Assignment 2: Warping, Matching, Stitching, Blending

Darshan Shinde(dshinde),
Virendra Wali (vwali),
Shashank Khedikar (skhedika),
Santosh Kangane (sckangan)

Part 1: Image matching and clustering

Abstract:

There are two major parts of Image matching and Clustering

1. Image Matching: Here we extract the image features i.e. Key points and ORB descriptors for each image. Then for each source image we compare each feature point with every other feature point in target image and calculate the distance matrix.
2. Image clustering: The distance matrix calculated in first step is used for image clustering in 2nd step. We tried using K-means for image clustering with distance matrix as Euclidean and Hamming distance.

Design decisions:

Two main design decisions we took in this part:

1. **Using hamming distance:**
   We have many distance options available like Euclidean, Hamming, Manhattan, Cosine distance, etc. But out of these, hamming distance gives us better accuracy in compare to all others.
2. **Threshold value for ratio:**
   To validate the uniqueness of feature points, we have taken a threshold value for ratio. If the ratio is less than threshold then the feature point is valid otherwise, we discard feature point. If we keep the threshold high, then we will get some incorrect match pairs along with correct matches. Because higher ratio means distance to closest feature point and second closest feature point are almost same which means the feature point of source is very generic e.g., a point in the blue sky and it will be mapped to any blue point like any point is sky or sea or any blue color image which is not correct. Hence, to avoid such cases, we should keep our threshold as low as possible. At the same time, threshold shouldn't be so less to remove all points and didn't get any match pair. At threshold = 0.7 we found fine balance, hence we have used default threshold value as 0.7.

Implantation:

1. *getMatches(image1, image 2, threshold):*
   inputs:
   i. *image1:* path of image 1
   ii. *image2:* path of image 2

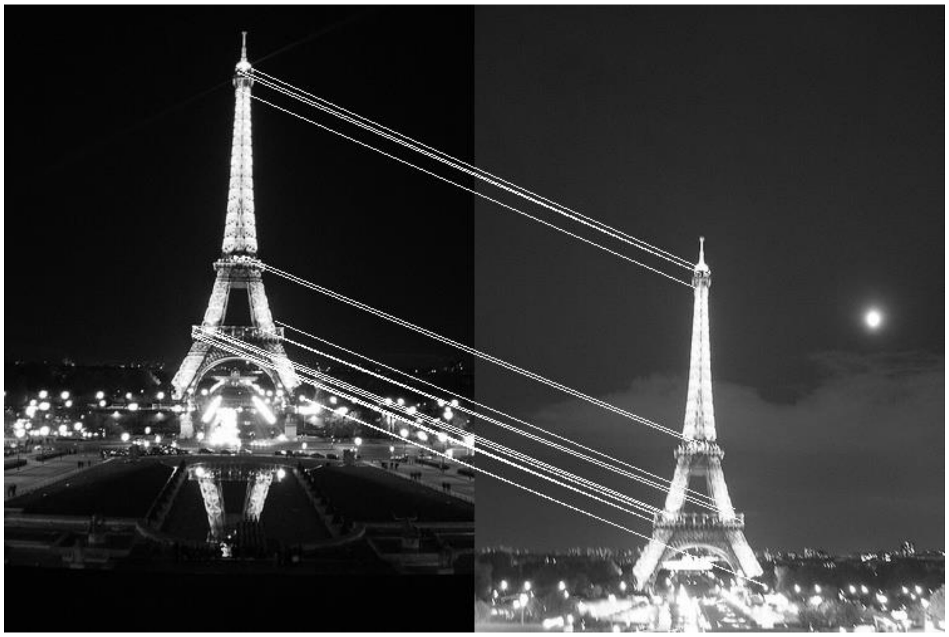
iii. *threshold:* threshold value to validate the uniqueness and importance of the feature point of the image1

implementation:

- We are using ORB from cv2 library to get feature points of the images. We have extracted keypoints, and descriptors of all feature points of both images.
- Then comparing every feature point of image 1 with every feature point of image 2.
- For each feature point of image 1, calculated the distance of all feature points of image 2 and picking only first 2 feature points i.e., closest and second closest to the feature point of image 1. We used normalized hamming distance between two feature points. (using norm(..) of cv2 library).
- If ration of distance to closest point and second closest point is less than threshold then only, we are considering the feature point of image 1 is valid and generating a match pair as feature point of image 1 and closest feature of image 2 from this. Otherwise, we are discarding this feature point of image 1.

output:

- List of all pair of keypoints of match pairs of between image 1 and image 2 in sorted order of distance in between them.



2. Calculate Distance Matrix:
    - We tried both Euclidean distance and Hamming distance matric to compare two images.
    - For each feature point in source image we calculated the distance with each feature point in target image
    - Then selected the first two features point which are closest and second closest from source image.

- If the ratio of closest and second closest feature point is below the threshold value, we select that as candidate distance between those two images.
- Repeated above steps to compare each input image with every other image and constructed a NxN distance matrix (where N is number of input image count)

3. Clustering the images:
   - We used k-means to cluster the images based on distance matrix calculated from above step.
   - We selected k random images as centroids.
   - For n iterations we are repeating following steps:
     - Using distances of all images from these k centroids, we tag images to k different clusters.
     - Update centroid of each cluster to the image which have sum of distance to all other images from same cluster is minimum.

Problem faced during clustering:

- We tried using Euclidean as well as hamming distance matric to calculate the distance between two images. However, the resultant distance matrix does not have clear class separation boundary.
- Hence, the K-mean clustering does not give consistent and clear grouping of images.
- We notice that the at times similar images shows larger distance compare to completely different images.

Part 2: Image transformations

There are two major parts of image transformation:

1. Generating transformation matrix using given set of points,
2. Creating transformed image using transformation matrix.

Transformation matrix is generated using given transformation type and given set of points from original image and transformed image.

Based on transformation type, we have generated homographic matrix by solving linear system of equations.

Transformation types:

1. **Translation:**
   To find translation coefficients we need only one point. We are using below transformation matrix for image translation:
   Translation in x direction: Difference in x co-ordinates of point in original image and related point in transformed image
   Translation in y direction: Difference in x co-ordinates of point in original image and related point in transformed image

2. **Euclidean transformation:**
   Degree of freedom for Euclidean transformation is 3. Hence, we need 2 points to find Euclidean transformation coefficients.
   There are 3 linear transformation type of Euclidean transformations:
   i. Scale

$$T = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

   ii. Rotate

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

   iii. Shear

$$T = \begin{bmatrix} 1 & sh_x & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

   Matrix A for Euclidean transformation:

$$T = \begin{bmatrix} x_1 & y_1 & 0 & 0 \\ 0 & 0 & x_1 & y_1 \\ x_2 & y_2 & 0 & 0 \\ 0 & 0 & x_2 & y_2 \\ x_3 & y_3 & 0 & 0 \\ 0 & 0 & x_3 & y_3 \\ x_4 & y_4 & 0 & 0 \\ 0 & 0 & x_4 & y_4 \end{bmatrix}$$

3. **Affine transformation:**
   Degree of freedom for affine transformation is 6. Hence, we need 3 points to find affine transformation coefficients. We used below transformation matrix for image for affine transformation:

   Homographic Matrix for Affine:

$$T = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix}$$

   Matrix A for Affine transformation

$$T = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & 1 \\ x_3 & y_3 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ x_4 & y_4 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_4 & y_4 & 1 \end{bmatrix}$$

4. **Projective transformation:**
   Degree of freedom for projective transformation is 8. Hence, we need 4 points to find projective transformation coefficients. We used below transformation matrix for image for affine transformation and matrix A:

   Homographic matrix for projective transformation:

$$T = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

   Matrix A for Projective transformation:

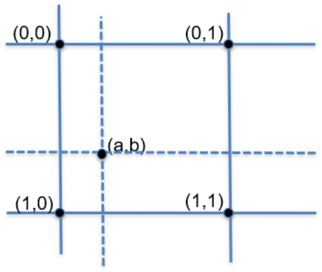
$$T = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1 * \bar{x_1} & y_1 * \bar{y_1} \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1 * \bar{y_1} & y_1 * \bar{y_1} \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2 * \bar{x_2} & y_2 * \bar{y_2} \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2 * \bar{y_2} & y_2 * \bar{y_2} \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3 * \bar{x_3} & y_3 * \bar{y_3} \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_3 * \bar{y_3} & y_3 * \bar{y_3} \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4 * \bar{x_4} & y_4 * \bar{y_4} \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4 * \bar{y_4} & y_4 * \bar{y_4} \end{bmatrix}$$

Image warping:

The second task was to apply the transformation matrix on input image. We used Inverse wrapping technique i.e. for each target pixel g(x’, y’) find the corresponding (x, y) = T-1(x’, y’), to avoid issues of mapped pixel falling in between of two pixel.

The mapped pixels which lands in between of pixel, were approximated based bi-liner interpolation. For each such pixel the values is approximated by giving more weightage to the nearest pixel among all 4 neighbors. bi-liner interpolation can be expressed as follows (Reference Module 6):

$$(1 - b)(1 - a) F_{0,0} + (1 - b) a F_{1,0} + b (1 - a) F_{0,1} + b a F_{1,1}$$



Results:

Image 1:

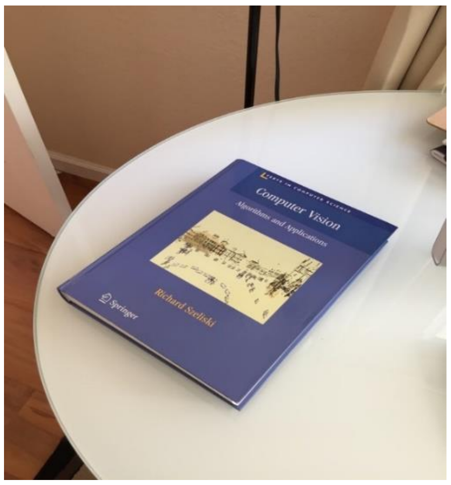
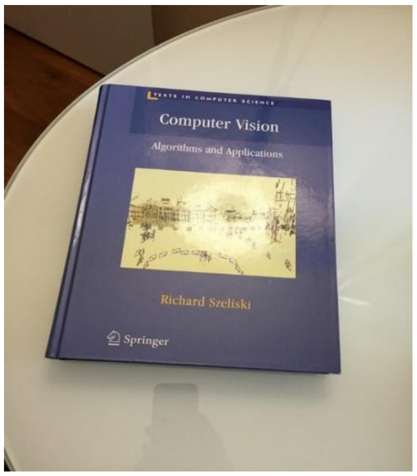
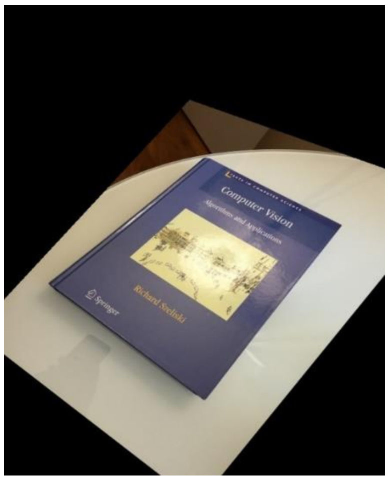


Image 2:



Warped Image:



Part 3: Automatic image matching and transformations

Abstract:

In this part 3 of the assignment 2, our objective is to stitch two images and generate a single big panoramic image. This problem statement is very high-level formulation of 3 sub-problems:

1. Align both images in same coordinate axis
2. Translate one of the images in such a way that we can join both images directly
3. Join both images

So, our approach is just aligned to above three points.

1. We are warping one of the images to align it to another image. I have implemented RANSAC to generate warped image.
2. Then we are translating second image using translation coefficients calculated using comparing warped image 1 and original image 2.
3. Then we are stitching warped image 1 and translated image 2 in such a way that for every pixel in final image if there is only 1 image has ON pixel at that co-ordinate then pixel value is directly copied to the final image and if both images have ON pixel at that position then average of both pixel values are copied to final image.

This can be extended for multiple image stitching, but as assignment requirement we are implementing this only for stitching 2 images.

Implementation:

Approach I followed to solve this problem is discussed in the abstract part. Now let's discuss the implementation part of the discussed approach. I am listing some functions written in code, with input descriptions and implementation part, problem we faced in this part and output.

1. *getMatches(image1, image 2, threshold):*
   inputs:
   iv. *image1:* path of image 1
   v. *image2:* path of image 2
   vi. *threshold:* threshold value to validate the uniqueness and importance of the feature point of the image1

   implementation:
   - We are using ORB from cv2 library to get feature points the images. We have extracted keypoints, and descriptors of all feature points of both images.
   - We are comparing every feature point of image 1 with every feature point of image 2.
   - For each feature point of image 1 we have calculated the distance of all feature points of image 2. And picking only 2 feature points i.e., closest and second closest to the feature point of image 1. We are calculating normalized hamming distance between two feature points using norm(..) of cv2 library.
   - If ration of distance to closest point and second closest point is less than threshold then only, we are considering the feature point of image 1 is valid and generating

a match pair as feature point of image 1 and closest feature of image 2 from this. Otherwise, we are discarding this feature point of image 1.

output:

- List of all pair of keypoints of match pairs of between image 1 and image 2 in sorted order of distance in between them.

2. *getTransformationMatrix(fp, tp):*

   inputs:

   i. *fp:* matrix of coordinates of feature points from image 1
   ii. *tp:* matrix of coordinates of matching feature points from image 2

   implementation:

   - By solving linear system of equations for 4 points of image 1 and corresponding 4 points of image 2 we have calculated transformation matrix.

   output:

   - 3*3 transformation matrix

3. *generateWarpedImage(image, k):*

   inputs:

   i. *image:* the original image we want to warp
   ii. *k:* transformation matrix

   implementation:

   - We generate image frame for warped image of same shape of original image.
   - Going in forward direction may give us some fraction values of corresponding pixel in warped image and this can mess the final output. Hence instead going in forward dimension, we prefer to go in backward direction here and perform bi-linear interpolation.
   - For every pixel in warped image, we are recollecting pixel coordinates of original image by taking dot product with inverse of transformation matrix.
   - If we get fractional pixel coordinates, then we are taking weighted sum of pixel values of nearest 4 pixels. This is nothing but the bi-linear interpolation.

   Problem we faced:

   - This approach fills pixel values of warped image by mapping it with point is original image. During warping, some corners of the original image can go outside the warped image frame. Or pixels in warped image frame does not cover all pixels of original image and hence we were missing some part of the image.

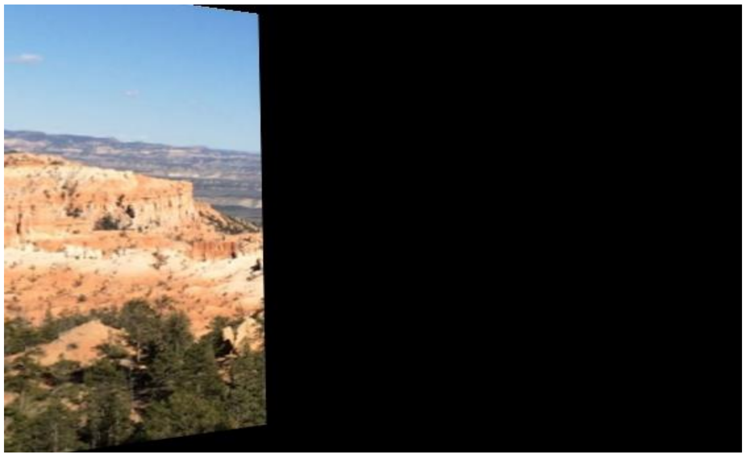
   Solution to the problem:

   - We calculated the corresponding pixel coordinates in warped image for all four vertices of original image by dot product with transformation matrix.
   - Then we are picking maximum x, y coordinate values as max_x, max_y and minimum x, y coordinate values as min_x, min_y.
   - If min_x, min_y > 0 then we are setting it to 0.

- We generated image frame for warped image of shape (max_x - min_x) * (max_y - min_y).
- Then we are shifting our origin (x = 0 and y = 0) of warped image frame to (x = -min_x and y = -min_y).
- Then we are filling all pixel values using new co-ordinates of warped image frame with pixel values corresponding pixels of original image (for fractional co-ordinates using bi-linear interpolation as already mentioned).

output:

- numpy array of warped image frame



Unable to get full image after warping

4. *ransac(matchPairs, itr = 1000, errorT = 25):*

   inputs:

   i. *matchPairs:* list of pair of keypoints of match pairs between 2 images
   ii. *itr:* number of iterations; default value = 1000
   iii. *errorT:* threshold for distance in actual point and estimated point to find inlier; default value = 25

   implementation:

   - We implemented following steps for itr number of iterations and search for the model with highest number of inliers.
   - We picked random 4 pairs from match points.
   - Separated all those 4 pairs into 2 matrices, points of image 1 (1st keypoint of every pair) as fp and points of image 2 (2nd keypoint of every pair) as tp.
   - We calculated transformation matrix using these fp and tp using already discussed getTransformationMatrix(..)
   - Then we generate 2 matrices for point in image 1 and image 2 by separating all pairs from matchPairs as A and B.
   - Using transformation matrix generated in above step, we calculate the estimated coordinates of all points in A by taking dot product of matrix A and transformation matrix.

- Then for each point, we are finding distance between estimated coordinates for points and actual coordinates from matrix B. If distance is less than errorT, then we are considering this point as inlier for the current model.
- We are keeping track of best_model as model with highest number of inliers and max_inliers as inliers number of inliers in it. If current model has a greater number of inliers than max_inliers then we are updating best_model with current model and max_inliers with number of inliers for current model.

output:
best_model i.e. model (transformation matrix) with maximum number of inliers

5. *getTranslationForTargetImage(source, target):*
   inputs:
   i. *source:* path of source image file
   ii. *target:* path of target image file

   implementation:
   - We get matchPairs from source image and target image using already discussed getMatches().
   - To calculate coefficient of translation one pair of point (point from source image and target image) is enough, hence we select 1st pair from matchPairs. We can select any pair but 1st pair has shorted distance in between means best match amongst all other will be best for our purpose, hence I select 1st pair.
   - Then I took difference between x, y co-ordinate values of point of source image and target image as transX and transY

   output:
   - transX, transY (translation coefficients in x and y direction)

6. *generateTranslatedTargetImage(image, transX, transY):*
   inputs:
   i. *image:* image which we want to translate (image which we want to relocate to new position)
   ii. *transX:* translation coefficient in direction x
   iii. *transY:* translation coefficient in direction y

   implementation:
   - We create a new image frame of size (width of image + transX * height of image + transY).
   - Copy pixal values of image to new image frame by shifting origin (x = 0, y = 0) to x = transX, y = transY

   output:
   - numpy array of new image frame

7. *stitch(image1, image2, output):*
   inputs:
   i. image1: path of image 1

ii. image2: path of image 2
iii. output: name of output file

implementation:

- We get matchPairs in between image1 and image2. From matchPairs we pick 1st pair.
- We generate a new image frame of shape as follows:
  length = x coordinate of match point of image1 + (length of image2 - x coordinate of match point of image2) and
  height = y coordinate of match point of image1 + (height of image2 - y coordinate of match point of image2)
- Fill pixel value for all pixels in image frame as follows:
  - If pixel co-ordinate not present in one of the images, then copy pixel value of another image.
  - If pixel value of one of the images is 0, then copy pixel value of another image.
  - Otherwise, put average of pixel values of both images.
- Save the image frame as image file with name as value of output variable.

Known problems:

1. RANSAC is not consistent, so sometimes both image1 and image2 does not align to each other and hence some part of the final image becomes ghost.
2. If image1 and image2 dont have enough common features, then merging creates ghost image.

Future work:

This implementation only stitches two images at a time. We can scale it to stich n number of images in a single execution in future.

Results:

Image: 1

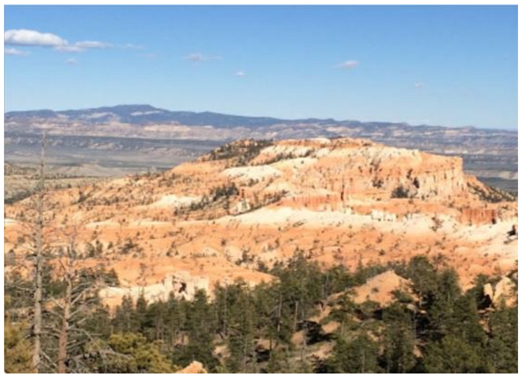
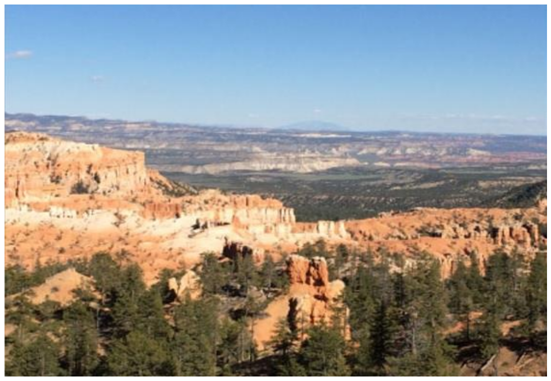
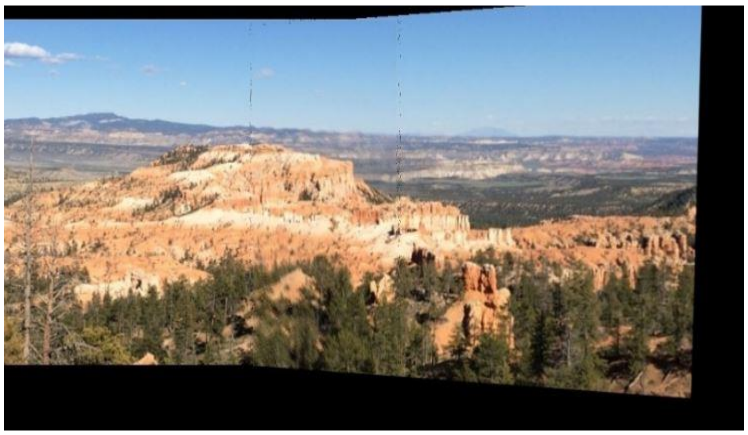


Image 2:



Parnaromic Image (Cropped):



Failure Case:

Image 1:

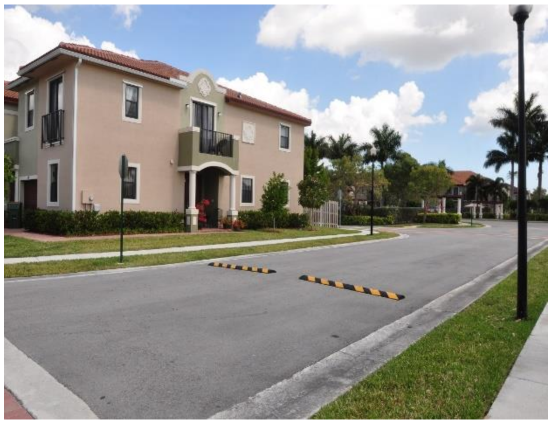
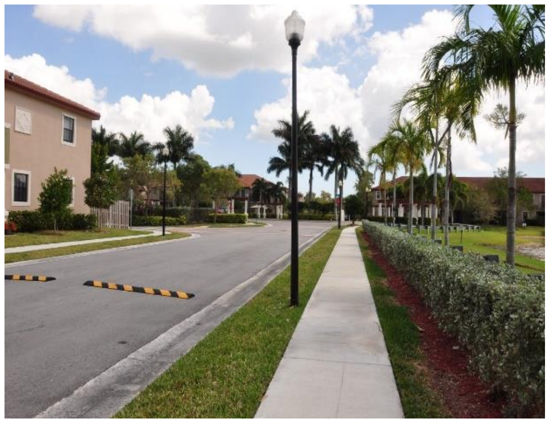


Image 2:



Panoramic Image (Cropped):