2D Projective Transformations.

Manuel Felipe Pineda Loaiza

Abstract—**In this work I applied the theory about 2D projective transformations studied in the computer vision class. The work consists of three experiments, the first experiment aims to remove the perspective of an image, the second one simulates a scanner, and the last one applies the perspective to one image in order to embed it into another image.**

Keywords—Computer vision, 2d projective transformations, 2d geometry, homographies, perspective.

I. Introduction

A projective transformation is a natural deformation of the geometrical shapes that we see every day, for example when we take photographies of doors, windows and similar objects, they do not look like their original shapes (rectangles) but our brain automatically removes the perspective and we can know how the objects really look. This is a very interesting problem that we can solve using linear algebra and computational geometry, and it is in fact part of the basics for more complex tasks in the computer vision.

In order to define a projective transformation, we need to define first the homogeneous coordinates: They are a system of coordinates used in projective geometry, as Cartesian coordinates are used in Euclidean geometry. They have the advantage that the coordinates of points, including points at infinity, can be represented using finite coordinates. [1]

Formally a projectivity is is an invertible mapping h from the projective space to itself (the space in homogeneous coordinates) such that three points x_1 , x_2 and x_3 lie on the same line if and only if $h(x_1)$, $h(x_2)$ and $h(x_3)$ do [2]. This is also known as ***homography***.

A. Homography estimation

For the goal of this work, we only focus in the *planar* projective transformations, they are linear transformations on homogeneous 3-vectors represented by a non-singular matrix of size 3 x 3 and we can use them to add or remove perspective on the images as it is shown in the further sections:

$$\begin{pmatrix} x'_1 \\ x'_2 \\ x'_3 \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

In matrix notation: $x' = Hx$

It can be shown that we can estimate the matrix h using 4 matching correspondences between x and x' where each correspondence induces a pair of equations to the linear system (assuming $h_{33} = 1$):

$$\begin{pmatrix} x & y & 1 & 0 & 0 & 0 & -x'x & -x'x \\ 0 & 0 & 0 & x & y & 1 & -y'x & -y'y \end{pmatrix} \begin{pmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{pmatrix} = \begin{pmatrix} x' \\ y' \end{pmatrix}$$

Note that this points correspond to the original matrices (images) in euclidean coordinates, not homogeneous coordinates.

This was programmed as a function that can receive 4 or more points and solves the linear system using numpy.linalg.lstsq [3] (least-squares solution) to gain robustness.

B. Bilinear interpolation

After finding the homography matrix H we can easily map each point from one image to the other using the above equation $x' = Hx$, however this implies a problem because all the coordinates in the images are integers but after the transformation they can be real numbers. To solve this problem, each point in the destination image is computed as an estimation of its corresponding point in the origin image.

```
for y in range(destination.shape[0]):
   for x in range(destination.shape[1]):
      p = transform_point((x, y), H)
      # interpolate_point receives the point
          and the matrix to interpolate from.
      ans[y][x] = interpolate_point(p,
          original)
return ans
```

In this case, I used a bilinear interpolation [4] as the estimation method for the experiments.

II. Experiments

This section describes a set of experiments to work with linear projective transformations. All the experiments were done in python3.6 using OpenCV to read and write the images, and numpy to perform mathematic operations with the data.

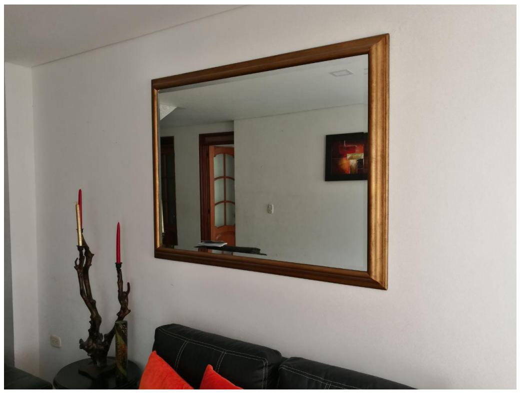


Fig. 1: Original image with perspective

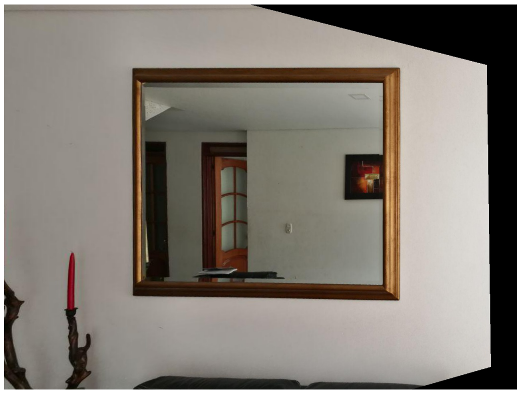


Fig. 2: Removed perspective using linear projective transformations

A. Remove perspective

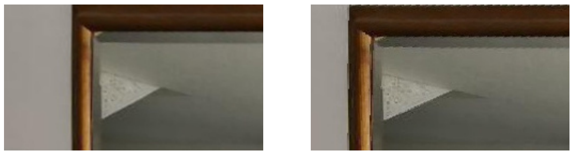
This experiment takes a picture of a real scene (fig 1) and removes the projectivity on the image (fig 2). To perform this, the pixels of the 4 corners of the mirror were selected and the homography matrix was estimated. Then the process to reconstruct the "destination" image was done using the bilinear interpolation.

In this task I used two different estimations of the pixels to compare their performance, they are shown in the figure 3. Both were generated from the same image, (a) was using a bilinear interpolation, and (b) was taken the nearest integer pixel to the real coordinates.

B. Scanner

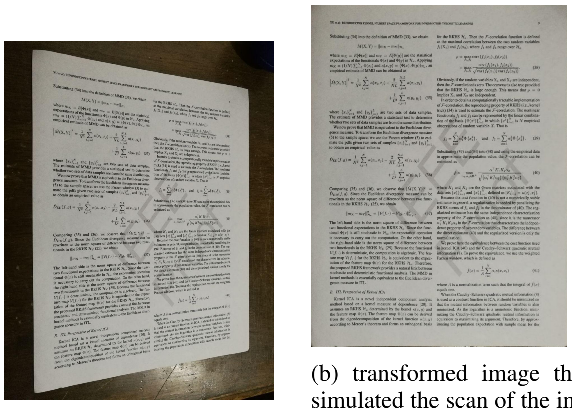
In this section, the same program was used to simulate a scanner, in this case the reference points were the corners of the paper sheet and the destination coordinates where the dimensions of the image.

It would be very interesting to merge this application with the corner detection algorithms seen in class, to automate the process.



(a) Bilinear (b) Nearest integer pixel

Fig. 3: Pixel estimation



(a) Original picture

(b) transformed image that simulated the scan of the image

Fig. 4: Scanner application

C. Add perspective

In this experiment, I took an image "without perspective" (fig 5) and then apply it a projective transformation to embed the image into another image. The result of this is the fig 6.

As the reconstruction algorithm iterates over the destination image, it is necessary to determine if a pixel is inside of the

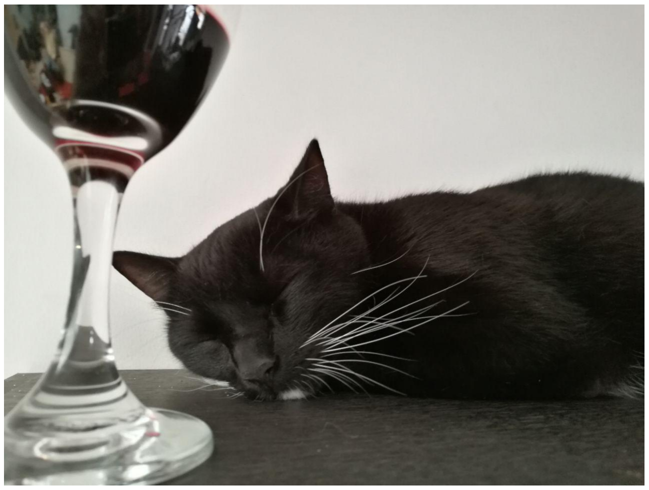


Fig. 5: Image "without perspective"

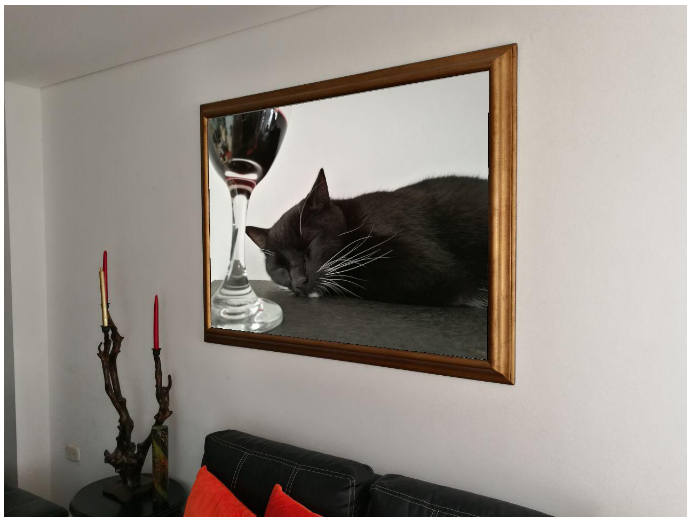


Fig. 6: Embedded image into another scene

"embedding area". To achieve this, I wrote another auxiliary function to determine if a point is inside a convex polygon, this function basically checks that the point is always to the same side of the edges of the polygon. The side of the point depends if the edges are traversed clockwise or anticlockwise. Besides this, no other code different from the one that was already used was necessary.

III. Conclusion

The theory about 2D projective transformations worked as expected.

It is very important to use robust methods to solve the linear system of the homography, with this we can add more matching points and avoid errors or wrong transformations.

The technique used to estimate can highly increase (or decrease) the quality of the results as we saw in the fig 3, other future improvements of this work could be to implement different estimation algorithms, they can be easily added to the reconstruction.

As result, I constructed a basic library to work with planar projective transformations that can be used from other programs.

References

[1] Wikipedia. Homogeneous coordinates — wikipedia, the free encyclopedia, 2017. [Online; accessed 17-September-2017].

[2] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, New York, NY, USA, 2 edition, 2003.

[3] numpy. Least-squares solution to a linear matrix equation, 2017. [Online; accessed 17-September-2017].

[4] Wikipedia. Bicubic interpolation — wikipedia, the free encyclopedia, 2017. [Online; accessed 16-September-2017].