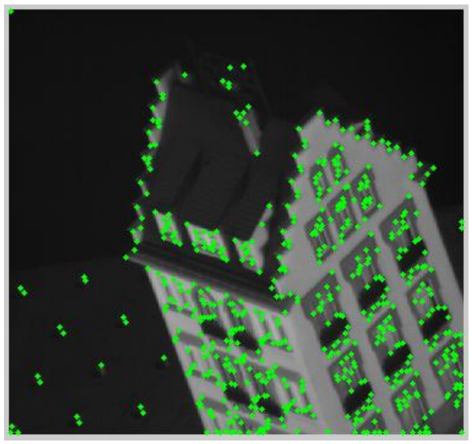
Seylom Ayivi HW2
CS 543 Computer Vision

1. **A Feature Tracker**

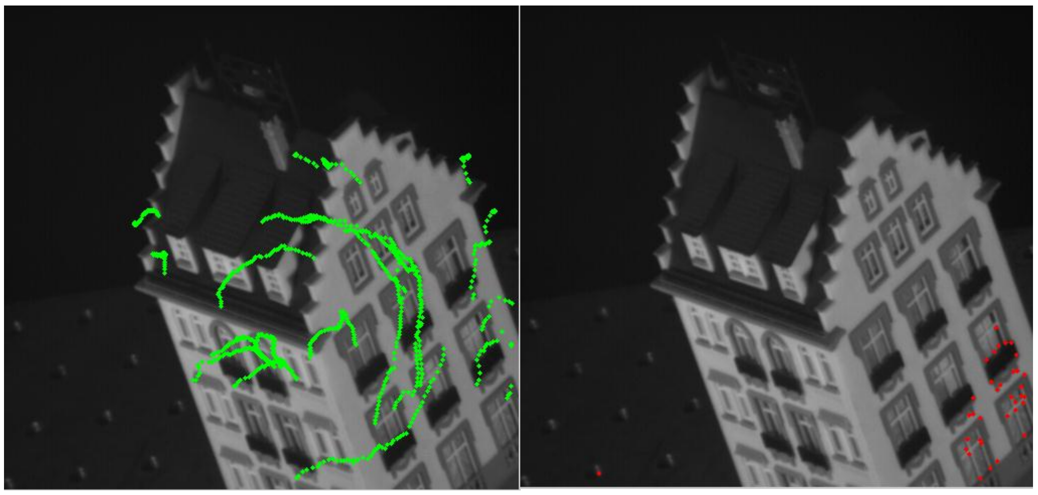
1.1 Keypoints Selection:

I am using the Harris criteria to find the features to be tracked. My steps are listed below:

- Convert the input image to the 0-1 range using im2double
- Perform a smoothing of the image with a Gaussian filter. Sigma = 1
- Compute the first order derivatives of the image by convolving the image with [1 0 -1] and [1 0 -1]'.
- Compute the square and product of the derivatives of the image obtained.
- Compute a new matrix of the same size as the image and compute the Harris criteria for each of the points in the image matrix.
- Using the input tau to threshold keypoints. In my implementation tau = 3. 1e-7; every point in my Harris criteria matrix with less than this value is nullified.
- Use ordfilt2 to select maximums in a 5 by 5 window.
- Use the result of the previous operation to nullify center pixels that were not the maximum of the 5 by 5 window. **im_res(im_res<ord_res) = 0** where im_res is the matrix resulting from the calculation of the Harris criteria at each point and ord_res the result of the ordfilt2 operation.
- Use the find method to return non zero elements of the matrix (row and column are returned)
- Overlay the points on the original image using a combination of *imshow*, *hold on* and *plot*



1.2 Tracking



The above figures show the path of 20 randomly selected points and only a few points lost during tracking. The reason I obtained so few lost point is probably because I don't process points with tracking window falling off the edge of the image in order to avoid interpolation issues.

Algorithm:

Since we only know the initial position of the tracked points from the first image feature extraction, for each point being tracked from frame to frame we evaluate what its displacement is, infer its new coordinates and use that information for subsequent frames.

My implementation takes easily 20 seconds on my computer to complete the calculation of all my 500 points in order to know which of the points are lost during the tracking. I was unable to vectorize the code for that section.

I also modified the suggested structure to include an additional return value that specifies if the point is still active (or lost). It is just a 0-1 value :

[newX newY active] = predictTranslation(…)

2. Shape Alignment

1. First algorithm (ICP with linear least square):

In order to align the two set of points:

- I start by an initialization phase consisting on aligning the centroids of both images. The centroids are just computed as the average of the non-zeros point coordinates. I then apply the difference between the two centroids to the second image.
- I use the Iterative Closest Point algorithm to progressively align both set of points. In the loop of the ICP algorithm, I first run **bwdist** on the initial image, retrieve the nearest neighbor coordinate and use the resulting pixel as a point match for the points in the second image. So for each of my non-zeros point in the second image, I find what the nearest point would be if that pixel was on the first image. The result is therefore 2 sets of points that I feed to my shape_align function.
- For the provided set of points we compute the transformation matrix that best maps the first set to the second set. Using homogenous coordinates we have

  $$\begin{pmatrix} U \\ V \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & tx \\ c & d & ty \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix}$$

  Where [u v 1] correspond to the result of applying the transformation to [x y 1]
  I chose this affine representation of the transformation and solve the system using Matlab built-in pseudo-inverse function.

- We apply the transformation T to the set of point and repeat the process of finding nearest neighbors for the result of the transformation. Applying again **bwdist** provides a new set of matching points to be fed to the algorithm.

I would like to note that instead of solving the system by determining the parameters of the transformation matrix, I just let Matlab compute the pseudo inverse of the vector [x y 1] in order to determine T (the transformation matrix).

⇨ $\begin{pmatrix} U \\ V \end{pmatrix} = \begin{pmatrix} a & b & tx \\ c & d & ty \end{pmatrix} \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix}$

⇨ Let A = $\begin{pmatrix} a & b & tx \\ c & d & ty \end{pmatrix}$ ➔ $\begin{pmatrix} U \\ V \end{pmatrix} = A * \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix}$

Apply transpose and compute pseudo-inverse. $A^t = (X \quad Y \quad 1) \backslash \ (U \quad V)$

Below are the Image alignments images. Run the file **runThis.m** to see the results

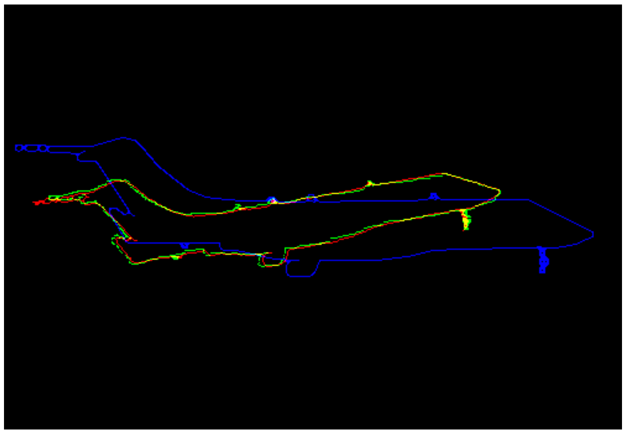


Figure 1 Object2t with Object2

Final Error: 1.050513
Runtime: 0.026833 seconds

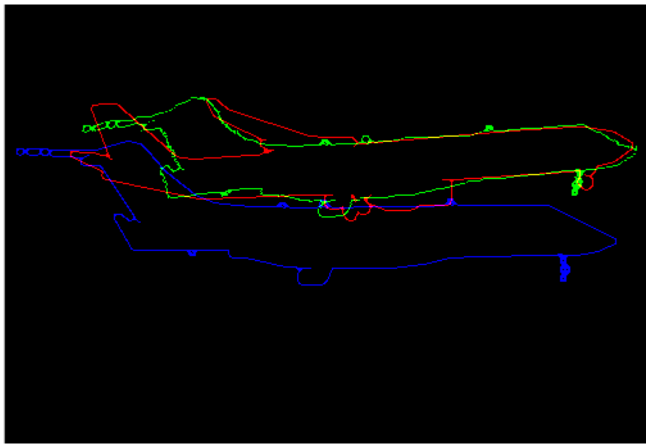


Figure 2 Object 2 with Object 1

Final Error: 5.299555
Runtime: 0.025040 seconds

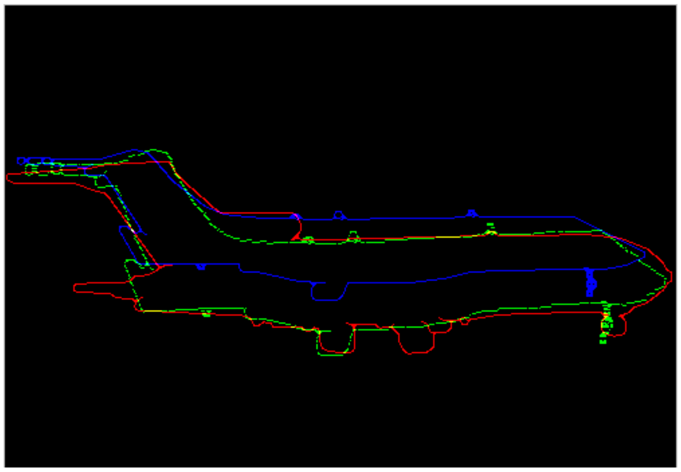


Figure 3: Object2 with Object 3

Final Error: 4.818236
Runtime: 0.026446 seconds

My ICP algorithm runs about 50 iterations and converges before reaching the 50 mark. I suspect my above-pixel error may be related to the basic least square algorithm.

2- Second Algorithm: ICP with RANSAC

The algorithm is identical to the first one except that instead of using the matrix obtained by solving our linear equation system, we use it in combination with RANSAC to evaluate how good it fits points. We run RANSAC with 3 = repetitions and threshold = 8. This combination seems to yield the best results

The file to run is called **runThisRansac.m**

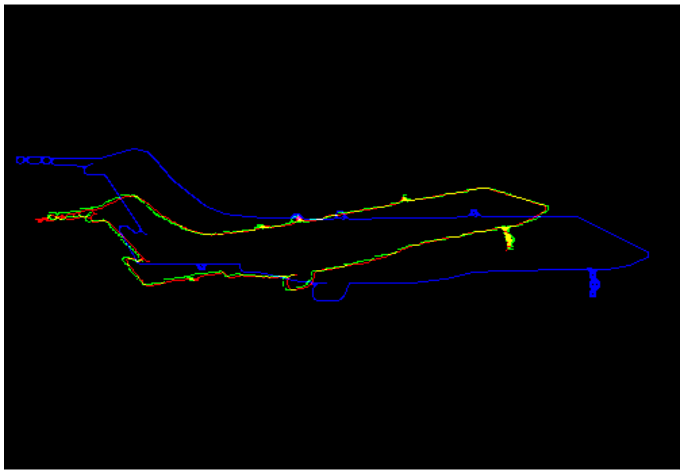


Figure 4: RANSAC object 2 to Object 2t

Final Error: error=0.871698

Runtime: 0.911044 seconds

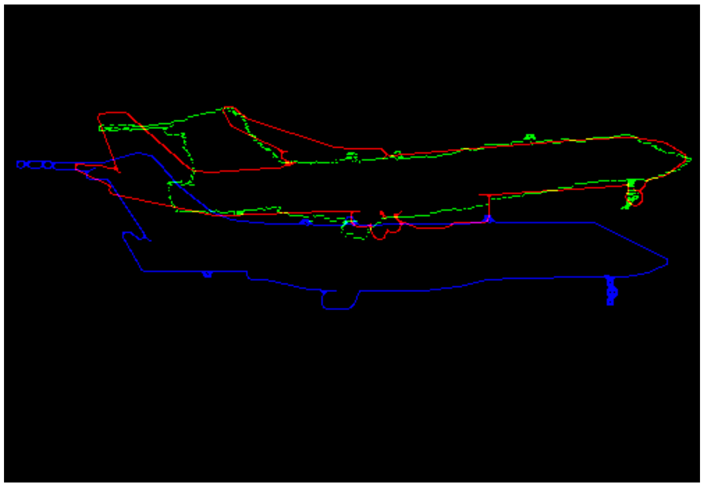


Figure 5: Object2 to Object 1

Final Error: error=5.620542

Runtime: 0.934941 seconds

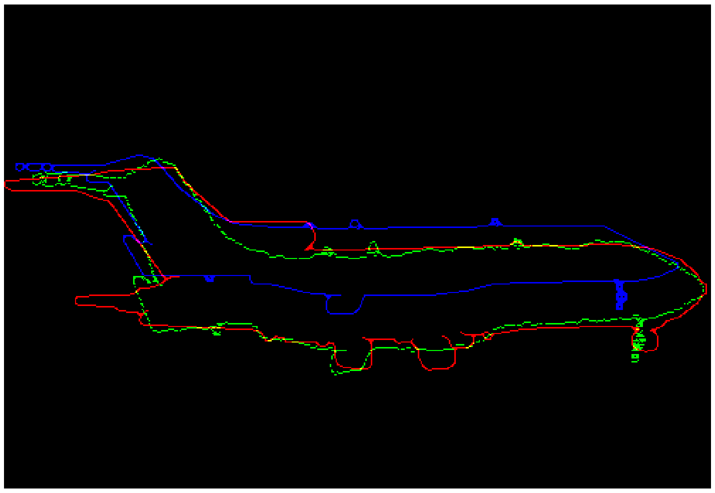


Figure 6 Object2 to Object 3

Final Error: error=4.643801

Runtime: 0.977866 seconds

3 – Object Instance Recognition.

3.1

Given a Keypoint descriptor G in order to determine possible matching keypoint descriptors in the second Image:

- Each of the feature descriptors votes for G's position scale and orientation. We use the Hough transform voting process to identify which of the matches can actually be kept by having cluster of features (possibly belonging to the same object) vote on parameters such as location, scale and orientation. The more features that cast votes for a particular set of parameter, the more likely the match.
- The set of keypoint votes are accumulated into bins and those with more than 3 votes are retained for further processing.
- After sorting the bins in decreasing order, we use bin's features cluster (using a hashtable) to determine a least square parameter affine transformation that mapped the initial object to the image points.
- We apply the transformation to the keypoints and verify that our matches' parameters are within a certain threshold and discard outliers. We repeat this process with the remaining points in the cluster until no more are lost.
- Bins that have less than three points are discarded and the bins with three points or more are considered a match and the related descriptors are kept.

3.2

The relative orientation of the object in the second image is
Ang = theta2 – theta1.
To compute the center position, we can first notice that the angle between an horizontal line and a line going through the center and the keypoint is the same.

Therefore we have the relation

$(Y - V1) /(X - U1) = (Y2 - v2)/(X2 - U2)$ (1)

Using the fact that the relative ratio between width and height for both object is the same we also have:
$H1/w1 = h2/w2$.
Using $w2 = x2/2$ and $h2 = y2/2$ we obtain:

$y2/x2 = h1/w1 \Rightarrow y2 = (h1/w1)*x2$ (2)

We can therefore substitute (2) in (1) and compute x2
From the knowledge of x2, the rest follows.