Affine and Metric Rectification

Michael Cruz
University of Central Florida
Orlando, USA
mi484725@ucf.edu

1 Introduction

When an image is taken, objects in the image can become warped due to many factors such as camera angle or noise such as light from the sun. The subtle changes to the image, often unnoticeable to the naked eye, are caused from the projection of 3D objects in the real world to a 2D space. This projection leads to subtle changed to the image, such as parallel lines seeming to intersect at a distant point or objects not appearing as they do in the real world. By projecting a 3D space to a 2D space, the objects in images go through a linear transformation, often referred to as a homography, that warps them. Rectification is a fundamental operation in 3D computer vision that reverses the transformation that skewed the image. This processed is completed by mathematically finding the inverse of the homography that transformed the image, and applying the new homography to the image, rectifying the previous the alterations caused by the initial transformation. This allows scientists to perform measurements on images, which enables them to do 3D reconstruction. I implemented code in Matlab that performs both affine and metric rectification. In this paper, I will explain the methods I used to accomplish both tasks and the results of the process.

2 Implementation

As previously discussed, two programs were created to affine and metric rectification named Affine.m and Metric.m respectively. Both pieces of code take an image and perform operations to mathematically calculate a homography that reverses the transformation that warped the original scene. In the next few sections, I will discuss the implementation of each program in more detail.

2.1 Affine Rectification

The affine rectification program rectifies images which have gone through a projective transformation. Once rectified, all parallel lines in the image no longer meet at a vanishing point, and the line at infinity is put in its correct canonical location at coordinates (0, 0, 1). To do this the following steps are performed for each skewed image:

(1) Import the image to be rectified

(2) Find two pairs of parallel lines in the image that are perpendicular to each other

(3) Take the cross product of each pair of parallel lines to find the two vanishing points v1 and v2 of the parallel lines at the line at infinity

(4) Take the cross product of v1 and v2 to compute the coordinates of the line at infinity in the image and normalize the coordinates by dividing all coordinates by the last coordinate

(5) Construct the rectification homography which is a matrix comprised of 3 1x3 vectors stacked together, with the last vector being the coordinates for the line at infinity previously computed

(6) Apply this homography to the image and rectify it up to affinity

These steps were completed via four functions: GetImage(), GetLineInf(), ConstructHomography(), and TransformImage(). I will briefly discuss the role of each function.

2.1.1 GetImage. This function enables the user to enter the name of the image file they wish to rectify and opens the image with the built-in imread function. Once the image has been read into the workspace, it is returned to be used for the rest of the program.

2.1.2 GetLineInf. This function accomplishes steps 2-4 previously described in the rectification process. The image from the GetImage function is passed in. First, the user is instructed to select 2 points on any line in the image. Next the user should select two points on any line that is parallel to the previous line chosen. Finally, the user should repeat these two steps for another pair of parallel lines, however the next pair need to be perpendicular to the previous pair of parallel lines. Using these coordinates chosen by the user, the line coordinates for the line at infinity in the image is computed and returned.

2.1.3 ConstructHomography. After the line at infinity is computed, it is passed as a parameter to this function. The rectifying homography is constructed by stacking the following vectors from top to bottom: (1, 0, 0), (0, 1, 0), (L1, L2, L3), where L1, L2, and L3 represent the coordinates of the line at infinity. This matrix can be used to rectify the input image up to affinity, and the transpose of the matrix is passed to the function TransformImage to rectify the actual image.

2.2 Metric Rectification

Metric rectification is a process that, like affine rectification, rectifies an image that has gone through a projective transformation. However, metric rectification rectifies the image up to similarity, which enables scientists to perform all the measurements unlocked through affine transformation in addition to the ratio of lengths, angles, and the ratios of areas. This can be done by finding the circular points and using them to construct the dual conic. Then, by taking the singular value decomposition of said conic, a rectifying homography can be constructed. These are the steps taken in the program to accomplish this goal:

(1) Import the image to be rectified

(2) Find two pairs of parallel lines in the image that are perpendicular to each other

(3) Take the cross product of each pair of parallel lines to find the two vanishing points v1 and v2 of the parallel lines at the line at infinity

(4) Find five points along any single circle in the image.

(5) Construct a homogeneous representation of a conic with the coordinates of the five points found

(6) Calculate the two points c1 and c2 at which the conic intersects the line at infinity via the vanishing points v1 and v2

(7) Using c1 and c2, calculate the dual conic through with a linear transformation of the two points

(8) Calculate the singular value decomposition of the dual conic, resulting in three matrices U, D, and V^T

(9) Take the square root of each of the diagonal elements of D and use the values to construct a new matrix D', which is another diagonal matrix with the square root values

(10) Multiply U and D' to construct the rectifying homography H and return the inverse of H

(11) Apply the transpose of the homography and rectify the image up to similarity

These steps were performed via these functions: GetImage(), GetVanishingPoints(), GetConicPoints(), conicfit(), CalculateDualConic(), and ConstructHomography(). Some function are the same as the functions used in affine rectification. I will describe any function novel to this implementation in the next sections.

2.2.1 GetVanishingPoints. This function uses the same process as the function GetLineInf from the previous section. However, instead of calculating the line at infinity, the two vanishing points v1 and v2 are returned from the function.

2.2.2 GetConicPoints. In this function, the user is instructed to click on five points along any singular circle in the image. After selecting the points, homogeneous representations are created of the coordinate of each point. Then a matrix is constructed with the transpose of each point. This matrix is returned and will be used to find construct the homogeneous representation of the conic.

2.2.3 CalculateDualConic. To calculate the dual conic, this function accepts the a homogeneous representation of a conic C and two vanishing points V1 and V2. First a systems of equations is set up to find solutions for the following quadratic equation:

$X^2 V_2^T C + 2X V_2^T C V_1 + V_1^T C V_1 = 0$

Two solutions are computed and by multiplying each point by V2 and adding the product to V1, the circular points C1 and C2 have been found. Using those two points, the dual conic is constructed by the linear combination:

$C_1 C_2^T + C_2 C_1^T$

The dual conic is then returned.

2.2.4 ConstructHomography. Unlike in affine rectification, this homography is constructed by first computing the singular value decomposition of the dual conic, which it takes in as input. This results in three matrices: two unit matrices U and V and a matrix of singular values D. Then, the square root of the diagonal elements of the D matrix are used to construct a new matrix D', which uses those values as it's diagonal elements. By multiplying U by D', and taking the inverse of that new matrix, the rectifying homography is constructed and returned to rectify the image up to similarity.

3 Results

Both programs were able to rectify several different images up to their designated transformations. The resultant images were noticeably changed, however the invariants of each transformation became evident after the transformation. Often times, the entirety of the initial image was not longer in view, and replaced with black pixels. Also, the rectified photos were sometimes flipped, rotated, shrunk, or stretched. However, the rectification seemed to eliminate distortions created through projection, and the result seemed much more like a 3 dimensional representation.

3.1 Problems

There were some issues that hindered the rectification process of each program. For instance, for affine rectification especially, it was crucial that you chose lines that you know to be parallel in the real world or the result would not properly rectify the image. In the example image Building.jpg, I was unable to perform affine rectification. I assume the problem must have either been that hit was more difficult to select points on a line. The space separating the parallel lines may have also caused issues. However, as long as there were clearly defined lines, preferably in a single square, the rectification was seamless. As for metric rectification, there was no way to perform the process if there was no circles in the the image. This is because I chose to implement the stratified method in which the image is first affine rectified, then metricly rectified. Implementing the non-stratified method would require me to find many pairs of parallel lines that are also all linearly independent of each other, which is very difficult to do for any image. Therefore, I was unable to do metric rectification on the sample image Building.jpg as the image had no circles in it.

4 Conclusion

In all, both rectification processes were not too difficult to implement, however there are intricacies that made figuring out the exact process hard. Metric rectification was definitely the hardest to implement, mainly because the circular points are large and complex, so there was no way to ensure my results were correct without running the entirety of the code. Nonetheless, both programs accomplish the task and produced satisfactory results on the majority of the sample images given in the assignment.