$

& let us define

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

x_2 & y_2 can be determined by using rotation matrix & then scaling & translating the box.

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

pt2 = Translation + Scaling x Rotation x [pt1]

$$[x_2, y_2] = (u_2, v_2) + s2/s1 * R * [x_1 - u_1, y_2 - v_2]$$