Creating Panoramas from Multiple Pictures
CS 374 Special Topics in Computer Science: Image and Video Understanding, Final Project

Matt Bellis and Helen Jiang, Boston College 2013

Abstract
We wanted to create a program that would accept two to five pictures and merge them together into one larger picture, called a panorama. The input pictures would be from the same location, but at different angles of viewing, and the output picture would encompass all of the angles of the input pictures.

1. Introduction
To make a panorama there are several steps. First we needed to match features. Then we needed to match images. After this we had to merge all of the pictures together. Finally we used multi-band blending to make the pictures look cleaner.

Figure 1: Example input pictures to our program

Figure 2: Output picture from our panorama creator
2. Strategy
The task is broken down into four steps. Feature matching, image matching, bundle adjustment and multi-band blending.

2.1 Preprocessing
First we created a function to read a number of images from an input file. Then these images were converted to doubles and resized to 25% of their original size for fast computation. We used 25% because anything smaller began to negatively impact the feature matching.

2.1.1 Code
```matlab
function [images] = readFromFolder(filePath)
    fd = pwd;
    cd(filePath)
    imagefiles = dir('*.jpg');
    nfiles = length(imagefiles); % Number of files found
    for i=1:nfiles
        currentfilename = imagefiles(i).name;
        currentimage = imread(currentfilename);
        currentimage = im2double(currentimage);
        currentimage = imresize(currentimage, .25);
        images{i} = currentimage;
    end
    cd(fd);
end
```

2.2 Feature Matching
We used Scale Invariant Feature Transform (SIFT) to find all the important features from a given photo. We used the sift code courtesy of David Lowe. We needed to use SIFT because our features could be shifted in scale because of the angle they were being viewed. SIFT helped us negate this problem when finding important features. Before performing SIFT we preprocessed the input images to make them 25% of their original size. This means less features which computes quicker.

Figure 3: The SIFT features of the first example input picture.
2.3 Image Matching
Our program requires the images to be in order to work properly. We compared the features of each picture with the picture after it. We used a random sample consensus (RANSAC) to compare the SIFT features of each photo. RANSAC takes three random points from one picture, and uses them to project points onto the second picture. It then computes the cost of these points, or how accurate the projection is. This is repeated a number of times, and the lowest cost result is saved and used.

![Image 1](image1.png)

Figure 4: The matching SIFT points of the previous picture. This is the second example input picture.

2.3.1 Code
We created a function called getPoints which accepted two picture inputs, and returned the two images as well as 8 point pairs from the feature points.

```matlab
function [pvec im1 im2] = getPoints(fname1, fname2)
%GETPOINTS Summary of this function goes here
% Detailed explanation goes here

im = fname1;
im1 = im;
imwrite(im, 'tmp.pgm');
[im1, feature1, loc] = sift_windows('tmp.pgm');
p_s = loc(:,1:2);

im = fname2;
imwrite(im, 'tmp.pgm');
```

2.4 Bundle Adjustment
To warp one picture to another we needed to calculate the homography and use a projective transformation to stitch two photos together. A projective transformation needs 4 matching point pairs, which our function getPoints calculated based off the SIFT feature and RANSAC. We created a function called warp_two which took two pictures and 8 points pairs as an input and returned the two images combined. The warp_two function only warped pictures from the right,
so to solve this we flipped the images upside-down before calculating the SIFT features, warped them, and then flipped them back if we need to warp from the left.

2.4.1 Code
(Note that this code also performs the multi-band blending, as it was more efficient to do so at this step.)

```matlab
function [finalImage] = warp_two(image1, image2, points)
    im1 = image1;
    im2 = image2;
    x1 = points(:, 1); y1 = points(:, 2);
    x2 = points(:, 3); y2 = points(:, 4);
    %save xy.mat x1 y1 x2 y2;

    load xycs.mat;

    ax = [-x1, -y1, ones(size(x1)), zeros(size(x1)), zeros(size(x1))];
    ax = [zeros(size(x1)), x2.*x1, x2.*y1, x2];
    ay = [zeros(size(x1)), zeros(size(x1)), zeros(size(x1)), y2.*x1, y2.*y1, y2];
    A = [ax; ay];

    [U,S,V] = svd(A);
    H = reshape(V(:,9), 3, 3);
    H = H';
    p = [x1';y1';ones(size(x1))];
    q = H'*p;
    x = x1; y = y1;
    a = floor(min(x,y));
    b = floor(max(x,y));
    mask = roipoly(imx, [cx x0, cy y0], [cx a, cy b]);
    imwrite(mask, 'im3.jpg');
    mask = cat(3, mask, mask, mask);
    im1 = imread('im1.jpg');
    im1r = double(im1(:,:,1));
    im2 = imread('im2.jpg');
    im2r = double(im2(:,:,1));
    Cr = BlendArbitrary(im1r, im2r, im3/255, 4);
    Cg = BlendArbitrary(im1g, im2g, im3/255, 4);
    Cb = BlendArbitrary(im1b, im2b, im3/255, 4);
    Cr = imwrite(Cr, 'Cr.jpg');
    Cg = imwrite(Cg, 'Cg.jpg');
    Cb = imwrite(Cb, 'Cb.jpg');
    finalImage = imx.*mask + imy .* (1 - mask);
end
```
2.5 Multi-band blending
To make the pictures look better we used multi-band blending to soften the edges where the pictures met. Often changes in lighting or other small tears would be noticeable (due to RANSAC and warping imperfections).

2.5.1 Code
We used and modified open-source code from http://cvlab.epfl.ch/~tola/open_source.html to perform the blending. We blended each picture as we warped them (see 2.4.1 for the code).
2.6 Merging all the images
We created a function that checked the number of pictures to be merged and wrote specific cases for each number. Our panorama creator can make any number between 2-5 pictures. We felt that any more than five photos would not only take too long, but be unnecessarily large. To merge two images that were already warped, we have to merge them from the already non-warped size. This meant we often had to flip images as described before and warp from right to left instead of left to right. A center picture was used for any odd number of pictures, and two center photos were merged for any even number.
2.6.1 Code
This function accepts a folder name and returns a panorama. The pictures must be in order from left to right.

```
function [finalPicture] = createPan(folderPath)
%CREATEPAN Makes a panorama out of several pictures
% Input argument for path of pictures, in order left to right
images = readFromFolder(folderPath);
if n == 1 | n > 5
    disp('Cannot do that number of pictures, error!');
    finalPicture = 1;
    return
end
if (n == 2)
    [p a b] = getPoints(images{1}, images{2});
    finalPicture = warp_two(b, a, p);
    return
end
if (n == 3)
    left = images{1};
    mid = images{2};
    right = images{3};
    [p a b] = getPoints(mid, left);
    newIm = warp_two(b, a, p);
    return
end
if (n == 4)
    left = images{1};
    mid = images{2};
    midr = images{3};
    right = images{4};
    lu = flip(left);
    mlu = flip(mid);
    [p a b] = getPoints(mlu, lu);
    leftNew = warp_two(b, a, p);
    leftNew = flip(leftNew);
    % figure(1)
    % imshow(leftNew)
    % [p a b] = getPoints(midr, right);
    rightNew = warp_two(b, a, p);
    % figure(2)
    % imshow(rightNew)
    % [p a b] = getPoints(leftNew, rightNew);
    finalPicture = warp_two(b, a, p);
    return
end
if (n == 5)
    left = images{1};
    leftmid = images{2};
    mid = images{3};
    rightmid = images{4};
    right = images{5};
    lu = flip(left);
    lmu = flip(mid);
    mu = flip(mid);
    nu = flip(muid);
    [p a b] = getPoints(lmu, lu);
    newLeft = warp_two(b, a, p);
    % figure[1]
    % imshow(newLeft)
    [p a b] = getPoints(mid, newLeft);
    newIm = warp_two(b, a, p);
    newIm = flip(newIm);
    % figure[2]
    % imshow(newIm)
    [p a b] = getPoints(rightmid, right);
    newRight = warp_two(b, a, p);
    % figure[3]
    % imshow(newRight)
    [p a b] = getPoints(newIm, newRight);
    finalPicture = warp_two(b, a, p);
    return
end
end
```

3. Drawbacks
Each picture must be taken a certain way for this program to work. Each photo needs to have a similar feature of the proceeding one for them to be merged. The photographer must be standing
in place and rotating, only changing the angle for each picture. These drawbacks are similar to other panorama creators though. The code can be run fairly quickly, but large pictures with thousands of features can take a minute or two to complete. The program usually creates a good panorama, but due to RANSAC and SIFT limitations it can fail if the input pictures are not good enough. A picture with not enough features will fail. Anything that would normally perform poorly with SIFT and RANSAC cannot perform well.

4. Finished Examples