Homography Transformations

Sai Vijay Rohit Pantam
Department of Computer Science and Engineering
University of South Florida
Tampa, United States
U56671661

I. Description of Algorithm

For this assignment, we divide the algorithm into two parts. First part is to provide the corresponding points of source image and target image. This part can be automated by using the previous assignment algorithm. Once we have the points we find the homography matrix using the delta of points, hessain matrix and jacobians found. Using the homography matrix, corresponding target points in source image are found and replaced.

II. Description of Code

A. *Finding Homography Matrix*

Initialize the homography matrix as identity matrix. Parameters are initialized as zeroes. We find the initial residue using the initialized homography matrix and parameters. Now, residue is reduced by updating the parameters using Jacobian and Hesian matrix. Using the target image co-ordinates we form the jacobian matrix. Hesian matrix is formed using the jacobian and jacobian transpose. Mathematically speaking Hesian matrix is summation of matrix multiplication of jacobian Transpose of a co-ordinate and jacobian of the same co-ordinate. An array(b) is found by the summation of matrix multiplication of jacobian transpose of a co-ordinate and delta vector of the same co-ordinate. Finally, Homography matrix is found by finding parameter matrix. Parameter matrix is found by multiplying inverse of hesian matrix and array(b). Iterate the above till the residue goes below a threshold value. Once it goes threshold, transform the pixel values.

$$\boldsymbol{J} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{p}} = \frac{1}{D}\begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -x'x & -x'y \\ 0 & 0 & 0 & x & y & 1 & -y'x & -y'y \end{bmatrix}$$

B. *Using Homography Matrix and Transforming the source*

Iterate through each pixel of the target image and find the points[1] which should be changed with the source. By multiplying homography with each point from the above collection, corresponding source location can be found. Once the location in source is found, the target pixel value is changed with that of source.

$$\tilde{\mathbf{x}}' = \tilde{\mathbf{H}}\tilde{\mathbf{x}}$$

Fig. 1. X‘ is target location, H is homography and X is source location.

$$\begin{bmatrix} a \\ b \\ D \end{bmatrix} = \begin{bmatrix} 1+h_{00} & h_{01} & h_{02} \\ h_{10} & 1+h_{11} & h_{12} \\ h_{20} & h_{21} & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

$$\mathbf{x}' = \begin{bmatrix} x_i' \\ y_i' \end{bmatrix} = \begin{bmatrix} a/D \\ b/D \end{bmatrix}$$

C. *Automation of finding the corners in the target image*

Automation of finding the corners in the target image is the second part of the algorithm. This is done using the knowledge gained in the previous assignment. In the previous assignment the first part i.e, finding the good points to track, will provide us the strongest color changing points in the image. For this assignment these points are nothing but corners of the polygon. The automation part works efficiently with 4 cornered polygons as of now.

III. Results

Fig 2 and Fig 3 are input source images. Fig 4 and Fig 5 are input target images.
Fig 6 and Fig 7 are output images after homography transformation. Fig 6 provides a classic example of this algorithm not the best one to use for two non identical source and target polygons. Fig 7 provides an example for the best use of the above algorithm.

IV. Conclusion

With this project homographic transformation is accomplished. Finding the right homography matrix is a challenging task. For finding the right homography matrix we use residual technique. That is to find residual for every iteration. Apart from this, we might face few Hesian matrices which are singular after certain iterations. This problem was solved by adding a downing parameter to make it a non singular matrix. This algorithm is confined to identical source and target polygons. If non identical polygons are used for transformation, we might lose a bit of source image pixels. Time complexity of this algorithm depends on the points selected and their order.

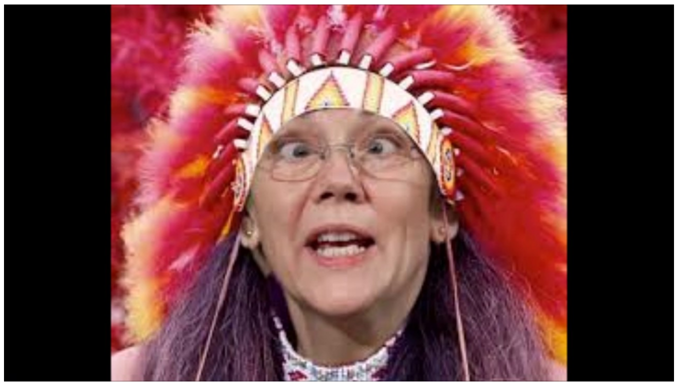

Fig. 2. Input Source File : Poca.jpg

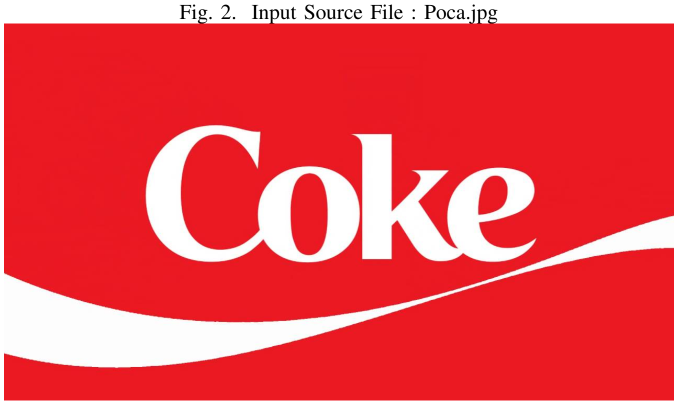

Fig. 3. Input Source File : cokeBig.png

References

[1] R. P.-C. R. Perez-Carrasco, "What's the fastest way of checking if a point is inside a polygon in python," Stack Overflow, 01-Mar-1966. [Online]. Available: https://stackoverflow.com/questions/36399381/whats-the-fastest-way-of-checking-if-a-point-is-inside-a-polygon-in-python. [Accessed: 26-Feb-2020].

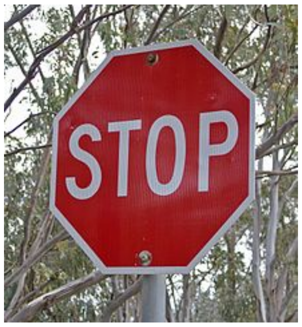

Fig. 4. Input Target File : stopSign.jpg

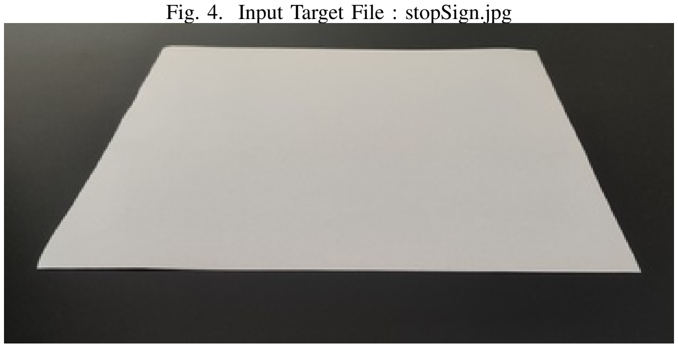

Fig. 5. Input Target File : WB.jpg

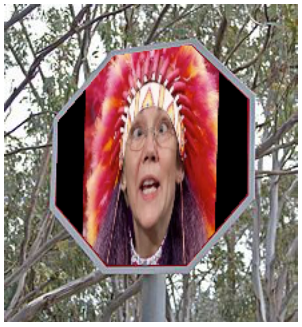

Fig. 6. Output : poca.jpg, stopSign.jpg

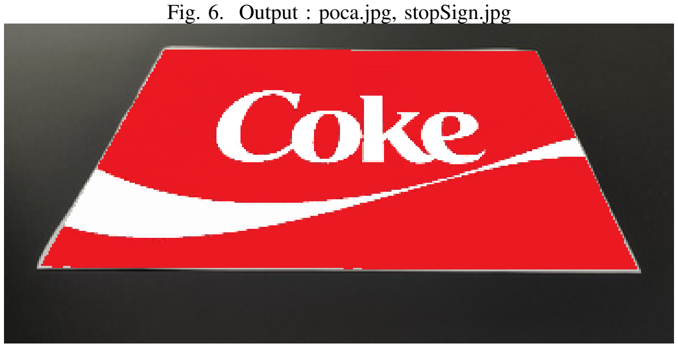

Fig. 7. Output : poca.jpg, WB.jpg