Computer Vision and Image Processing

Final Project: Panorama Registration & Stitching

1 Introduction

In this exercise you will perform automatic panorama generation. The panorama is based on the assumption that different images are related only by rotation of the camera and therefore the motion (or transformation) between the images can be accurately modelled by homography.

The input of such an algorithm is a sequence of images horizontally scanning (due to camera rotation) different parts of a scene, with significant overlap in the field of view of consecutive frames. The output is then a large field of view image of the scene in which all these input images have been combined. You should implement the following steps:

- Registration: The geometric transformation between each consecutive image pair is registered by (1) detecting and matching descriptors between each pair of images and eventually (2) fitting a homography transformation that agrees with a large set of inliers matches using the *RANSAC algorithm*.
- Stitching: The homographies between each pair of frames are used in such a way that transforms the frames into a common single coordinate system are produced. The panorama is then rendered in this common coordinate system by *backwarping* from each frame into a vertical strip of the panorama.

2 Image Pair Registration

In this section we will focus on computing the geometric transformation between a pair of consecutive frames, I_i and I_{i+1}, of some image sequence. For an example of such a frame pair, have a look at the example input image sequences provided in `project1.zip`. These are located in the directory `pro-`

`ject/data/inp/examples`. Although the provided input sequences are of RGB images, in the current registration phase we will only require I_i and I_{i+1} to be grayscale images (load them accordingly).

2.1 Feature point detection, descriptor extraction and matching

At this phase you will be using opencv to match features. The output will be a set of points coordinates in both images, pos1 and pos2, of size nx2 that are most likely matched. Note that these can also include outliers.

2.2 Registering the transformation

We will now use the feature point matches found above to compute the most fitting homography that transforms between the two frames I_i and I_{i+1}. To do this we will first need to implement a function that applies a homography transformation on a set of points. This is the `applyHomography` function you are required to implement

```
function name: applyHomography
% APPLYHOMOGRAPHY Transform coordinates pos1 to pos2 using homography H12.
% Arguments:
% pos1 - An nx2 matrix of [x,y] point coordinates per row.
% H12 - A 3x3 homography matrix.
% Returns:
% pos2 - An nx2 matrix of [x,y] point coordinates per row obtained from
% transforming pos1 using H12.
```

As a reminder to how homographies transform points - given a point (x_1, y_1) in coordinate system 1 and a homography matrix $H_{1,2}$, `applyHomography` should transform these points to (x_2, y_2) in coordinate system 2 in the following way

$$\begin{bmatrix} \tilde{x}_2 \\ \tilde{y}_2 \\ \tilde{z}_2 \end{bmatrix} = H_{1,2} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} \tag{1}$$

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} \tilde{x}_2 \\ \tilde{y}_2 \end{bmatrix} \Big/ \tilde{z}_2 \tag{2}$$

We will now use `applyHomography` in our RANSAC (Random SAmple Consensus) homography fitting code. Let's first recall how RANSAC operates. Given 2 sets of N matched points P_1 and P_2 s.t. $P_{1,j}$

are the x-y coordinates of the jth match in image 1 and $P_{2,j}$ are the x-y coordinates of the jth match in image 2

- Pick a random set of 4 point matches from the supplied N point matches. Let's denote their indices by J. We call these two sets of 4 points in the two images $P_{1,J}$ and $P_{2,J}$.
- Compute the homography $H_{1,2}$ that transforms the 4 points $P_{1,J}$ to the 4 points $P_{2,J}$. As discussed in class, there is a closed form solution that does this. To simplify matters you have been provided with the `leastSquaresHomography` function that performs this step.
- Use $H_{1,2}$ to transform the full set of points P_1 in image 1 to the transformed set P_1' (using the above `applyHomography`) and compute the squared euclidean distance $E_j \equiv ||P'_{1,j} - P_{2,j}||^2$ for $j = 1..N$. Mark all matches having $E_j <$ `inlierTol` as inlier matches and the rest as outlier matches for some constant threshold `inlierTol`.

RANSAC performs several iterations (later denoted `numIters`) of these 3 steps, keeping a record of the largest inlier match set J_{in} it has come upon. Once these iterations are complete the homography is recomputed over the matches J_{in}. To do this you will simply need to rerun `leastSquaresHomography` on these inlying point matches $P_{1,J_{in}}$ and $P_{2,J_{in}}$ and obtain the final least squares fit of the homography over the largest inlier set. Your RANSAC implementation should have the following interface

```
function name: ransacHomography
% RANSACHOMOGRAPHY Fit homography to maximal inliers given point matches
% using the RANSAC algorithm.
% Arguments:
% pos1,pos2 − Two nx2 matrices containing n rows of [x,y] coordinates of
% matched points.
% numIters − Number of RANSAC iterations to perform.
% inlierTol − inlier tolerance threshold.
% Returns:
% H12 − A 3x3 normalized homography matrix.
% inliers − An array containing the indices in pos1/pos2 of the maximal set of
% inlier matches found.
```

To visualize the full set of point matches and the inlying matches detected by RANSAC implement the following display function

```
function name:displayMatches
% DISPLAYMATCHES Display matched pt. pairs overlayed on given image pair.
% Arguments:
% im1,im2 − two grayscale images
```

```
% pos1,pos2 − nx2 matrices containing n rows of [x,y] coordinates of matched
% points in im1 and im2 (i.e. the i'th match's coordinate is
% pos1(i,:) in im1 and and pos2(i,:) in im2).
% inlind − k−element array of inlier matches (e.g. see output of
% ransacHomography)
```

This function should display a horizontally concatenated image `[im1,im2]` of an image pair `im1` and `im2`, with the matched points provided in `pos1` and `pos2` overlayed correspondingly as red dots. Each outlier match, say at index j, is denoted by plotting a blue line between `pos1(j,:)` and the shifted `pos2(j,:)`. Similarly inlier matches are denoted by plotting a yellow line between the match points (see section **??** for an explanation of how you can use the functions `imshow` and `plot` to do this). As an example of how the output should look like, below is the figure resulting from running `displayMatches` on an image pair in one of the provided example sequences together with its match points `pos1`, `pos2` (obtained from `findFeatures` + `matchFeatures`) and inlier index set `inlind` (obtained from `ransacHomography`).

This figure shows the large number of outlier matches the RANSAC algorithm had to deal with. As opposed to the outlier match set (shown in blue) the inlier set (shown in yellow) shows a more consistent motion. Also note that although many features detected in the cloud texture were probably matched correctly, these were marked as outliers. This is probably due to the fact that the clouds had enough time to move between these two shots.

3 Panorama Stitching

Our efforts in the previous section eventually provided us with the set of registered homographies $H_{i,i+1}$, for $i = 1..M-1$, between consecutive frames in a given image sequence of M frames I_i. We would now like to use these homographies to stitch these M frames into one combined panorama frame.

3.1 Transforming to a common coordinate system

The first stage in doing this is to pick a coordinate system in which we would like the panorama to be rendered. We will (somewhat arbitrarily) choose this coordinate system to be the coordinate system of the middle frame I_m in our image sequence, where `m = ceil(M/2)`. What we mean by this is that the resulting panorama image will be composed of frame I_m with all the other frames backwarped so that they properly align with it. To achieve this we will need to translate the set of homographies $H_{i,i+1}$ that transform image coordinates in frame I_i to frame I_{i+1} into a different set of homographies $\bar{H}_{i,m}$ that transform image coordinates in frame I_i to this middle frame I_m.

First we note that given 2 homography matrices $H_{a,b}$ and $H_{b,c}$ and a point p_a in coordinate system a, we can transform the point to coordinate system c by first operating with homography $H_{a,b}$ to obtain an intermediate point p_b in system b and then by operating with homography $H_{b,c}$ to obtain p_c in system c. Each time we operate with a homography H on a point $p = (x, y)$ we do two things: First we multiply the homogeneous column 3-vector $\tilde{p} = (x, y, 1)^t$ from the left by H and then we renormalize to keep the 3rd element equal to 1. When we operated on p_a first with $H_{a,b}$ and then with $H_{b,c}$ we could have equally well multiplied the homogeneous vector $\tilde{p}_a$ once from the left by the matrix $H_{b,c}H_{a,b}$ and then normalized only once at the very end. We see then that multiplying the homography matrices $H_{a,b}$ and $H_{b,c}$ has produced a new homography matrix $H_{a,c} \equiv H_{b,c}H_{a,b}$ that now transforms from coordinate system a directly to c.

We may now use this property of homographies to obtain $\bar{H}_{i,m}$ from $H_{i,i+1}$. We do this as follows

- For $i < m$ we set $\bar{H}_{i,m} = H_{m-1,m} * ... * H_{i+1,i+2} * H_{i,i+1}$
- For $i > m$ we set $\bar{H}_{i,m} = H^{-1}_{m,m+1} * ... * H^{-1}_{i-2,i-1} * H^{-1}_{i-1,i}$
- For $i = m$ we set $\bar{H}_{i,m}$ to the 3×3 indentity matrix $I =$ `eye(3)`

This procedure should be implemented in the following function

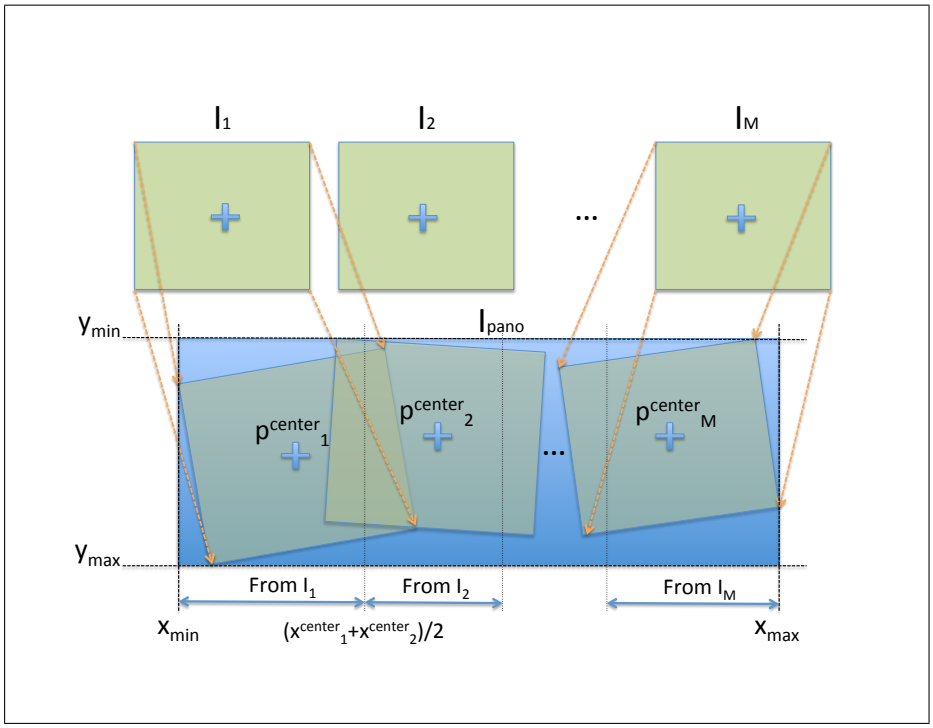
```
function name: accumulateHomographies
% ACCUMULATEHOMOGRAPHY Accumulate homography matrix sequence.
% Arguments:
% Hpair − array or dict of M−1 3x3 homography matrices where Hpair{i} is a
% homography that transforms between coordinate systems i and i+1.
% m − Index of coordinate system we would like to accumulate the
% given homographies towards (see details below).
% Returns:
% Htot − array or dict of M 3x3 homography matrices where Htot{i} transforms
% coordinate system i to the coordinate system having the index m.
```

```
% Note:
% In this exercise homography matrices should always maintain
% the property that H(3,3)==1. This should be done by normalizing them as
% follows before using them to perform transformations H = H/H(3,3).
```

In this function interface the array or dict of homographies `Hpair` corresponds in the above discussion to the set of homographies $H_{i,i+1}$ and the returned `Htot` corresponds to the set $\bar{H}_{i,m}$.

3.2 Rendering the panorama

Now that we have the set of M homographies $\bar{H}_{i,m}$ transforming pixel coordinates in frame I_i to the panorama coordinate system, we now proceed to describe the way in which the panorama frame I_{pano} is rendered. First we will need to define where we want I_{pano} to be rendered. We would like this region to be large enough to include all pixels from all frames I_i. To do so we compute where the 4 corner pixel coordinates (top-left, top-right, bottom-right, bottom-left) of each frame I_i get mapped to by $\bar{H}_{i,m}$, denoting these positions in the panorama coordinate system by $p_i^{corner-k}$ where $k = 1..4$. The coordinates of a rectangle bounding the set of $4 * M$ corners $p_i^{corner-k}$ denoted $x_{max}, x_{min}, y_{max}, y_{min}$, will then define the region in which the panorama image I_{pano} should be rendered. We will now define what parts of I_{pano} should be obtained from which frame I_i. We divide the panorama to M vertical strips each covering a portion of the full lateral range $[x_{min}, x_{max}]$. The boundaries between these strips are taken to be the following $M - 1$ x-coordinates: $(x_i^{center} + x_{i+1}^{center})/2$ for all $i = 1..M - 1$, where we denote by x_i^{center} the x-coordinate of the center of the ith image p_i^{center} in the panorama coordinate system (obtained again using $\bar{H}_{i,m}$). This division of the panorama to vertical strips is shown in the following figure



Now you should (Back)Warp the images on the the strips of the panorama. should then be preformed as follows. Let's assume you have I_{pano} to represent the image intensities and X_{pano}, Y_{pano} represent the x and y coordinates of each of the panorama pixels. Each of them is of dimensions $(y_{max} - y_{min} + 1) \times (x_{max} - x_{min} + 1)$ and the range of the coordinates are $[x_{min}, x_{max}]$ and $[y_{min}, y_{max}]$. Now, for each strip $i = 1..M$ the elements in X_{pano} and Y_{pano} within this strip, denoted $X^{strip-i}$ and $Y^{strip-i}$, should be transformed by $\bar{H}_{i,m}^{-1}$ using `applyHomography` *back* to the coordinate system of frame i. We call these $X'^{strip-i}$ and $Y'^{strip-i}$. These backwarped coordinates can now be used to perform the backwarp operation from frame I_i into strip i of the panorama frame. The function implementing the panorama rendering should have the following interface

```
function name: renderPanorama
% RENDERPANORAMA Renders a set of images into a combined panorama image.
% Arguments:
% im − array or dict of n grayscale images.
% H − array or dict array of n 3x3 homography matrices transforming the ith image
% coordinates to the panorama image coordinates.
% Returns:
% panorama − A grayscale panorama image composed of n vertical strips that
% were backwarped each from the relevant frame im{i} using homography H{i}.
```

4 Tying it all Together

To ease the development burden, the main function, `generatePanorama` responsible for all the housekeeping has been provided. `generatePanorama` essentially goes through the following steps

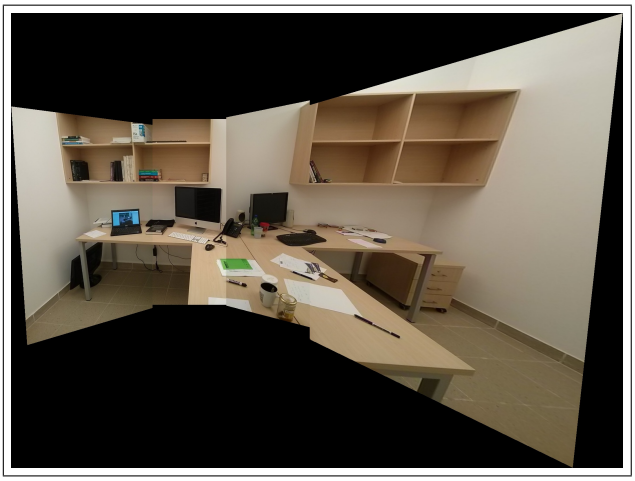
- Read grayscale frames.
- Find and match features.
- Register homography pairs (using your `ransacHomography`).
- Display inlier and outlier matches (using your `displayMatches`).
- Transform the homographies (using your `accumulateHomographies`).
- Load the RGB frames.
- Render panorama of each color channel (using your `renderPanorama`).

5 Your Panoramas

You should submit two panorama image sequences having the same naming convention as the example sequences. Place these image sequences in your github under `/data/inp/mine` directory. Also submit a script called `myPanorama.py` which produces panoramas out of your image sequences. These two resulting panoramas should be saved into the `/data/out/mine` directory.

Good luck and enjoy!

(a) oxford



(b) office

(c) backyard