Image Stitching and Mosaicking

Sharif Anani
South Dakota School
of Mines & Technology
501 E. St. Joseph St.
Sharif.Anani@mines.sdsmt.edu

Abstract

In this project, we are to use what we have been taught in the class to create and utilize homographies. Homographies are matrices that are used to describe a 2D transform between two images. One use of homographies is image stitching. Panorama images became very popular after mobile phones became capable of stitching images together to create a wider field of view imitating that of the normal human eye. In this project, we go through the steps of computing the homography and applying it to create a new image that results in a panoramic view.

1. Introduction

Image stitching and mosaicking has been achieved for a while now, and it cam be found on today's moderate smartphone. In this project, we attempt to create a mosaic from two images that are taken from the same scene. The result should be an image with a wider field of view covering the total FOV from both images.
This serves as one of the most basic motivations for developing image mosaicking algorithms. Image mosaicking algorithms are also used in a multitude of other things

2. Basic Approach

Image transforms can be computed with a minimum of one point. The number of points can, however, restrict the type of transforms possible.
For example, to translate an image, one can use one point. The coordinate information $[x, y]$ is sufficient to calculate a transformation matrix $H^{3,3}$ that when multiplied by a point, results in the coordinates transferring from $(x, y, 1) \rightarrow (x^{'}, y^{'}, 1)$. For an affine transform transformation, we need a minimum of four points.
As with any transformation, to solve for the transformation we can solve the equation:

$$\alpha P = HP^{'} \tag{1}$$

Where:

1. α is a scale factor
2. P is the point
3. H is the transformation matrix
4. $P^{'}$ is the projection of the point

Since the vectors $P\&P^{'}$ are homogeneous, then they are guaranteed to have the same direction, but can have different lengths. This means that we can have another equation:

$$P \times HP^{'} = 0 \tag{2}$$

To be able to solve for the transformation matrix H, we can use equation 2 and rewrite it in a way that allows us to solve it.
If each row in H is denoted as h^j, then HP can be written as:

$$HP = \begin{bmatrix} h^1 P \\ h^2 P \\ h^3 P \end{bmatrix}$$

And if $P^{'} = [x^{'}, y^{'}, w]^T$ then for each correspondence:

$$P' \times HP = [y^{'} h^3 - w^{'} h^2 P \tag{3}$$

which can be rearranged into the equation:

$$\begin{bmatrix} 0^{1\times 3} & -w_i^{'} P_i^T & y_i^{'} P_i^T \\ -w_i^{'} P_i^T & 0^{1\times 3} & -x_i^{'} P_i^T \end{bmatrix} \begin{bmatrix} (h^1)^T \\ (h^2)^T \\ (h^3)^T \end{bmatrix} = 0 \tag{4}$$

If we to this for four points, we have an equation in the form of $Ah = 0$ where h is 9×1 and A is 9×8. We can obtain the solution by finding the nontrivial solution to the null space of A. We know the solution will be of size 9×1, because $rank(A)$ is at most 8.
The solution h will contain the entries of the transformation matrix H, which we can obtain from reshaping h.

2.1. Obtaining the best homography

Obtaining the homography can be done in two ways.

- Using human interference
- Automated using features

The first method uses `Matlab`'s `ginput` function to obtain four point coordinates $[x, y]$ and their matching points $[x', y']$
For the second method, we can use `Matlab`'s `detectSURFFeatures` and match the points. We can also use the scripts developed in the previous project.

2.1.1 Using More Than Four Correspondences Using RANSAC

To use more than four correspondences to find the best homography, we can generate a random number of homographies (i.e. 500 homographies), each generated from a four random correspondences of the generated correspondences. We can then use each homography to inverse-warp all the other matching features. An ideal homography should land each feature at its correspondent feature, but this is not the case in practice.
We can take the squared distances between the inverse-warped features and their correspondences and then count the inliers within a tolerance, and pick the homography that results with the most inliers.
This process is known as Random Sampling and Consensus (RANSAC). In the code folder, this is implemented in `homogRANSAC.m`. One feature noticed about RANSAC - at least in this implementation - is that depending on the homographies and number of matches found, it does not always yield the same homography as the best one. this can be mended with a ratio-test-analogue type of test where the algorithm will guarantee the returned homography is at least however many folds better than the next one. The following code illustrates how the best homography is returned in this project.

```
function [Hres] = homogRANSAC(locs1, locs2, numIter)
A = zeros(8,9);
H = cell(1,numIter);
Hinv = H;
for j = 1:numIter
    i=1;
    k=1;
    c = floor(rand(1,4)*length(locs1))+1;
    while(i<=7)
        x1 = locs1(c(k),1);y1=locs1(c(k),2);X1=locs2(c(k),1);Y1=locs2(c(k),2);
        A(i,:) = [x1,y1,1,0,0,0,-x1*X1,-y1*X1,-X1];
        A(i+1,:) = [0,0,0,x1,y1,1,-x1*Y1,-y1*Y1,-Y1];
        k=k+1;
        i=i+2;
    end
    N = null(A);
    h = [N(1),N(2),N(3);
        N(4),N(5),N(6);
        N(7),N(8),N(9)];
    h=h/h(end,end);
    H{1,j} = h;
    hi = inv(h);
    hi = hi/hi(end,end);
    Hinv{1,j} = hi;

end
jMat = cell(2,size(Hinv,2));
for j = 1:size(jMat,2)
    jMat(1,j)=Hinv(j);
    X = locs2(:,1); Y = locs2(:,2);
    P = [X';Y';ones(1,length(locs2))];
    Pp = Hinv{j}*P;
    for i = 1:size(Pp,2)
        v=Pp(:,i);
        s=v(3);
        Pp(:,i)=v/s;
    end

    Xp = Pp(1,:)'; Yp = Pp(2,:)';
    locs2p = [Xp, Yp];
    diffxy = locs1-locs2p;
    diff = (diffxy(:,1).^2 + diffxy(:,2).^2);
    a=find(diff<1);
    a=numel(a);
    jMat{2,j} = a;

end
a = jMat(2,:);
a=cell2mat(a);
[val,ind]=max(a);
Hres = cell2mat(H(ind));
end
```

3. Mosaicking

After inverse warping one of the images into the frame of the other, we are left with another problem, how do we

actually create a mosaic of both images? The two images need to be aligned in a manner such that one completes the other, and to do that, we need to calculate an offset that will translate the first image into the newly created canvas containing the warped image.
To calculate the offset, we can use the inverse warped corners that were transferred from the second image into the first. The offset in both the width and height coordinates will be the minimum between the warped corners and zero, whichever the smallest without being a negative number. Subtracting this offset and using those coordinates on the newly created canvas to position the first image will place it where it needs to be in order for the two images to be properly mosaicked, leaving only the parts of the warped image that do not exist on the second visible.

3.1. Summary of algorithm

The algorithm can be summarized in the following steps:

1. Find matches on both images
2. Find the best homography using RANSAC
3. Inverse warp the the corners of the second image into the first
4. From the inverse warped corners, calculate the size of the new image to hold both images
5. Create a meshgrid of that size
6. Interpolate the second image into the newly created image
7. Calculate the offset of the first non-warped image
8. Copy the non-warped image into the frame

4. Example Results

The following results show some of the obtain results. Figure 1 shows the SDSM&T quad looking out of campus, the artefacts are visible because of the difference in lighting, which can be noticed because the sun hits one of the images very differently than the other.

Figure 2 shows the mosaic from two images of lake sylvan. The artefacts are much less visible because of more forgiving lighting.

Figure 1. Mosaic of two images from the SDSM&T Quad

Figure 2. Mosaic of two images from the Sylvan Lake

Another result showcasing the manual operation of the user choosing the points for warping is shown in Figure 3. This can be useful for applications in TV broadcasts where graphics can be overlaid over the video. Homographies of this kind can also be usefulfor visual odometry with the knowledge of other parameters such as the camera parameters and distance from object, then distances can be measured on the image.

Figure 3. ECE students framed on the lab whiteboard

The final result is one taken at the Colorado State University (CSU) campus. The result from this sequence falls in between the two, as alignment is fairly good, but again, color

adjustments are needed to hide the edge where the two images meet. The result is shown in 4

Figure 4. Mosaic at CSU

A collection of all the result images is available on the gitHub page of the project, which is reachable through `https://github.com/sharifanani/ComputerVisionF15/blob/master/Project%202/Paper/Results.tar.bz2`

5. Further Improvement

There are many areas that this mosaicking method can be improved in; from image blending to having consistently better matches. One are of application for this is to take it to mosaicking depth (or disparity) maps, which takes the applications to another spectrum of the computer vision industry.

6. Conclusion

All in all, this project was a great way to apply what was learned in the classroom, and a good method to develop better intuition in computer vision problems. More importantly, it is the first building block for a bigger project that includes mosaicking disparity maps to create a wide view disparity to be used in automotive All code and images can be found on `https://github.com/sharifanani/ComputerVisionF15/tree/master/Project%202`
All Information on how to achieve these results can be found in [1]

References

[1] R. Szeliski. *Computer Vision (Texts in Computer Science)*. Springer, 2013.