Contents

1. Introduction

Humans are quite good at matching different images and identifying items or people in images. If an image of an object is given to us, and it is required to identify this object in another image, most of the time it takes less than 1 second for us to identify that object. Somehow, when we see an object, we focus on some properties to create a model of it and then store that model in our minds. Then, every time we see that object we can easily connect that object with the model in our mind and therefore identify it. Moreover, we can accurately do so even when there exist several variations in orientation, scale, brightness, or viewpoint.

However, when it comes to satellite images and identifying specific areas / places in these images, people start struggling. The problem here is that the size of the target area in a satellite image may be so small (especially when the image is captured very far from the ground) that humans cannot find it. Moreover, when a satellite image is very dense (such as big cities), and there are no significant differences in that image, locating a specific area becomes difficult.

To overcome that problem, we can use image processing techniques to analyze images and extract the necessary information from them. Then, we can use that information to perform our tasks.

In this work, we have analyzed, implemented, and tested two different methods for image feature extraction and matching: Harris Corner and SIFT Matcher. The first goal is to test these two methods in general scenarios to decide which one is the best. Then, the best method is used to locate specific places in a satellite image.

2. Image matching process

The general idea of image matching is to extract some very descriptive unique features (interesting points) from different images and then match these features to find out if the images match or not.

Features can be specific patterns, edges, corners, textures, or any distinctive characteristics that help differentiate one image from another. A good, interesting point or feature is:

- Descriptive: The feature should contain relevant and discriminative information about the object or scene it represents.
- Robust: The feature should be robust to variations in scale, rotation, illumination, noise, and other common image transformations.
- Stable: The feature should exhibit stability in the presence of noise or occlusions.
- Local: The feature should capture local information within a limited region or

neighborhood of the image.

- Efficient: The feature extraction process should be computationally efficient, allowing for real-time or near real-time performance.

After we have found (extracted) some good features, we have to describe these features in order to have a way of comparing them. For that, we use a feature descriptor that gives each feature a kind of signature that describes the image appearance around the point/feature. The better the descriptor, the better the feature correspondence will be.

After we have extracted features and assigned them a descriptor, we can compare features from one image with features from the other image, using their descriptors, to find corresponding features and therefore to match the images.

To sum up, there are three main steps in the image matching process:

1. Extract good features from each image.
2. Describe the found features.
3. Find matching features between images.

3. **Harris detector with NCC**

3.1 Algorithm and implementation

The Harris corner detector, introduced by Chris Harris and Mike Stephens in 1988, is a popular method for detecting and locating corners or interest points in images. It operates by analyzing local intensity variations and identifying regions with significant changes in multiple directions. The algorithm follows these steps:

1. Convert the image to grayscale if it is not already in grayscale.
2. Gradient calculation to capture the intensity changes in the x and y directions, respectively.
3. Construct the structure tensor M: $M = \sum_{(x,y)\in W} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} = \begin{bmatrix} \sum_{(x,y)\in W} I_x^2 & \sum_{(x,y)\in W} I_x I_y \\ \sum_{(x,y)\in W} I_x I_y & \sum_{(x,y)\in W} I_y^2 \end{bmatrix}$
4. Calculation of Harris response (approximated) R: $\frac{\det(M)}{\mathrm{tr}(M)}$
5. Apply a threshold to the corner response values.
6. An additional constraint: Corners must be separated with a minimum distance.

The implementation is given below:

```
def compute_harris_response(im,sigma = 3):
  # Compute the Harris corner detector response function for each pixel
  # in a graylevel image.
  # Arguments are the image and the sigma value of Gaussian
```

```
  # 2. Derivatives of the image matrix as convolution process with
Gaussian filter
  imx = zeros(im.shape)
  filters.gaussian_filter(im, (sigma,sigma), (0,1), imx)
  imy = zeros(im.shape)
  filters.gaussian_filter(im, (sigma,sigma), (1,0), imy)
  # 3. Construct the structure tensor M
  Mxx = imx*imx
  Mxy = imx*imy
  Myy = imy*imy
  # determinant and trace
  Mdet = Mxx*Myy - Mxy**2
  Mtr = Mxx + Myy
  # 4. Return Harris response
  return Mdet / Mtr

def get_harris_points(harrisim,min_dist=10,threshold=0.1):
  # Input: Harris response matrix, min_dis, threshold
  # Output: List of Harris good points
  # 5. Find top corner candidates above a threshold
  corner_threshold = harrisim.max() * threshold
  harrisim_t = (harrisim > corner_threshold) * 1
  # get coordinates of candidates and their values
  coords = array(harrisim_t.nonzero()).T
  candidate_values = [harrisim[c[0],c[1]] for c in coords]
  # sort candidates
  index = argsort(candidate_values)
  # 6. Corners must be separated with a minimum distance
  # Store allowed point locations in array
  # start with a min distance from image constrains
  allowed_locations = zeros(harrisim.shape)
  allowed_locations[min_dist:-min_dist,min_dist:-min_dist] = 1
  # select the best points taking min_distance into account
  filtered_coords = []
  for i in index:
    if allowed_locations[coords[i,0],coords[i,1]] == 1:
      filtered_coords.append(coords[i])
      # Update allowed position for the next possible points
```

```
        allowed_locations[(coords[i,0]-min_dist):(coords[i,0]+min_dist),
        (coords[i,1]-min_dist):(coords[i,1]+min_dist)] = 0
    #return all harris good points (their cordinates in image)
    return filtered_coords
```

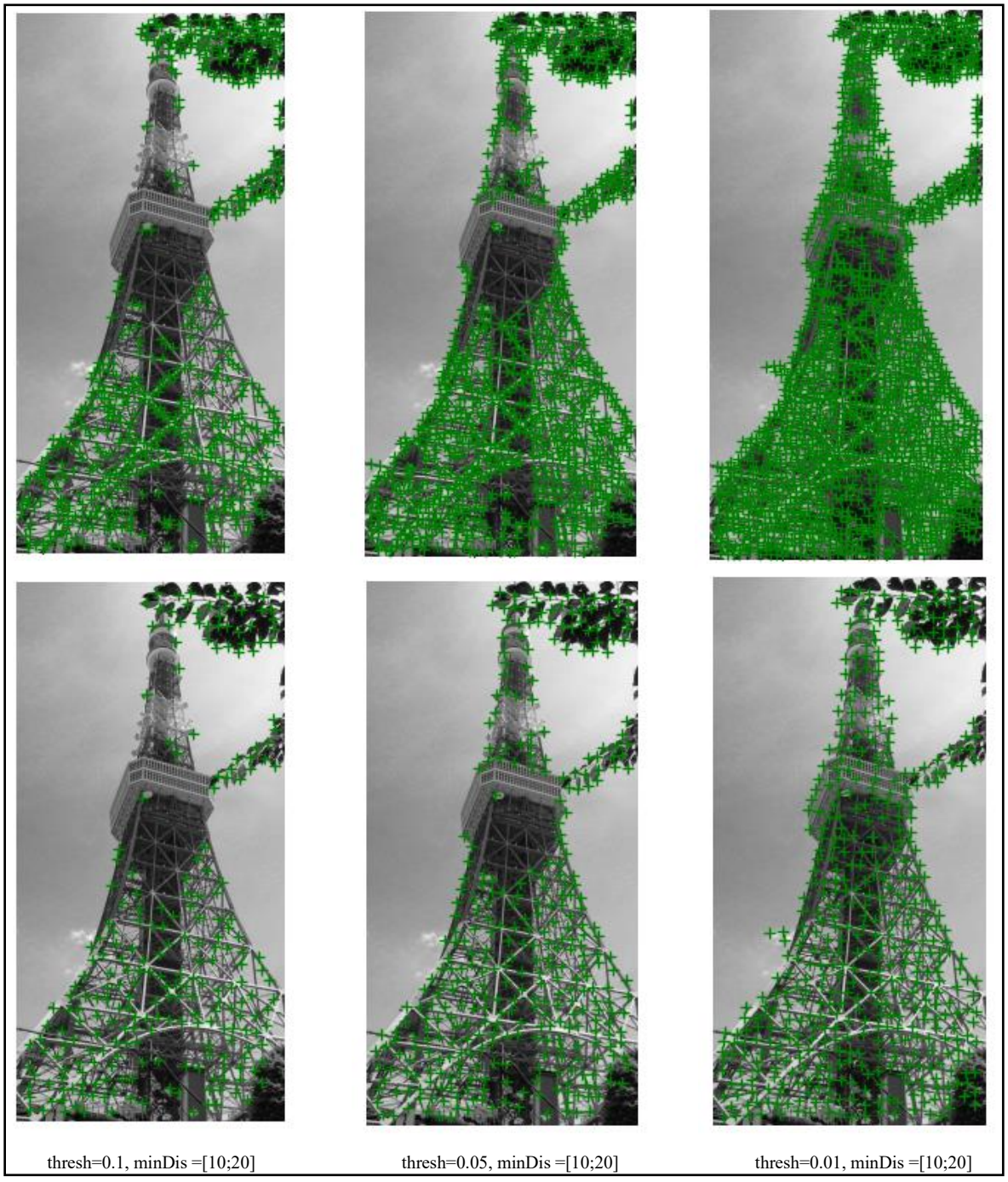

Fig.1 Changes in threshold value and min distance between corners.

As we can see from Fig. 1, when both threshold value and min distance are high, the number of detected corners is low and vice versa.

Now that we have extracted corners a descriptor is needed. Here a simple descriptor consisting of the graylevel values in a neighboring image patch is implemented as below:

```
def get_descriptors(image,filtered_coords,wid=5):
  # For each point return pixel values around the point
  # using a neighbourhood of width 2*wid+1. (Assume points are
  # extracted with min_distance > wid).

  desc = []
  for coords in filtered_coords:
    patch = image[coords[0]-wid:coords[0]+wid+1,
              coords[1]-wid:coords[1]+wid+1].flatten()
    desc.append(patch)
  return desc
```

Then, to compare features, NCC is used: $ncc(I_1, I_2) = \frac{1}{n-1} \sum_{\mathbf{x}} \frac{(I_1(\mathbf{x}) - \mu_1)}{\sigma_1} \cdot \frac{(I_2(\mathbf{x}) - \mu_2)}{\sigma_2}$ For better results, a two-sided match is performed, where we match from first to second, and from second to first. Matches that are not best both ways are filtered out.

```
def match(desc1,desc2,threshold=0.5):
  # For each corner point descriptor in the first image,
  # select its match to second image using
  # normalized cross correlation.

  n = len(desc1[0])
  # pair-wise distances
  d = -ones((len(desc1),len(desc2)))
  for i in range(len(desc1)):
    for j in range(len(desc2)):
      d1 = (desc1[i] - mean(desc1[i])) / std(desc1[i])
      d2 = (desc2[j] - mean(desc2[j])) / std(desc2[j])
      ncc_value = sum(d1 * d2) / (n-1)
      if ncc_value > threshold:
        d[i,j] = ncc_value

  # sorting matches (for one corner in image1) in ascending order
  ndx = argsort(-d)
  matchscores = ndx[:,0]

  return matchscores

  def match_twosided(desc1,desc2,threshold=0.5):
```

```
# Improve matching by comparing matches both ways
# Remove matches that are not the best both ways

matches_12 = match(desc1,desc2,threshold)
matches_21 = match(desc2,desc1,threshold)

ndx_12 = where(matches_12 >= 0)[0]
# remove matches that are not symmetric
for n in ndx_12:
  if matches_21[matches_12[n]] != n:
    matches_12[n] = -1

return matches_12
```

3.1 Tests – Harris detector with NCC

Three tests are conducted to evaluate the performance of the implemented method. First test, when there is no change in scale and no rotation. Second test, no change in scale, but rotation with 90°. Third test, change in scale, no rotation. The selected parameter values are:

Sigma of Gaussian	5
Corner threshold	0.05
Corner min distance	20
Neighboring image patch width	10
Matching threshold	0.8

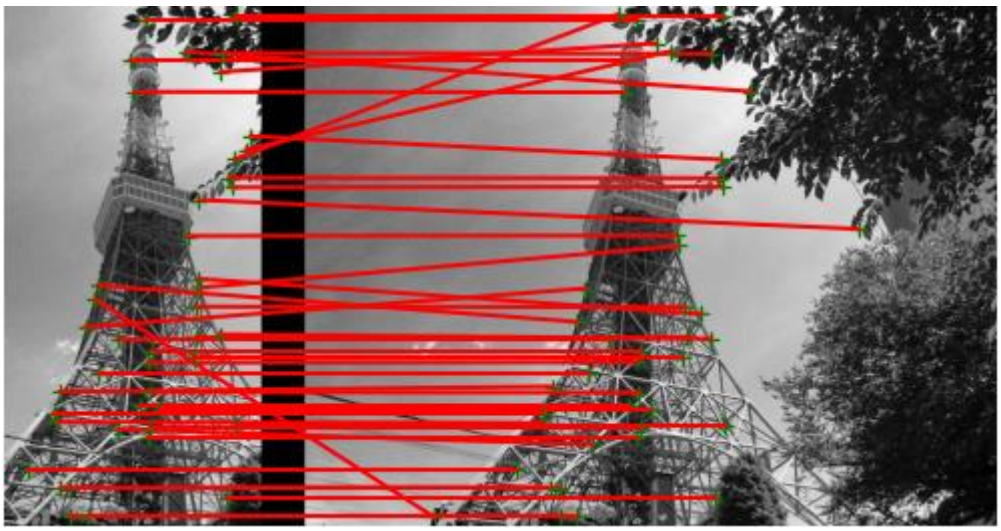

Fig. 2 No scale change, no rotation

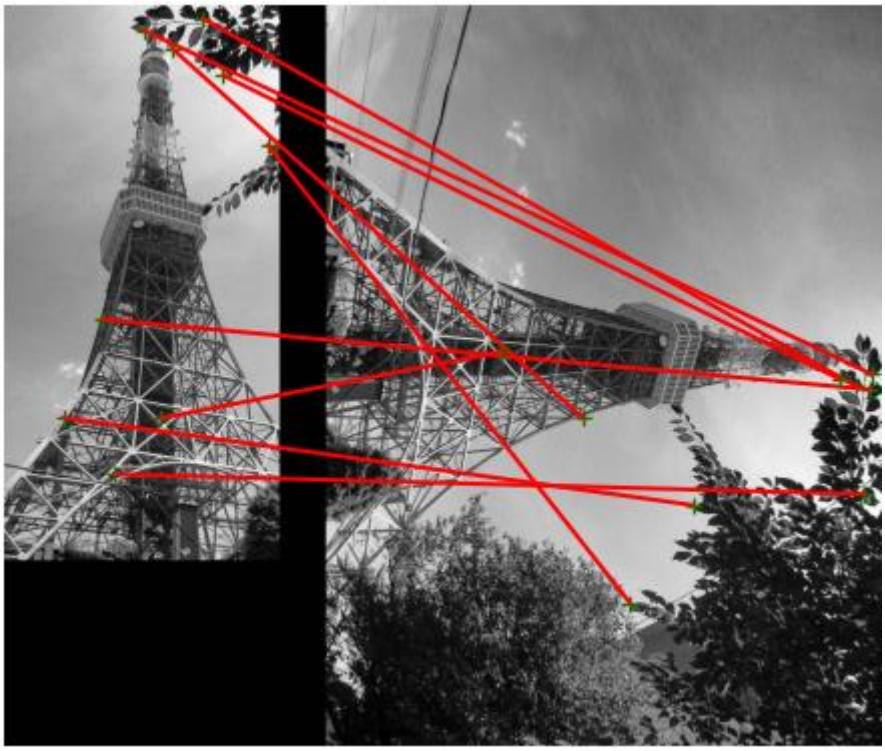

Fig. 3 No scale change, rotation with 90°

Fig. 4 Scale change, no rotation

From the first test we can see that, when there is no changes in scale nor rotation the algorithm performs well with only a few bad matches. On the other hand, when rotation is presented even though the algorithm can find some matches they are bad matches. For the case when changes in scale are presented the algorithm can not find any matches at all. Based on these result, we can confirm that Harris detector with NCC method is not scale invariant and rotation invariant. It makes this method not useful for practical applications, where there exists changes in scale and rotation.

4. SIFT matcher

4.1 Algorithm and implementation

The Scale-Invariant Feature Transform (SIFT) is a widely used method for extracting and matching robust features from images. It was introduced by David Lowe in 1999 and has become a fundamental technique in computer vision and image processing. The SIFT algorithm consists of the following key steps:

1. Scale-space representation and local extrema detection: At first, a pyramid of DoG images is constructed. Then, the DoG images are examined to detect local extrema, which correspond to potential keypoint locations at different scales.
2. Keypoint localization: The detected extrema are filtered by eliminating low contrast keypoints and keypoints along edges. A threshold value is applied here.
3. Orientation assignment: For each keypoint, an orientation is assigned to make the feature descriptor rotationally invariant. This is done based on a histogram of local gradient directions.
4. Keypoint Descriptor Calculation: A local image patch is divided into subregions or bins. Each subregion has its own histogram. The histograms from all subregions are combined to form the keypoint descriptor.
5. Keypoint Matching: Keypoints from different images are matched by comparing their descriptors. The distance between descriptors is calculated. Manhattan and Euclidean distances are used.

Different from the implementation of Harris detector where no important libraries are used, to implement SIFT matcher, OpenCV library is used. The implementation is as follows:

```
 # 1. Initializing the SIFT detector:
```

Here an object of SIFT class is created, using the member function create(). This class is used for extracting keypoints and computing descriptors using the SIFT algorithm by D. Lowe. The most important arguments are:

1. nOctaveLayers (3) -> How many layers we want in Image pyramid
2. contrastThreshold (0.04) -> Filters out weak features in low-contrast regions.The larger the threshold, the less features are produced by the detector.
3. sigma (1.6) -> The sigma of the Gaussian applied to the original image.

```
sift = cv2.xfeatures2d.SIFT_create(nOctaveLayers = 3, contrastThreshold
= 0.04, sigma = 1.6 )
```

```
# 2,3,4. Detecting keypoints and computing descriptors:
```

Next, the method detectAndCompute() is used to detect keypoints and their respective descriptor. Each keypoint is a special structure which has many attributes like its (x,y) coordinates, size of the meaningful neighbourhood, angle which specifies its orientation, response that specifies strength of keypoints etc.

Here keypoints_1 will be a list of keypoints from image 1, descriptors_1 is a numpy array of shape (Number of Keypoints) x 128.

```
keypoints_1, descriptors_1 = sift.detectAndCompute(img1,None)
keypoints_2, descriptors_2 = sift.detectAndCompute(img2,None)
```

```
# 5.1. Initializing the Brute-force matcher:
```

In this step a object of BFMatcher class is created using the member function create(). For each descriptor in the first set, this matcher finds the closest descriptor in the second set by trying each one (Brute-force descriptor matcher). The used arguments are:

1. normType -> How to compute the distances between every two features. NORM_L1 is the Manhattan distance, and NORM_L2 is the Euclidean distance.
2. crossCheck -> To check or not for the best matches in both ways. If B is the closest match for feature A, the tuple (A,B) is consider consistent pair only if A is also the closest match for feature

```
bf = cv2.BFMatcher.create(normType = cv2.NORM_L1, crossCheck = True)
```

```
# 5.2. Performing feature matching:
```

Next, the method match() is used to get the best matches in two images, using their descriptors. The distance between every two features (descriptors) is return. The lower, the better it is. Therefore, we sort them in ascending order to get the best matches first.

```
matches = bf.match(descriptors_1,descriptors_2)
matches = sorted(matches, key = lambda x:x.distance)
```

Fig. 5 In the left features extracted by Harris detector, and in the right features extracted by SIFT method

From Fig. 5, it is obvious that features extracted by SIFT method are more meaningful than corners. They are with specific directions (rotation invariant) and with different sizes (scale invariant).

4.1 Tests – SIFT

In this section, the same tests as in Harris detector with NCC are performed. The results are shown below:

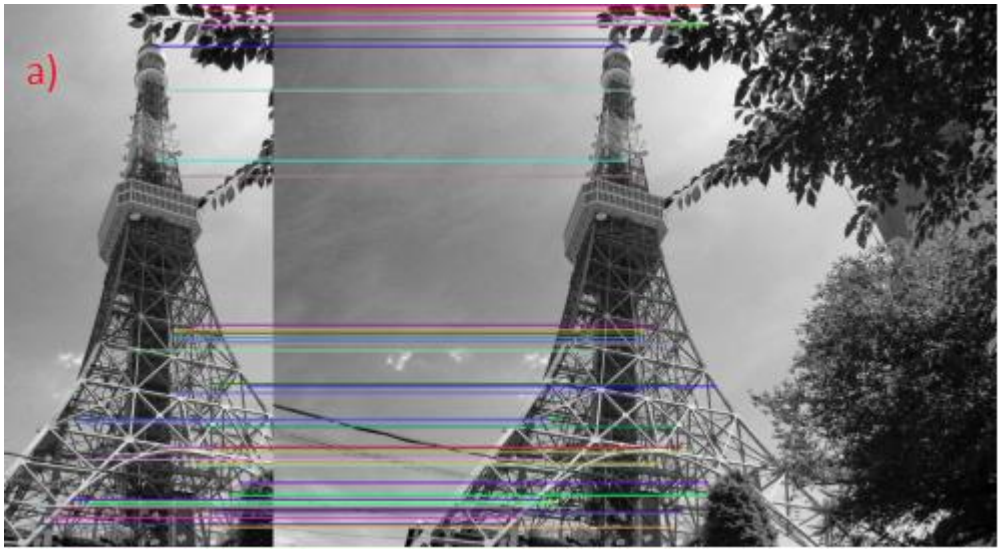

Fig. 6 No scale change, no rotation

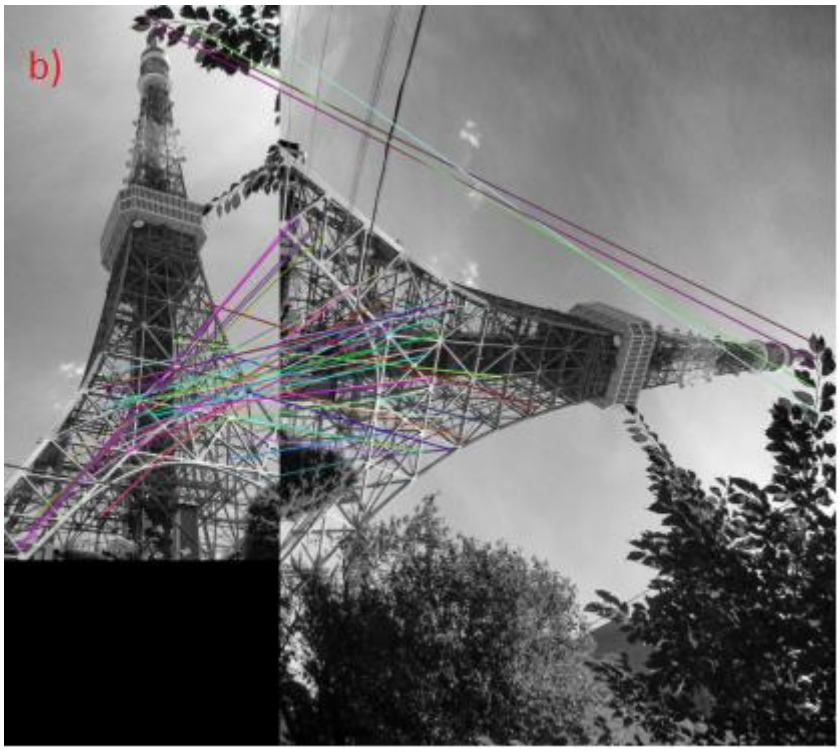

Fig. 7 No scale change, rotation with 90°

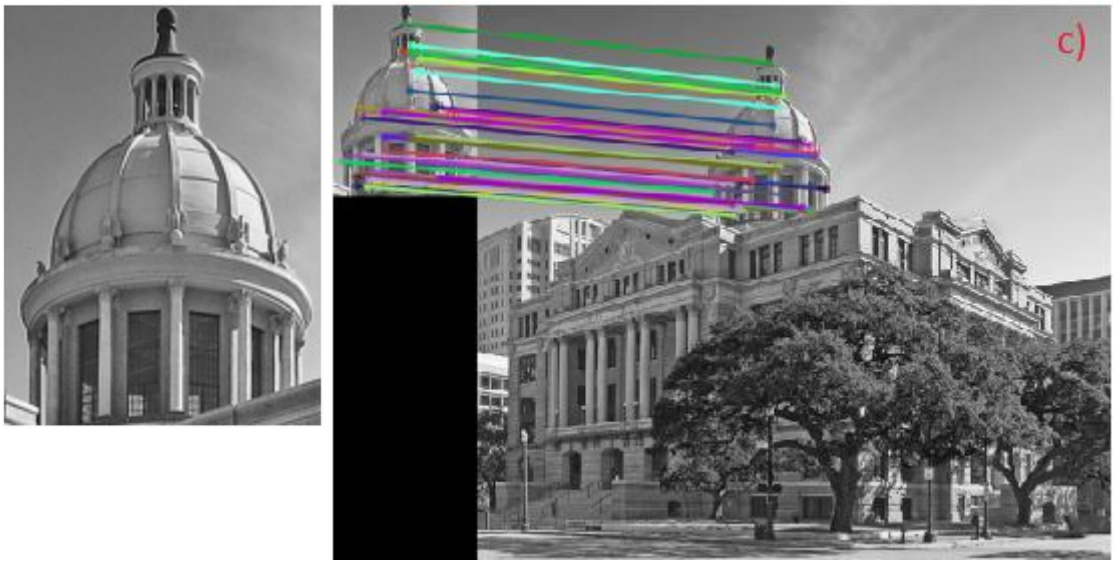

Fig. 8 Scale change, no rotation

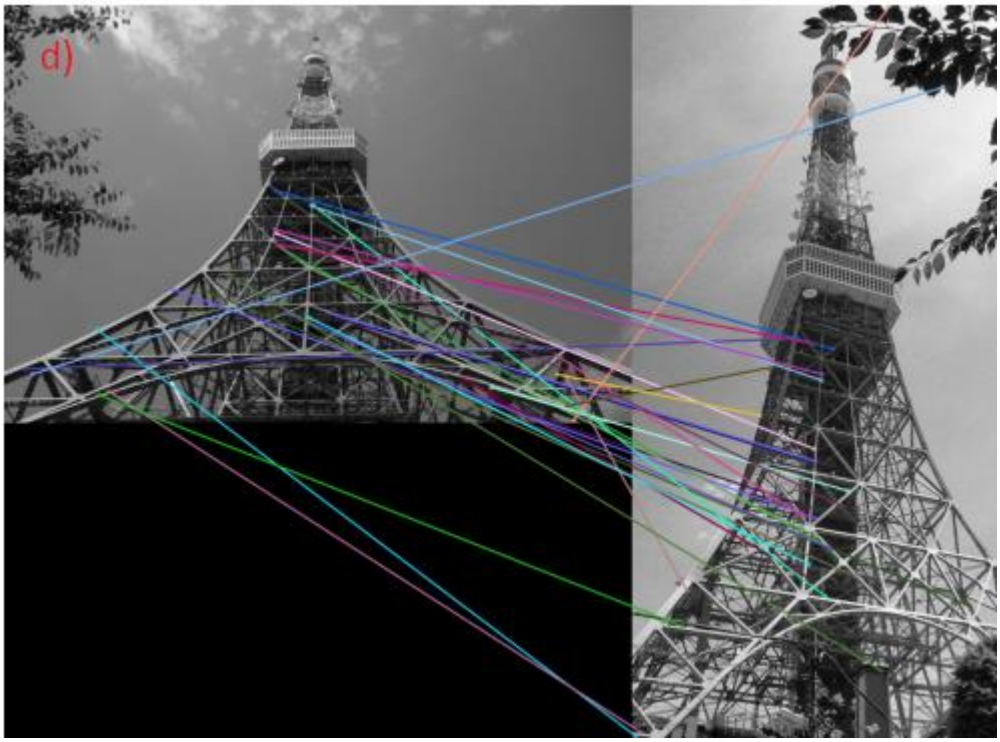

Fig. 9 Scale change, viewpoint change

These results show that SIFT features and the SIFT matching method are invariant to changes in scale and rotation. The performance of SIFT matcher is much better than Harris with NCC. Moreover, even when there exist some small changes in viewpoint, the SIFT matcher can still provide some good matches. The robustness of SIFT matcher, in scale and rotation, makes it suitable for practical uses. In the following section SIFT matcher will be used to identify airports and landing runways from aerial images.

5. Locating specific places in a satellite image with SIFT matcher

In this section, the task of SIFT matcher is to locate specific places in a satellite image. Places to be located are Haneda Airport and Narita Airport. The satellite image is from Google Earth, captured 134 km above sea level. The satellite image, Haneda Airport and Narita International Airport are shown below:

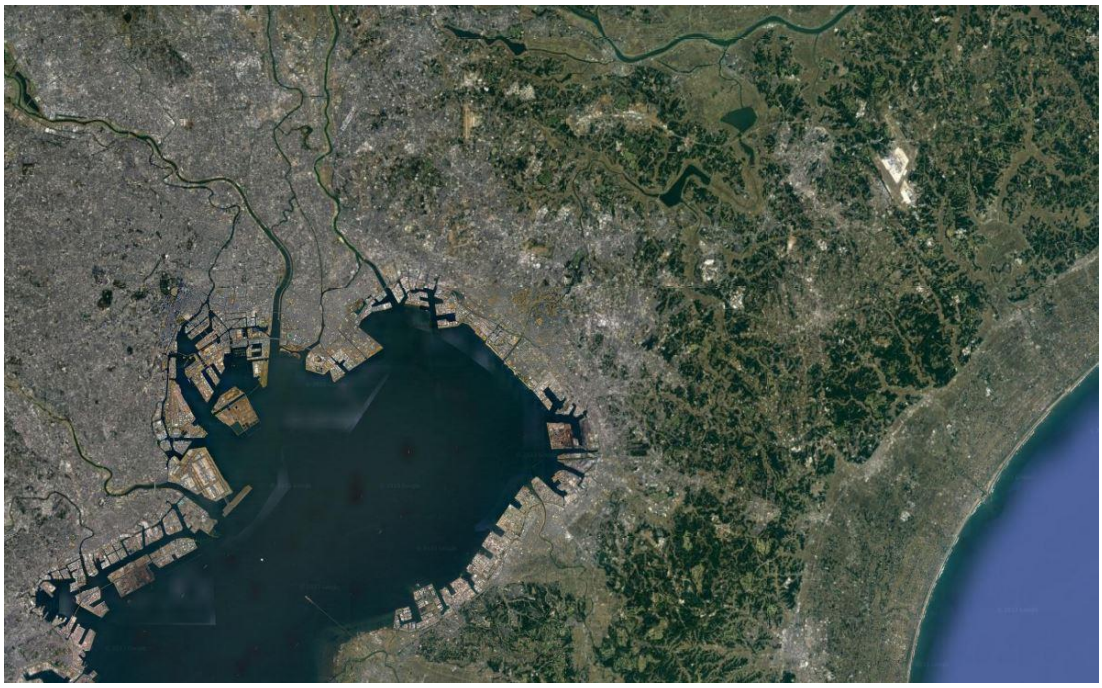

Fig. 10 Satellite image of Tokyo and Chiba from Google Earth, captured 134 km above sea level

Fig.11 Satellite images of: a) Haneda Airport, and b) Narita International Airport

At first, we have to extract SIFT features from images, and the extracted features for each pair of images are shown below:

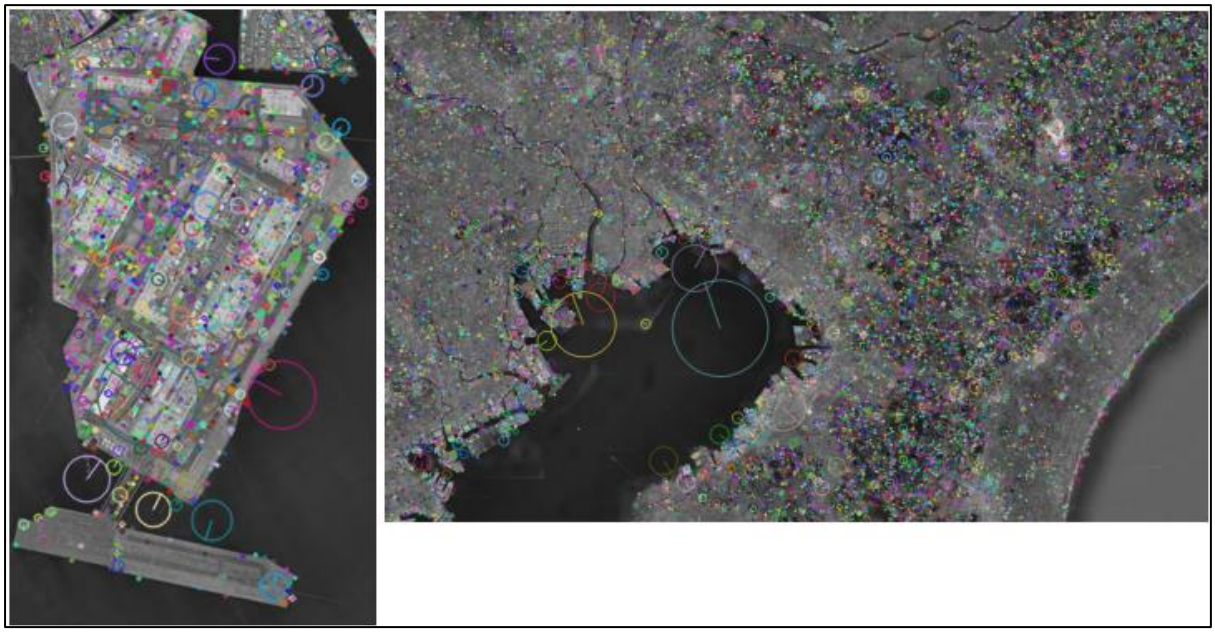

Fig. 12 Extracted SIFT features of Haneda Airport and the satellite image

After that, we perform feature matching, and the result of matching is shown below:

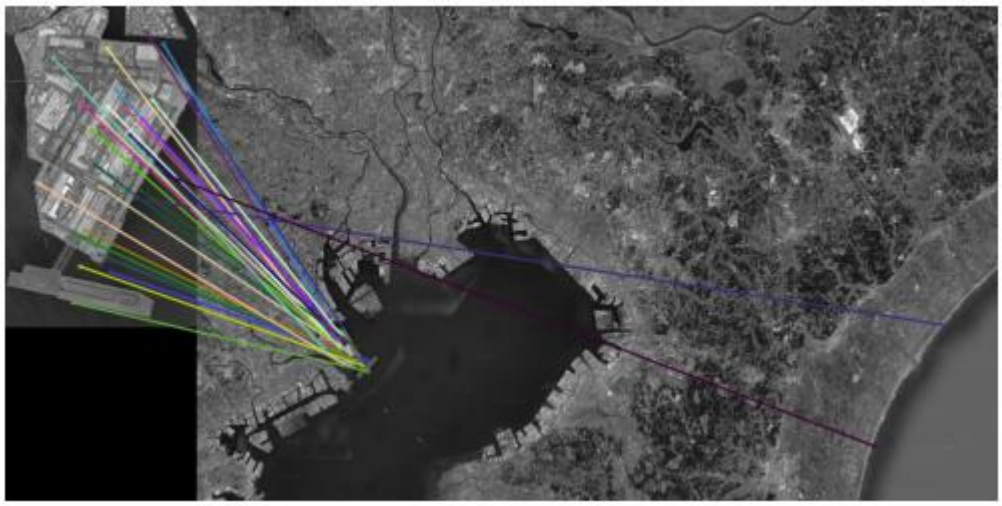

Fig. 13 Matched features between Haneda Airport and the satellite image

We can perform the same steps for Narita Airport, and feature extraction and feature matching results are shown below:

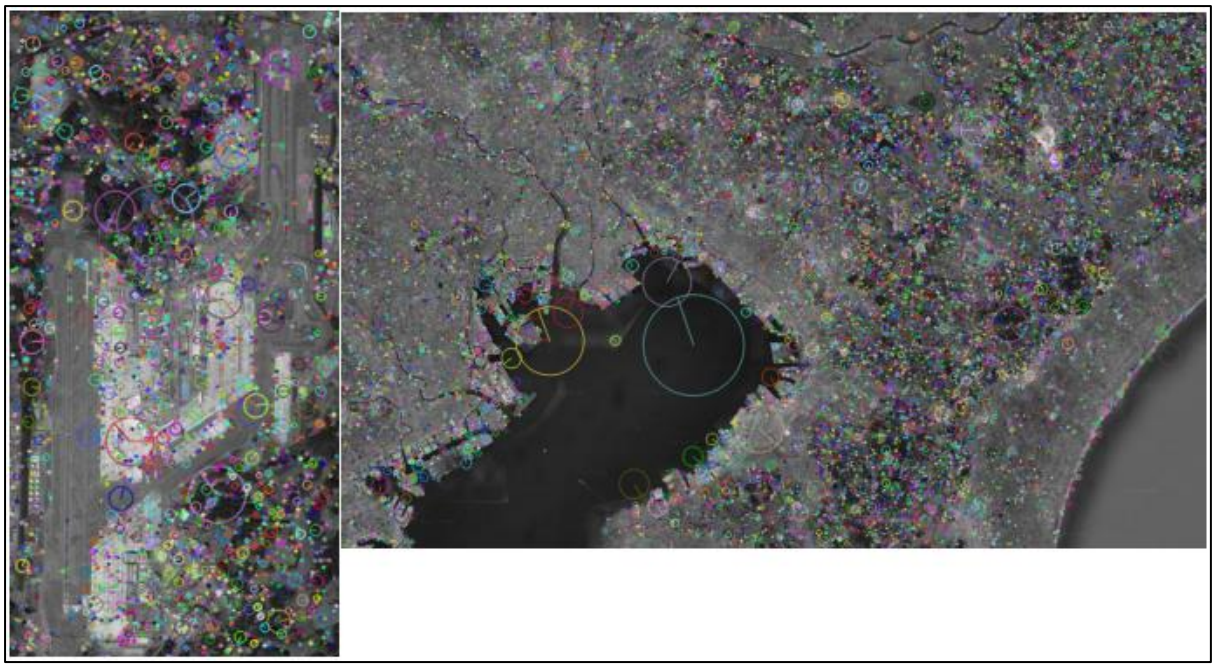

Fig. 14 Extracted SIFT features of Narita Airport and the satellite image

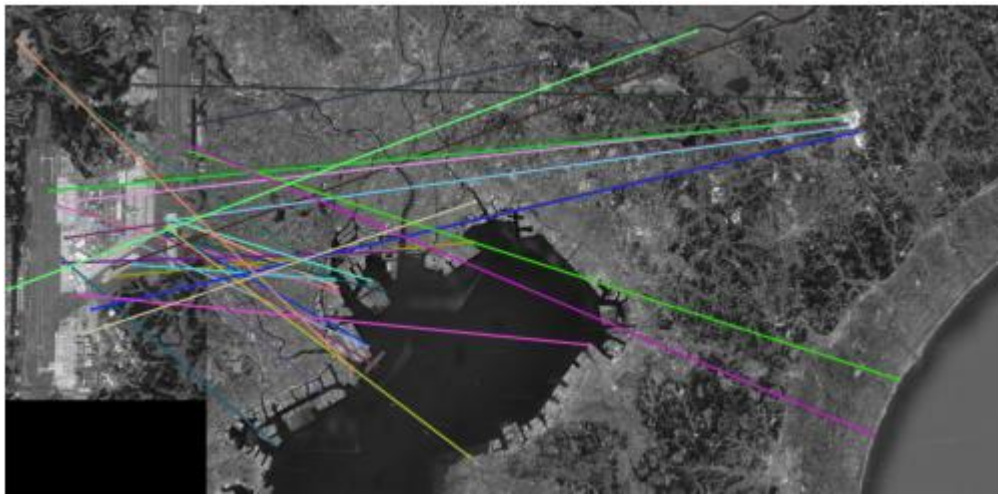

Fig. 15 Matched features between Haneda Airport and the satellite image

In the case of Haneda Airport, the feature matching is almost perfect, and we can easily locate it in the satellite image. This is possible because Haneda Airport has many distinct characteristics. It is an island with a specific shape, and the contrast with sea is high helping SIFT matcher to have great results.

For Narita Airport, there are many incorrect matches, making it difficult to locate Narita Airport in the satellite image. The reason is that Narita Airport is surrounded by fields and ground which doesn't offer a good contrast. Then, when Narita Airport and its surroundings get smaller due to zooming out, this contrast gets worse, and SIFT matcher performance is lost.

6. Conclusions

From the conducted experiments, we conclude that:

- Harris detector with NCC method is not scale invariant and rotation invariant. It makes this method not suitable for practical applications where changes in scale and orientation exist.
- SIFT matcher is scale invariant, rotation invariant, and invariant to small changes in viewpoint. It makes this method more suitable for practical applications.
- SIFT matcher can be used to locate specific places in satellite images, with great performance when the place to be located has many distinct characteristics.

7. References

[1] Jan Erik Solem, Programming Computer Vision with Python, O'Reilly, 2012.

[2] https://www.analyticsvidhya.com/blog/2019/10/detailed-guide-powerful-sift-technique-image-matching-python/

[3] https://docs.opencv.org/4.x/da/df5/tutorial_py_sift_intro.html

[4] https://docs.opencv.org/3.4/d7/d60/classcv_1_1SIFT.html

[5] https://docs.opencv.org/4.x/dc/dc3/tutorial_py_matcher.html

[6] https://docs.opencv.org/4.x/d3/da1/classcv_1_1BFMatcher.html

[7] https://docs.opencv.org/4.x/d4/d5d/group__features2d__draw.html