Homework 4 - Panorama Stitching, Interpolation, K-Means
Computer Vision Fall 2018

Prof. Dr. Francesco Maurelli, Jacobs University Bremen

Handed out 28.11.2018, due 09.12.2018 23:55:00

Please use the moodle system to upload your homework (`moodle.jacobs-university.de`). The system will shut down at the deadline and homeworks will not be accepted at a later time. No other form of submission is allowed.
Please upload one single zip file, with a main pdf file which can guide the instructors through your work and the various Matlab files and images generated. Be as clear as possible.
Use of LaTeXis recommended, though not compulsory. Add your source code and result images when requested and properly reference them in the pdf file.

1 Panorama Stitching

At `http://robotics.jacobs-university.de/TMP/bunker_valentin/drone_panaroma/drone_sequential_images.zip` you will find a set of images taken by a drone (full video at `http://robotics.jacobs-university.de/TMP/bunker_valentin/drone_panaroma/DJI_0009.MOV`). Based on the previous homework, arrange a panorama stitching. The best result will win a (small) prize. You are free to downsample the images (due to high resolution) and/or subsample from the sequence (due to high overlaps). If you have any query about the data, you can contact Arturo Gomez Chavez `<a.gomezchavez@jacobs-university.de>`, who was piloting the drone.

2 Frame Interpolation using Optical Flow

Suppose we are given a pair of frames I_0 and I_1 from a video sequence which occur at times $t = 0$ and $t = 1$, respectively. We wish to generate additional interpolated frames I_t for $0 < t < 1$. This is a common problem in video processing. For example, if we wish to display a video with 24 frames per second on a television with a refresh rate of 60 Hz, then interpolating the frames of the input video can result in smoother playback.

The code and data for this problem are located in the `FrameInterpolationSkeleton` directory. Start by looking at the file `FrameInterpolation.m`; this file sets up the frame interpolation task for you by loading a pair of images and computing the optical flow between them.

NOTE: In the equations below, we use Cartesian coordinates (x, y). However, the MATLAB code for this problem uses matrix coordinates $(row, column)$. Keep this in mind when implementing your solution.

a) The simplest way of generating interpolated frames is to use linear interpolation, also known as cross- fading. In this model, the interpolated images It are given by

$$I_t(x, y) = (1 - t)I_0(x, y) + tI_1(x, y) \tag{1}$$

Implement this method in the file `ComputeCrossFadeFrame.m`. After implementing this method, you can view the interpolated video sequence by uncommenting the indicated lines in `FrameInterpolation.m`. Make sure you include the code and the image generated in your writeup.

b) More sophisticated techniques for frame interpolation take the appearance of the input images into account. One of the most natural ways to incorporate this information is to use optical flow fields. In this problem the horizontal and vertical components of the velocity of the point ${}_It(x, y)$ will be denoted $u_t(x, y)$ and $v_t(x, y)$ respectively. As discussed in class, we can compute the optical flow between the images I_0 and I_1 to obtain the velocities $(u_0(x, y); v_0(x, y))$ of every point of I_0. After computing the optical flow, a simple strategy for computing interpolated frames is to carry the pixels of I_0 forward along their velocity vectors; this algorithm is known as forward warping. More concretely, the interpolated images are given by

$$I_t(x + tu_0(x, y), y + tv_0(x, y)) = I_0(x, y) \tag{2}$$

Implement this method in the file `ComputeForwardWarpingFrame.m`. After implementing this method, you can view the interpolated video by uncommenting the indicated lines in `FrameInterpolation.m`. Make sure you include the code and the image generated in your writeup.

c) Forward warping gives much better results than cross-fading, but it can still lead to significant artifacts in the interpolated images. Give at least one explanation for the types of artifacts that we see when using forward warping to interpolate images.

d) A more robust strategy is to use backward warping. If we knew the velocities $(u_t(x,y), v_t(x,y))$ for all of the points in I_t, then we could compute the pixel values of I_t by setting

$$I_t(x,y) = I_0(x - tu_t(x,y), y - tv_t(x,y)) \tag{3}$$

Unfortunately we do not know the velocities at time t. However, we can estimate the velocities at time t by forward-warping the known velocities u_0 and v_0 at time 0. We can increase the robustness of this estimate by using a few heuristics.

For each point (x,y) of I_0, we compute the quantities $x' = x + tu_0(x,y)$ and $y' = y + tv_0(x,y)$. Then for all pairs (x'', y'') with $x'' \in \{floor(x'), ceil(x')\}$ and $y'' \in \{floor(y'), ceil(y')\}$, we set

$$\begin{aligned} u_t(x'', y'') &= u_0(x,y) \\ v_t(x'', y'') &= v_0(x,y) \end{aligned} \tag{4}$$

This can help account for small inaccuracies in the computed optical flow. If this would cause multiple points (x,y) of (u_0, v_0) to be assigned to the point (x'', y'') of (u_t, v_t), then pick the one that minimizes the color difference

$$\left| I_0(x,y) - I_1(x + u_0(x,y), y + v_0(x,y)) \right| \tag{5}$$

The quantity $I_1(x+u_0(x,y), y+v_0(x,y))$ is appearance of the pixel to which (x,y) would be sent if we forward warped all the way to time 1. The idea behind this heuristic is that if two or more pixels at time 0 collide when forward warped to time t, then we keep only the pixel whose appearance changes the least when forward warped all the way to time 1; hopefully this is the pixel that also changes the least when forward warped to time t. Additionally, if any points of the interpolated flow (u_t, v_t) remain unassigned after this procedure, we use linear interpolation to fill in the gaps. The function `FillInHoles` can be used for this step. Implement forward warping of the flow field as described above in the function `WarpFlow` of the file `ComputeFlowWarpFrame.m`.

e) Once we have estimated the velocities (u_t, v_t), we can use backward warping to compute the interpolated image I_t as described above. Implement backward warping in the function `ComputeFlowWarpFrame` of the file `ComputeFlowWarpFrame.m`. After implementing this method, you can view the interpolated video by uncommenting the indicated lines in `FrameInterpolation.m`. Make sure you include the code and the image generated in your writeup.

3 K-Means - local minima

Consider the points:

$$x_1 = (0,16); x_2 = (0,9); x_3 = (-4,0); x_4 = (4,0)$$

Suppose we wish to separate these points into two clusters and we initialize the K-Means algorithm by randomly assigning points to clusters. This random initialization assigns x_1 to cluster 1 and all other points to cluster 2.

a) With these points and these initial assignments of points to clusters, what will be the final assignment of points to clusters computed by K-Means? Explicitly write out the steps that the K-Means algorithm will take to find the final cluster assignments.

b) In this case, does K-Means find the assignment of points to clusters that minimizes the quantity in Equation 6?

$$\sum_{i=1}^{m} d(x^{(i)}, \mu_{\ell_i}), \tag{6}$$

where m is the number of points $x^{(1)}, \ldots, x^{(m)}$, k is the desired number of clusters, ℓ_i is the label corresponding to the cluster i, such as $\ell_i \in \{1, \ldots, k\}$, and $d(x,y)$ is the distance between points x and y. In case it does not, can you propose a different initialisation that will lead to a correct clusterisation?