COMPUTER VISION - LAB 6

Topics: Object recognition and tracking

Goal: Detect and track a set of objects in a video

Notes: Group project

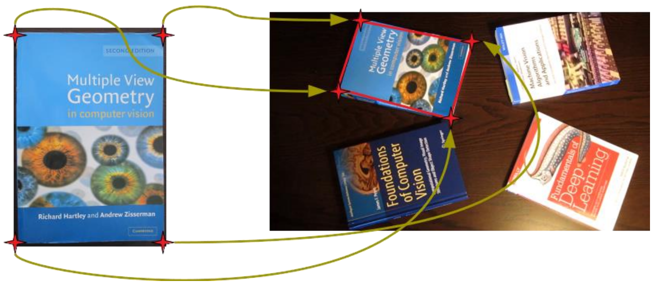
Write a C++ application, appropriately structured in classes and/or functions, that:

1. Read each frame from an input video or a camera, e.g. by using the Opencv `cv::VideoCapture` object:

   ```
   cv::VideoCapture cap("video.avi”);
   if(cap.isOpened())  // check if we succeeded
   {
     for(;;)
     {
       cv::Mat frame;
       cap >> frame;

      // Do something with frame ..
     }
   }
   ```

2. Locate into the first image of the video a set of objects of interest, represented by an example image (one example for each object). To locate objects, match the local features of the example images with the input image. Use for instance the SIFT features, and the `cv::BFMatcher` class as feature matcher. Visualize the keypoints that belong to different objects with different colors.
   Try as possible to discard outliers (i.e., wrong matches) by using the RANSAC algorithm provided inside the `findHomography()` function , as done in Laboratory 5.
   Starting from here, stop using any feature extraction and matching procedure! Now we are going only to *track* features.
3. Draw a rectangle (see the OpenCV `cv::line()` function) around each located object. This could be done by exploiting the planar homography provided by the `findHomography()` function used to reject outliers in the previous point. A planar homography H is a linear relationship between the coordinates of points on a plane in the scene and those of that point in the image. In other words, given the (homogeneous) coordinates p of a point in a plane (e.g., one of the example images of the object), it is possible to recover the position q (in homogeneous coordinates) of such point into the image plane simply by multiplying it by H, i.e. $q = Hp$.



4. Track (and show) the features of each object using the pyramid Lukas-Kanade tracker implemented in OpenCV (check the `cv::calcOpticalFlowPyrLK()` functions). **Try to preserve more features as possible, while discarding outliers**.
5. Update at each new image the rectangles around each located object, e.g., by estimating the translation and rotation of the objects by using the optical flow of the related points.

A sample video and a set of four example objects are provided, but of course you can build your own dataset with your objects, e.g. by using your smartphone. Just take into account that, since we are using planar homographies to compute example-image correspondences, planar objects are required.
Note: Three components' groups shall also provide a new dataset with at least three different objects.