Assignment 2

Team- Divya Lalwani and Vidhixa Joshi

Code Organization:

We have following files:

1. CommonFunctions.h: This file has common functions like get_gaussian, convolve_rows, transpose, convolve_separable, multiply (it multiplies respective pixels of both input images) , sub(it multiplies respective pixels of both input images), kTimesTrace, Calygrad, Calxgrad.

2. Harris.h : This file has a function named findR, which is used to calculate Harris R Score i.e. Response score.

3. ProjTransform.h : This file has a function named transform, which performs projective transformation using bilinear interpolation.

4. Feature_det : It consists of find_theta function which computed the inverse tan of gradients and brings them in a range of -180 to 180 which is further divided into 8 orientations for each window that we take in finding feature descriptors.

Step performed:

Step 1:
Files were successfully untared and run on Tank linux machines.

Step 2:

- First step that we did was to perform smoothening on the image using Gaussian filter and sigma = 1.0. Since 'lincoln.png' didn't have much noise, sigma = 1.0 was enough. We observed that values greater than 1.0 made image blurred, so it was difficult to detect corners.
- Then we have performed Harris Corner Detection as described on page 27 of this pdf: http://www.cse.psu.edu/~rcollins/CSE486/lecture06.pdf.)

Here is the brief description of the steps performed:

1.) Convolve input image with x-gradient and y-gradient Sobel using convolve_separable function. This will give us Ix and Iy.
2.) Compute Ix2, Iy2 and IxIy by squaring Ix, Iy and multiplying Ix with Iy respectively.
3.) Compute Six2, Siy2 and Sixiy by convolving Ix2, Iy2 and IxIy with Gaussian filter (sigma = 3.0 gave best results on trial and error method). Now, we have our required matrix:

H =

Six2	Sixiy
Sixiy	Siy2

4.) Now compute determinant pf this matrix. Compute AB(multiplication of Six2 and Siy2) and C2(square of Sixiy) and then Determinant = AB – C2.

5.) Compute k*trace using function kTimesTrace. Trace is (Six2 + Siy2), we take square of this term and multiply this value with k = 0.05 (which worked best on trial and error basis).
6.) Finally, we have all terms to calculate R, which is:
R = Det (H) – k*(Trace(H))square
7.) We now call find_corners function passing R as parameter. We then perform thresh holding on value of R using value 0.05. We also perform non maxima suppression using a window of 25. If R value of current pixel is maximum in it's neighborhood window, then we push this point to our vector.

Finally, we display corners using overlay_rectangle function on above points. The result of this step could be seen in hcorner.png ouput image.

Step 3:

- In this step, we were required to create a feature descriptor for every corner that we get from the previous step.
- We compute the inverse tan from gradients using 'find_theta', 'Calxgrad' and 'Calygrad'.
- We take a window of 8*8 pixels on top left, top right, bottom left and bottom right of the corner.
- We create a descriptor vector which increments the bin value according to the theta detected at the point. It's a 32 length descriptor.
- We observed that for most pixels close by, the angle values were close to each other and thus the bins looks similar. This will help us in matching features with other image.

Step 4:

We first tried to perform projective transformation using forward warping. But as discussed in lecture, it gave some sort of holes in the image as shown below:

So, we have used inverse warping to solve this issue. First we have calculated the inverse of given transformation matrix and hardcoded it's value. Let's say the inverse is:

a	b	c
d	e	f
G	h	i

- So, the formula to calculate position of input image pixel using homogenous coordinate system becomes:

  Y = (Y'*a + X'*b + c) / (Y'*g + X'*h + i)

  X = (Y'*d + X'*e + f) / (Y'*g + X'*h + i)

- Where X' and Y' are row and column positions of output image.
- Since the values of x and y are somewhere in between the actual pixels, we use bilinear interpolation to calculate the intensity at actual pixel position using below formula:

$$
\begin{aligned}
f(x,y) \approx \quad & \frac{f(Q_{11})}{(x_2-x_1)(y_2-y_1)}(x_2-x)(y_2-y) + \\
& \frac{f(Q_{21})}{(x_2-x_1)(y_2-y_1)}(x-x_1)(y_2-y) + \\
& \frac{f(Q_{12})}{(x_2-x_1)(y_2-y_1)}(x_2-x)(y-y_1) + \\
& \frac{f(Q_{22})}{(x_2-x_1)(y_2-y_1)}(x-x_1)(y-y_1) \\
= \quad & \frac{1}{(x_2-x_1)(y_2-y_1)}\Big(f(Q_{11})(x_2-x)(y_2-y) + \\
& \qquad f(Q_{21})(x-x_1)(y_2-y) + \\
& \qquad f(Q_{12})(x_2-x)(y-y_1) + \\
& \qquad f(Q_{22})(x-x_1)(y-y_1) \Big)
\end{aligned}
$$

(Reference: http://en.wikipedia.org/wiki/Bilinear_interpolation)
Where x2, y2 are ceiling values and x1, y1 are floor values of x and y.
In this step, we are cutting down the pixels which fall outside the range of the input image.

Step 5:
We tried to find correspondence between two images using two different methods, namely
1. Normalized cross correspondence(NCC)

NCC had the following steps:

1. First find the xy gradient of each feature point on both the images
2. Normalize it by considering a window around the point and taking mean intensity.
3. For each feature point in first image, we compare it with all features in second image.
4. Matching takes place by sum of squared distances method.
5. Closest match will have the maximum value of SSD.

We have found the correspondences using this.
2. SIFT -- Finding correspondences using the feature descriptor. We couldn.t successfully implement this. Thus we commented it in the code.

On the basis of corresponding points that we take in a vector<Coordinate>, we take four points to compute RANSAC. We take the collinear corresponding points in both the images and create matrix. This code we commented, because it showed issues that we couldn't resolve.

Step 6:

- We have generated sequence of images using the transformation matrix given in step 4.
- Images can be found in the zip folder.