Computer Graphics – Spring 2016

Exercise 2 – Features, Transformations, RANSAC, Edge Detection

Submission: Monday, May 16th 2016 at 23:55

1) Implement a function that computes a projective transformation of an input image. The function should have the following signature:

   `[TranformedIm] = ComputeProjective(Im, H)`

   where `Im` is the input image and `H` is the transformation.

   For example:


```
H=[1 .2 0; .1 1 0; 0.5 0.2 1]
```

   Show your example of a pair of images and specify the transformation between them.

2) Extract and match interest points from the pair of images from 1) using the SIFT detector and descriptor. Matlab code for extracting and matching can be found in: http://www.cs.ubc.ca/~lowe/keypoints/.

   Implement a new match function that has the following signature:

   `[num_matches,matches,dist_vals] = match(image1, image2, distRatio)`

   where `matches` is a matrix containing all the correspondences, `num_matches` is the number of correspondences returned, `dist_vals` are the corresponding ratios, `image1` is the original image from 1), `image2` is the transformed image from 1), and `distRatio` is described in the original `match` function.

   Implement a function that displays a chosen number of the top (ratio-wise) correspondences and has the following signature:

   `[displayedCorr] = DisplayCorr(image1, image2, matches, dist_vals, x)`

where `matches` is a matrix containing all the correspondences, `image1` is the original image from 1), `image2` is the transformed image from 1), `dist_vals` are the corresponding ratios, `x` is the number of top correspondences, and `displayedCorr` is a matrix containing the `x` correspondences displayed.

Display and explicitly write the top correspondences obtained using this technique (and the two functions described above). Display the order of the correspondences according to the match ratio.

For example:

```
230.67,749.49 -> 285.83,708.2
248.53,531.28 -> 273.34,512.68
255.8,437.17 -> 267.88,428.72
443.97,646.63 -> 450.77,641.47
659.47,539.78 -> 617.06,570.51
373.41,758.65 -> 406.33,735.67
166.58,855.98 -> 245.88,797.93
380.97,758.03 -> 412.41,735.71
361.86,713.22 -> 392.86,696.62
392.31,649.38 -> 407.89,638.14
```

3) Implement the normalized DLT (Direct Linear Transformation) algorithm:

Objective

Given n≥4 2D to 2D point correspondences $\{x_i \leftrightarrow x_i'\}$, determine the 2D 3x3 homography matrix H such that $x_i' = Hx_i$ (in homogenous coordinates).

Algorithm

(i) Normalize points x_i: First move their average to (0,0) by substracting their mean. Then scale them to average length of $\sqrt{2}$. The final transformation is: $T = T_{scale} * T_{translate}$

(ii) Do the same to points x_i', and obtain the transformation T'

(iii) For each correspondence $x^i \leftrightarrow x_i'$ compute the matrix A_i (using the transformed points); see explanation and definitions below

(iv) Assemble the n 2x9 matrices A_i into a single $2n$x9 matrix A

(v) Obtain the SVD of A. The unit singular vector corresponding to the smallest singular value is the solution **h**. Specifically, if $A = UDV^T$ with D diagonal with positive diagonal entries, arranged in descending order down the diagonal, then **h** is the last column of V

(vi) Determine the homography H from **h**

(vii) Denormalize the solution: inv(T')*H*T

(see "Multiple View Geometry in computer vision", by R. Hartley and A. Zisserman, p. 92)

See the appendix for more information. Your DLT implementation should have the following signature:

```
[H] = DLT(matches)
```

where `matches` is a matrix from 2) and `H` is the computed homography matrix.

Evaluate the obtained transformation in a qualitative manner. Are there any parameters you can tune in order to improve the results?

4) For evaluating the algorithm, we can use the error in the second image after transferring points from the first image using the computed homography matrix. This is the L1 image distance in the second image between the ground truth location of the point x' and the point Hx to which the corresponding point x is mapped. If $d(x, y)$ is the Euclidean distance between the inhomogeneous points represented by x and y, then the transfer error for the set of correspondences is:

$$E = \sum_i d(x_i', Hx_i)^2$$

Implement the proposed error function using the following signature:

```
[error] = ComputeError(pnts_gt,pnts_computed)
```

Implement the following helper function as well:

```
[pnts_gt,pnts_computed] =
     ComputeTestPoints(H_gt,H_computed)
```

where `H_gt` is the homography from 1) and `H_computed` is the homography from 3).

5) Use RANSAC to try to improve results. Code can be found here (or you can choose to implement the RANSAC function on your own): http://www.csse.uwa.edu.au/~pk/research/matlabfns/
   - ☒ If you choose to use code from the website, use the general purpose implementation of RANSAC (ransac.m).
   - ☒ You can check your implementation by comparing your results to the more specific implementation available also at the website (ransacfithomography.m).

Implement a wrapper for the RANSAC function using the following signature:

```
[H_ransac] = RANSAC_Wrapper(matches, fittingfn, distfn, degenfn, s, t, feedback, maxDataTrials, maxTrials);
```

where `matches` is a matrix from 2), and the rest of the parameters are described in detail in the ransac.m file.

 Explain how you adapted the RANSAC function in order to use it here and how you set the parameters (and why).

 Show an example where RANSAC does not give an improvement compared to the DLT implementation from 3). Explain how you evaluated the results (qualitatively? quantitatively?) How do the errors compare? What did you do/change? Write explicitly which image was used, what was the transformation, and any other parameter necessary in order to re-compute the results.

 Write a script titled `Example1.m` that uses the functions in the previous sections and backs-up your arguments. Display these arguments clearly (either by using the display() function or by displaying a figure or both).
Remember: The code should run on our computer!

 Show an example where RANSAC does give an improvement compared to the DLT implementation from 3). Explain how you evaluated the results (qualitatively? quantitatively?) How do the errors compare? How do the errors compare? What did you do/change? Write explicitly which image was used, what was the transformation, and any other parameter necessary in order to re-compute the results.

 Write a script titled `Example2.m` that uses the functions in the previous sections and backs-up your arguments. Display these arguments clearly (either by using the display() function or by displaying a figure or both).
Remember: The code should run on our computer!

6) **Edge detection**, download the image lena.pmb attached to the exercise and open it as a matrix M. We wish to calculate the gradient of the image. The derivative D_x is given by:

$$\begin{aligned} D_x(i,j) = {} & 2M(i-1,j-1) - 2M(i-1,j+1) + \\ & M(i,j-1) \quad\quad - M(i,j+1) \quad\quad + \\ & 2M(i+1,j-1) - 2M(i+1,j+1) \end{aligned}$$

D_y is calculated similarly. You can think of the derivatives as masks. Now define the gradient matrix $G(i,j) = \sqrt{D_x(i,j)^2 + D_y(i,j)^2}$. Finally normalize G and use imshow() to display the result.

 Write a script titled `EdgeDetection.m` that does the above and prints the final image obtained from lena.pgm.

 Add the resulted image to your PDF and explain the results.

General instructions

1. Submission is in pairs.
2. Submit your zip file through the Moodle site of the course.
   - ☒ **It should contain a PDF/word file with answers to:**

     

     This PDF/word file should contain explanations and figures, as specified above, along with the relevant question number.
   - ☒ It should also contain all the Matlab functions specified by:

     

     along with a JPG image. The two scripts in question 5 should run on our machines, so *include all the Matlab functions necessary in the zip file*!
3. Questions can be posted on the designated forum in Moodle.

Appendix

- Direct Linear Transformation (DLT):

(taken from "Multiple View Geometry in computer vision", by R. Hartley and A. Zisserman, p. 71)

- The homography is a 3x3 matrix, whose rows are $\mathbf{h}^{1\mathsf{T}}, \mathbf{h}^{2\mathsf{T}}, \mathbf{h}^{3\mathsf{T}}$; it relates every pair of corresponding points $\mathbf{x}_i, \mathbf{x}_i'$ (equality up to scale) as

$$\mathbf{x}_i' = \mathbf{H}\mathbf{x}_i = \begin{pmatrix} \mathbf{h}^{1\mathsf{T}}\mathbf{x}_i \\ \mathbf{h}^{2\mathsf{T}}\mathbf{x}_i \\ \mathbf{h}^{3\mathsf{T}}\mathbf{x}_i \end{pmatrix} \qquad \mathbf{x}_i' = (x_i', y_i', w_i')^{\mathsf{T}}$$

- This equation may be expressed in terms of the vector cross product as

$$\mathbf{x}_i' \times \mathbf{H}\mathbf{x}_i = \begin{pmatrix} y_i'\mathbf{h}^{3\mathsf{T}}\mathbf{x}_i - w_i'\mathbf{h}^{2\mathsf{T}}\mathbf{x}_i \\ w_i'\mathbf{h}^{1\mathsf{T}}\mathbf{x}_i - x_i'\mathbf{h}^{3\mathsf{T}}\mathbf{x}_i \\ x_i'\mathbf{h}^{2\mathsf{T}}\mathbf{x}_i - y_i'\mathbf{h}^{1\mathsf{T}}\mathbf{x}_i \end{pmatrix} = 0$$

- This gives a set of three equations on the entries of H (a 9-dimensional vector):

$$\begin{bmatrix} \mathbf{0}^{\mathsf{T}} & -w_i'\mathbf{x}_i^{\mathsf{T}} & y_i'\mathbf{x}_i^{\mathsf{T}} \\ w_i'\mathbf{x}_i^{\mathsf{T}} & \mathbf{0}^{\mathsf{T}} & -x_i'\mathbf{x}_i^{\mathsf{T}} \\ -y_i'\mathbf{x}_i^{\mathsf{T}} & x_i'\mathbf{x}_i^{\mathsf{T}} & \mathbf{0}^{\mathsf{T}} \end{bmatrix} \begin{pmatrix} \mathbf{h}^1 \\ \mathbf{h}^2 \\ \mathbf{h}^3 \end{pmatrix} = 0$$

- Only two equations are independent, therefore we omit the 3rd one:

$$\begin{bmatrix} \mathbf{0}^{\mathsf{T}} & -w_i'\mathbf{x}_i^{\mathsf{T}} & y_i'\mathbf{x}_i^{\mathsf{T}} \\ w_i'\mathbf{x}_i^{\mathsf{T}} & \mathbf{0}^{\mathsf{T}} & -x_i'\mathbf{x}_i^{\mathsf{T}} \end{bmatrix} \begin{pmatrix} \mathbf{h}^1 \\ \mathbf{h}^2 \\ \mathbf{h}^3 \end{pmatrix} = 0$$

  and we re-write the two equations as $A_i h=0$, for a 2x9 matrix A_i

- Finally, we put n images together as: $\begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_n \end{bmatrix} \mathbf{h} = 0$

- Useful Matlab functions:
  - maketform. Note that it gets the homography matrix by transpose as in Matlab they multiply from left – xA and not Ax.
  - Imtransform.