Fitting & Alignment

S.Sanjith : 190562G

Abstract

Despite the advancement in fitting approaches, RANSAC is a good algorithm that can deal with higher outlier ratios up to 50%. In this report, we will talk about the implementation of RANSAC in two different situations. In the first part of the report, we will fit a circle with given points. Then we try to fit two shapes using homographic transformation. In the third part, we will analyze an application of finding homographic transformation between two images. Finally, we will try to generalize the RANSAC approach to find the best circle and suitable homographic transformation used in stitching two images.

1 Fitting Circle

Any circle can be defined by a minimum of three points. That is a circle is fully defined if it is expected to go through three points. Hence it is not suitable to fit a circle accurately with a given number of points. By defining the total euclidean distances as the measure of error, a better choice for the circular fit can be achieved.

Randy Bullock

Two famous approaches for the fitting circle are circular regression and randy bullock fit. With few implementations, it is observed that the randy bullock fit is doing better in the noisy environment.
Python Implementation : Randy Bullock Fit

```python
def randy_bullock_fit(points):
    N = len(points)
    X, Y = points[:, 0], points[:, 1]
    x_bar, y_bar = np.mean(X), np.mean(Y)
    U, V = X - x_bar, Y - y_bar
    s_uu, s_uv, s_vv = np.sum(U * U), np.sum(U * V), np.sum(V * V)
    s_uuu, s_uvv, s_vuu, s_vvv = np.sum(
        U * U * U), np.sum(U * V * V), np.sum(V * U * U), np.sum(V * V * V)
    A = np.array([[s_uu, s_uv], [s_uv, s_vv]])
    B = np.array([[1 / 2 * (s_uuu + s_uvv)], [1 / 2 * (s_vvv + s_vuu)]])
    [[u_c], [v_c]] = np.linalg.inv(A) @ B
    alpha = math.pow(u_c, 2) + math.pow(v_c, 2) + (s_uu + s_vv) / N
    r = math.sqrt(alpha)
    c_x, c_y = u_c + x_bar, v_c + y_bar
    circ = Circle(c_x, c_y, r)
    return circ
```

2 Homography Calculation

Following algorithm implements the ***Normalized Direct Linear Transformation (DLT)*** method (described in the *Multiple View Geometry in Computer Vision*(Second Edition), by *Richard Hartley* and *Andrew Zisserman*), to find the homography matrix $\mathtt{M}_{3\times 3}$ using 5 or more pairs of corresponding points. Let $x_i = [x, y, 1]$ and $x'_i = [x\prime, y\prime, w\prime]$ be two corresponding points in the given two images. Then the transformation is given by $x'_i = Hx_i$. This equation can be represented by using the vector cross product as $x'_i \times Hx_i = 0$ sine both the components have the same direction even though they differ in magnitude. This equation can be further simplified to obtain the following linearly independent system of two equations corresponding to each pair of points. Where h^j indicates the j^{th} row of the Homography matrix and $j = 1, 2, 3$.

$$\begin{bmatrix} 0^\top & -w\prime x_i^\top & y\prime x_i^\top \\ w\prime x_i^\top & 0^\top & -x\prime x_i^\top \end{bmatrix} \begin{bmatrix} h^1 \\ h^2 \\ h^3 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & -w\prime.x & -w\prime.y & -w\prime.1 & y\prime.x & y\prime.y & y\prime.1 \\ w\prime.x & w\prime.y & w\prime.1 & 0 & 0 & 0 & -x\prime.x & -x\prime.y & -x\prime.1 \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} = O$$

We can obtain 5 pairs of such equations and they can be put into a single matrix to solve for the unknown h^js through ***Singular Value Decomposition*** as described in the `Algorithm 4.1, 4.2` in the above mentioned reference book. Since this is a very basic algorithm, it is prone to error. We will discuss about the performance of this algorithm in the final part of this report.
Python Implementation : Find Homography

```
def calcHomography(m_points):
    """
    The normalized DLT for 2D homographs.
    """
    p1, p2 = m_points[:, 0], m_points[:, 1]
    # Normalizing the points using predefined function
    T1, p1 = Homography.normalizePoints(p1)
    T2, p2 = Homography.normalizePoints(p2)
    # Initialising an array to keep the coefficient matrix
    A = np.zeros((2 * len(p1), 9))
    row = 0
    # Filling rows of the matrix according to the expressions
    for point1, point2 in zip(p1, p2):
        A[row, 3:6] = -point2[2] * point1
        A[row, 6:9] = point2[1] * point1
        A[row + 1, 0:3] = point2[2] * point1
        A[row + 1, 6:9] = -point2[0] * point1
        row += 2
    # Singular Value decomposition of A
    U, D, VT = np.linalg.svd(A)
    h = VT[-1]
    # Reshaping to get 3x3 homography
    H = h.reshape((3, 3))
    # Renormalization
    H = np.linalg.inv(T2).dot(H).dot(T1)
    return Homography(H)
```

3 Super-Imposing Images

Super-imposed images look strange if they have different orientations and translations. In fitting an image in a specific location of background, it is crucial to ensure the images are matching properly. This can be performed by finding homographic transformation between two images and transforming one image into the other domain before image stitching.

The following implementation matches four corners of two billboards with two posters shown next to them(1). Although the matching seems to have better performance(2) it required some fine-tuning along the way. It is because four matches alone are not sufficient enough to find an exact homographic transformation.

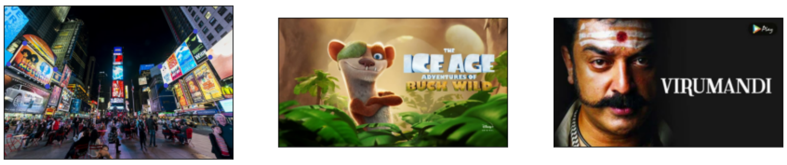


Figure 1: Billboards and Posters

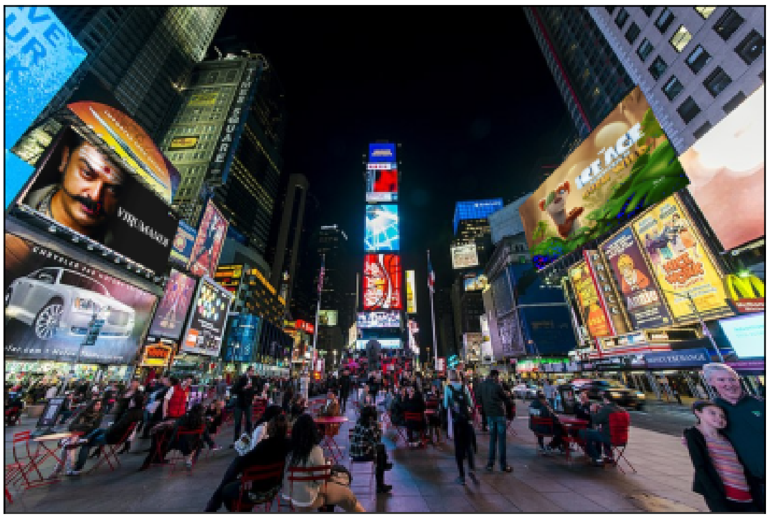


Figure 2: Overlayed Image

4 The RANSAC

Fitting a model in a set of noisy features leads to unacceptable models. In such cases, it is critical to differentiate between inliers and outliers. RANSAC is a strategy that could be adapted to find the best model fit even in noisy scenarios. The key idea of the approach is to let each point vote for all the models that are compatible with it. The model with the most votes is chosen as the best fit.

Algorithm

1. Draw minimum number of points required to fit a model uniformly at random and fit model to the points.
2. Find the count of inliers to the model (points whose distance or residual w.r.t model is less than threshold)
3. If there are enough inliers, accept the model.
4. Perform this until the maximum iteration count required is reached.
5. Find the best model from the accepted models and refit using all the model points.

Python Implementation

```python
def fit_model(data, model, threshold=0,min_iteration=0):
    num_iterations,iterations_done = math.inf,0
    max_inlier_count,best_model = 0,None
    desired_prob,data_size = 0.95,len(data)

    # Minimum number of sample requirement (circle -> 3 , Homography -> 5)
    num_sample= 3 if model==’circle’  else 5

    while num_iterations > iterations_done:
        np.random.shuffle(data)
        sample_points = data[:num_sample]
        estimated_model = generate_model_from_points(sample_points)

        if inlier_count > max_inlier_count:
            max_inlier_count = inlier_count
            best_model = estimated_model
            prob_outlier = 1 - inlier_count / data_size
            num_iterations =  log(1 - desired_prob) /log(1 - (1 - prob_outlier) ** num_sample)

            # Adjusting maximum iteration requirement according to maximum inlier count
            if min_iteration>0 and num_iterations < min_iteration:
                num_iterations = min_iteration
        iterations_done = iterations_done + 1
    final_model = generate_model_from_points(inliers)
    return final_model
```

Scenario-I : Fitting Circle

The following figures(3,4) illustrate two circular models approximated for the points shown together. With proper tuning of the threshold parameters of first model(3,R=9.856 & O≡[0.123,-0.156]) stays closer to the parameters of the circle used to create these noisy points(R=10 & O≡[0,0]). The second model(4) seems to depict the nature of noisy line used to create half the data points.

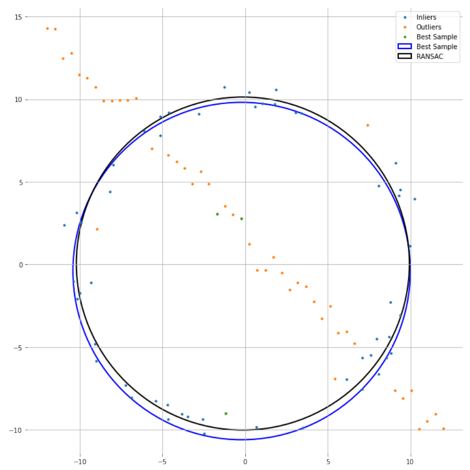


Figure 3: Model I

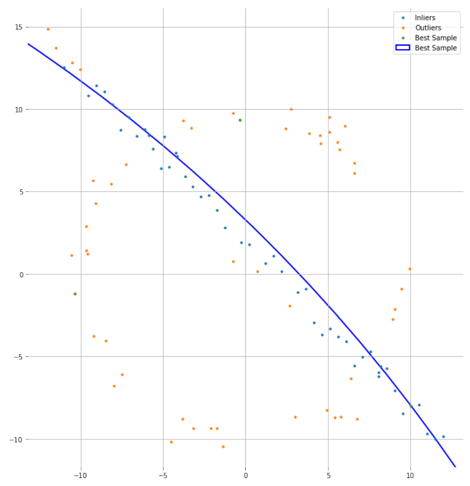
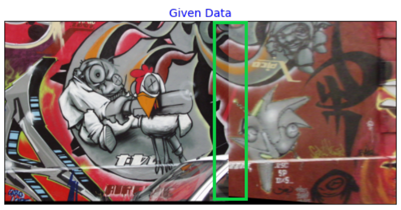


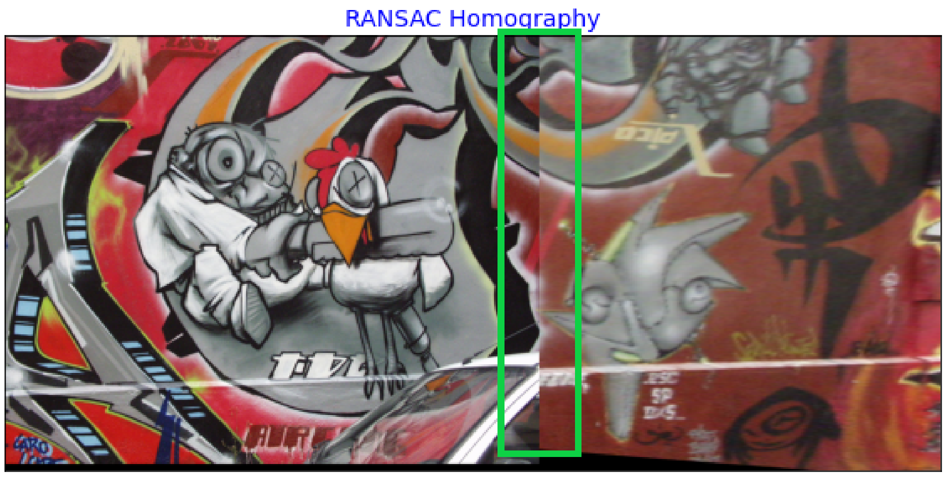
Figure 4: Model II

Scenario-II : Fitting Homography

The following figures show the transformation of the Graffiti *im1.ppm* onto *im5.ppm* using the provided homography. Consider the area enclosed using the green-colored bounding box. The transition between two images is almost indistinct and it will be completely seamless if the intenseness of the pixels around the transitions are accrately matched.



The following illustration shows the same stitching using the homography extracted from a RANSAC implementation. Firstly the matching features between two images are extracted using the SIFT feature. Then these features are matched using a RANSAC execution with proper thresholding to achieve the transformation homography.



Direct matching between two images doesn't seem to work due to the scarcity of good matching feature count. Hence it is decided to match adjacent figures(img1 & img2, img2 & img3....) which are having more than 1000 matches. And deriving the overall transformation by multiplying adjacent ones. Although a slight translation is visible inside the green bounding box the result can be believed as a tighter extraction for a basic implementation.

$$H_{1\to5} = H_{4\to5} \cdot H_{3\to4} \cdot H_{2\to3} \cdot H_{1\to2}$$

$$H_{given} = \begin{bmatrix} 0.625 & 0.058 & 222.012 \\ 0.222 & 1.165 & -25.606 \\ 0.001 & 0.000 & 1.000 \end{bmatrix} \qquad H_{ransac} = \begin{bmatrix} 0.594 & 0.023 & 227.644 \\ 0.215 & 1.066 & -8.788 \\ 0.001 & 0.000 & 1.000 \end{bmatrix}$$

References

1. Randy Bullock Fit
2. opencv
3. An article about implementation of RANSAC for circle fitting.
4. Panoramic stitching using RANSAC algorithm
5. Youtube playlist on Image Stitching