Technische Universität München
Fakultät für Informatik
Dr. Slobodan Ilic

WS 12/13
Exercise Sheet 8
January 10, 2013

Exercises in Tracking & Detection

Exercise 1 Learning the Hyperplane Template Matching Tracker

In this exercise we want you to get to know a bit more about the very wide and important topic of Tracking. Tracking supposes that the inter-frame motion from image taken at time $t-1$ to an image taken at time t is small. Therefore, the pose of the object in image $t-1$ is able to serve as an initialization for the Tracking algorithm in image t. Very famous Trackers are the Lucas-Kanade Tracker, the ESM Tracker, the Inverse Compositional Tracker and the Hyperplane Template Matching Tracker (*Hyperplane Approximation for Template Matching*, Jurie and Dhome, PAMI, 2002). The difference of the latter one to the rest is that it's Jacobian is not analytically computed but learned from a set of training samples.

For this exercise we try to track a planar object in a sequence of images which is uploaded on the website (take the image sequence of exercise 6).Due to the planarity, the pose can be represented by a simple homography. In the following we define a rectangular part in the first image as the object to track. In order to parametrize the homography we use the corners of the quadrangle (for each corner we have one x and one y value; so finally we have 8 parameters). For tracking we do not use all points within this rectangular region but only some sample points defined on a regular grid within the rectangle. A good distance between the grid points is e.g. 5 pixels.
As you have seen in the lecture the approach needs the following formula:

$$\delta\mathbf{p}(t) = \mathbf{A}\delta\mathbf{i}(t) \tag{1}$$

where $\delta\mathbf{p}(t)$ is the parameter update vector for the pose parameters at time t, $\mathbf{A}$ is the Hyperplane matrix (or update matrix) and $\delta\mathbf{i}(t)$ is the current intensity difference at time t on the template.

a) write a Matlab function that takes the input image and the coordinates of the rectangle and randomly warpes the rectangluar region with small transformations. This is done by adding random values (in a certain range) to the x and y coordinate of the rectangle, by computing the transformation (use DLT and homographies) between the rectangular region and the new (random) quadrangle region and by warping the image with this transformation. Use back-warping (=> each point in your destination patch is back-warped to the original image with the inverse transformation. The location the back-warped point hits determines the intensity of the point in the warped patch)! Then normalize the intensity values such that the mean is zero and the standard deviation is one. Finally, add some random noise to the normalized intensity values.

b) generate n ($n \geq NumOfGridPoints$) random warped samples of the rectangle (as mentioned above), compute the differences between the original normalized intensity values and the warped normalized intensity values and save the normalized intensity differences column-wise in a matrix $\mathbf{I}$. At the same time save the corresponding corner displacements column-wise in a matrix $\mathbf{P}$ (so the difference in the x and y coordinate for each corner between the original rectangle and the new quadrangle).

c) compute $\mathbf{A} = \mathbf{P}\mathbf{I}^\top(\mathbf{I}\mathbf{I}^\top)^{-1}$ where $\mathbf{I}$ is the matrix with the different intensity differences and $\mathbf{P}$ the corresponding matrix holding the displacement vectors.

In order to get better results, compute a series of matrices $\mathbf{A}_n$ instead of only one matrix $\mathbf{A}$ where the displacements of the corners of the rectangle are successively decreased (for instance, for $\mathbf{A}_n$ allow large x and y coordinate displacements whereas in $\mathbf{A}_0$ you only allow very small coordinate displacements). This is necessary in order to obtain a trade-off between convergence region and exactness (for more explanation read the paper of Jurie and Dhome).
Tip: A good size for your template is 100x100. A good number of update matrices is 10. The largest parameter update should be in the range of $[-30; 30]$. For the smallest update the range $[-3; 3]$ should do fine. Depending on the template size - these numbers may vary! Try out different ranges to see what works best!

<u>Exercise 2</u> Tracking with the Hyperplane Template Matching Tracker

Having learned the update matrices $\mathbf{A}_{i=0..n}$, the parameter vector (the x- and y-components of the template corners) obtained from the last frame is updated iteratively:

a) Extract the image values at the sample positions warped according to the current parameter vector.

b) Normalize the extracted image values as done in the learning stage.

c) Substract the normalized image values of the reference template from the current normalized image values.

d) Compute the parameter update by multiplying the update matrix with the obtained image value difference vector $\delta\mathbf{i}$:
$$\delta\mathbf{p} = \mathbf{A}_n\delta\mathbf{i}. \tag{2}$$

e) Update the parameter vector using the parameter update:

- Compute the current homography $\mathbf{H}_c$ between the initial parameter vector and the current parameter vector.
- Compute the update homography $\mathbf{H}_u$ which corresponds to the update only.
- Multiply the current homography by the update homography:
$$\mathbf{H}_n = \mathbf{H}_c\mathbf{H}_u 4. \tag{3}$$
- Compute the new parameter vector based on the new homography.

These steps are iterated starting with the update matrix computed for large motions. Each update matrix should be applied multiple times (5 times per matrix should be enough).