16720 Computer Vision: Homework 3
Template Tracking and Action Classification.

Instructor: Martial Hebert
TAs: David Fouhey, Heather Knight and Daniel Maturana
Due Date: October 17^{th}, 2012.

1 Instructions

- A complete homework submission for this homework consists of three parts. A `pdf` file with answers for the conceptual questions and the writeup and three directories of Matlab files for the implementation portions of the homework, which should be named `q1`, `q2` and `q3`.

  The three directories and the `pdf` should be placed in the main submission directory that is named after your andrew id.

- All questions marked with a **Q** require a submission. For the implementation part of the homework please stick to the function headers described.

- Final reminder: **Start early!**

2 Lucas-Kanade Tracking

In this section you will implement a Lucas-Kanade template tracker. As seen in class, the tracker estimates the displacement between two image frames under the assumption that the intensity of the objects has not changed significantly.

A thorough reference for the Lucas-Kanade tracker is given below.

> Lucas-Kanade 20 Years On - Baker, S. and Matthews, I. http://www.ri.cmu.edu/publication_view.html?pub_id=4031

Starting with a rectangle R_t on frame I_t, the Lucas-Kanade Tracker aims to displace it by an offset (u, v) to obtain another rectangle R_{t+1} on frame I_{t+1}, so that the pixel squared difference in the two rectangles is minimized.

Q1. (5 points) Starting with an initial guess of (u, v), we can compute the optimal $(u^*,\ v^*)$ iteratively. In each iteration, the objective function is locally linearized and optimized by solving a linear system has the form $A\ \Delta p = b$, where $\Delta p = (u, v)^T$ is the template offset.

- What is $A^T A$?

- What conditions must A^TA meet so that the template offset can be calculated reliably?

Q2. (10 points) Implement a function:

```
[u,v] = LucasKanade(It,It1,rect)
```

that computes the optimal local motion from frame I_t to frame I_{t+1} minimizing Eqn. 1. Here `It` is the image frame I_t , `It1` is the image frame I_{t+1} , and rect is the 4×1 vector representing a rectangle on the image frame I_t . The four components of the rectangle are $[x1, y1, x2, y2]$, where $(x1, y1)$ is the top-left corner and $(x2, y2)$ is the bottom-right corner. The rectangle is inclusive, i.e., it includes all the four corners. To deal with fractional movement of the template, you will need to interpolate the image using the Matlab command `interp2`. You will also need to iterate the estimation until the change in (u, v) is below a threshold (which you set empirically).

Q3. (10 pts) Write a script `testCarSequence.m` that loads the video sequence (`carSequence.mat`) and runs the Lucas-Kanade Tracker to track the speeding car. The rectangle in the first frame is $[x1, y1, x2, y2] = [318, 202, 418, 274]$. In your answer sheet, print the tracking result (image + rectangle) at frames 20, frame 60 and frame 80, and include images of the sequence with a rectangle around the template for each of these frames.

Make a `.mat` file which has top-left and bottom-right corners of your tracked object for us to evaluate your tracking result. The `.mat` file should contain a variable called `TrackedObject` which is a $n \times 4$ matrix; n is number of frames in `CarSequence` and each row in the matrix contains 4 numbers: `x1 y1 x2 y2`, where indicates the coordinates of top-left and bottom-right corners to the object you tracked. Name the `.mat` file as `trackCoordinates.mat`.

Q4. (5 points) What are the possible reasons the tracker could fail? How could you possibly fix some of these problems?

3 Tracking with Dominant Motion Estimation

In this section, you will implement a tracker for estimating dominant affine motion in a sequence of images and subsequently identify pixels corresponding to moving objects in the scene. You will be using the images in the file `Sequence1.tar.gz` consisting of aerial views of moving vehicles from a non-stationary camera.

3.1 Dominant Motion Estimation

You will start by implementing a tracker for affine motion using the equations for affine flow described in class and in the Motion slides from class.

To estimate dominant motion, the entire frame at time t, I_t, will serve as the "template" to be tracked in image I_{t+1}, i.e. I_{t+1} is assumed to be approximately an affine warped version of I_t.

Using the equations for the affine model of flow, you can recover the 6-vector $\Delta p = \begin{bmatrix} a & b & c & d & e & f \end{bmatrix}^T$ of affine flow parameters. They will be related to the equivalent affine transformation matrix as:

$$M = \begin{bmatrix} 1+a & b & c \\ d & 1+e & f \\ 0 & 0 & 1 \end{bmatrix} \tag{1}$$

relating the homogenous image coordinates of I_t to I_{t+1} according to $\mathbf{x}_{t+1} = M\mathbf{x}_t$ (where $\mathbf{x} = \begin{bmatrix} x & y & 1 \end{bmatrix}^T$). For the next pair of consecutive images in the sequence, image I_{t+1} will serve as the template to be tracked in image I_{t+2}, and so on through the rest of the sequence. Note that M will differ between successive image pairs. As before, each update of the affine parameter vector, Δp is computed via a least squares method using the pseudo-inverse as described in class.

Unlike previous examples where the template to be tracked is usually small in comparison with the size of the image, image I_t will almost always not be contained fully in the warped version of I_{t+1}. Hence the matrix of image derivatives, A, and the temporal derivatives, I_t , must be computed only on the pixels lying in the region common to I_t and the warped version of I_{t+1} to be meaningful.

Q5 (20 pts) write a function

```
function [M] = LucasKanadeAffine(It, It1)
```

Where `M` is the affine transformation matrix, and `It` and `It1` are I_t and I_{t+1} respectively.

`LucasKanadeAffine` should be relatively similar to `LucasKanade` from last question.

3.2 Moving Object Detection

Once you are able to compute the transformation matrix M relating an image pair I_t and I_{t+1}, a naive way of determining pixels lying on moving objects is as follows. Warp the image I_t using M so that it is registered to I_{t+1}, and subtract it from I_{t+1}. The locations where the absolute difference exceeds a threshold can then be declared as corresponding to locations of moving objects. For better results, hysteresis thresholding may be used. We have provided Matlab code for hysteresis thresholding (`hysthresh`).

Q6 (10 Points) Using the above method for dominant motion estimation, write a Matlab function

```
[moving_image] = subtractDominantMotion(image1, image2)
```

where image1 and image2 (in uint8 format) form the input image pair and `moving_image` is a binary image of the same size as the input images, in which the non-zero pixels correspond to locations of moving objects. The script `test_motion.m` that we will use for grading this section has been provided to you for testing your function. This script simply makes repeated calls to `subtractDominantMotion` on every consecutive image pair in the file `Sequence1.tar.gz` , makes an AVI movie out of the `moving_image` returned for every image pair processed, and saves it as a file *motion.avi* for your offline viewing

pleasure. An example of such a movie is also provided consisting of results using the methods described above.

You will notice that the estimates of moving pixels are a bit noisy. [1] We do not expect perfect results – this is, after all, real world data! You are free to implement any algorithm of your choice for estimating the `moving_image` following the affine registration step[2]. We will be awarding **QX1 Extra Credit (5 Points)** to the best looking results as judged by the instructors.

4 Action classification with Space-Time Interest Points and Optical Flow Histograms

4.1 Overview

In this section you will implement a simplified pipeline for action classification, based on the following papers:

> *On Space-Time Interest Points* (2005), I. Laptev; in International Journal of Computer Vision, vol 64, number 2/3, pp.107-123
>
> *Local Descriptors for Spatio-Temporal Recognition* (2004), I. Laptev and T. Lindeberg; in ECCV Workshop "Spatial Coherence for Visual Motion Analysis"

Both can be downloaded from `http://www.di.ens.fr/~laptev/actionrecognition.html`.

You will evaluate the system on the popular KTH Action dataset. The original dataset may be found at `http://www.nada.kth.se/cvap/actions/`, but Matlab has trouble reading the videos directly. We have decoded the videos into images for you. You can download them from `http://ladoga.graphics.cs.cmu.edu/dmaturan/hn45/pub/nada_kth`.

As in the previous homework, the system is based on the bag of words technique. However, in this case the features to be clustered into words are small descriptors of "movement" computed with optical flow between consecutive video frames.

Unlike the previous homework, the descriptors will not be extracted at random points. The KTH dataset videos have a relatively static background that is generally not informative for action classification. Moreover, optical flow on many parts of the image is undefined or unreliable. Therefore we will compute these movement descriptors in selected keypoints known as Spatio-Temporal Interest Points. The keypoints will be computed as a straightforward 3D generalization of the Harris keypoint seen in class, where the third dimension is time.

[1] For de-noising your output, you may also want to look at the imerode and imdilate morphological operators in Matlab.

[2] A basic version of this would involve computing the difference between the warped version of the first image and the second image and thresholding.

Once found, the optical flow around these keypoints is summarized with a histogram of the optical flow over a small area around the keypoint. We will call these histograms Optical Flow Histograms or OFH [3].

To summarize all the OFH in a video sequence we will again use the "bag of visual words" technique. First a "dictionary" of OFH will be constructed using k-means on a sample of OFH descriptors. Then each OFH in the video sequence is quantized by matching it to a visual word, and the entire video is summarized as a histogram over these words [4].

Finally, classification is performed by matching the word histogram of each video in the testing set to its nearest neighbor in the training set.

In the following, we will walk you through the implementation.

4.2 Loading the data

To make the homework simpler, we will only use four of the original six action categories in the dataset. This leaves 1083 sequences, with each sequence depicting a person performing an action (boxing, handwaving, etc).

We have provided a structure containing metadata for the dataset, saved in `nada_kth_seq.mat`. The structure has the following members:

`partition` whether each sequence belongs to each sequence belongs to the training (0), validation (1) or test (2) set.

`action` the label of each sequence. The mapping to action names is in `action_ix`.

`fnames` cell array of image filenames corresponding to each sequence.

`flow_fnames` name of filename storing flow fields for each sequence.

To load a sequence, we have provided the function `loadImageSequence`. For example, you can load the first image sequence like so:

```
load nada_kth_seq;
seq = nada_kth_seq.fnames{1};
is = loadImageSequence(seq);
```

This will return the images in `seq` as a three-dimensional array of size (w, h, t), i.e. width, height and number of frames respectively.

4.3 Harris 3D Space-Time Interest Points

The first step is to compute Harris 3D interest points. We have provided a reference implementation of Harris 3D with the following signature[5]

[3] Not to be confused with the slightly more complicated Histograms of Optical Flow (HOF) described elsewhere in the literature.

[4] Note there are two kinds of histograms in this system: one summarizes optical flow around each keypoint, whereas the other summarizes visual words.

[5] This code uses some functions from Piotr Dollar's Matlab toolbox, found in `http://vision.ucsd.edu/~pdollar/toolbox/doc/`, which we have included with the homework. You are encouraged to

```
function points = computeHarris3D(images, sigma, tau, maxpoints)
```

where `images` is an image sequence as loaded above and `points` is an $N \times 3$ array containing coordinates (i, j, t) such that `images(i,j,t)` is an interest point and N is the number of points, a data-dependent number. `sigma` and `tau` are scale parameters. You are encouraged to read the papers mentioned above for the background on Harris 3D and what the scale parameters do. In short, the algorithm computes an analog of the Harris "corners" seen in class, but in three-dimensions instead of two.

The scales are related to the scale of the space-time "corners" you expect to detect; in practice, they are tuned empirically and are quite important for the performance of the algorithm. Moreover, the above papers use multiple scales. Our simplified pipeline will use just a single scale. Using $\sigma^2 = \tau^2 = 1$ gives acceptable results; feel free to experiment with these values.

`maxpoints` is a parameter that limits the maximum number of keypoints returned. This is related to the expect points you expect in the data and how you much you can handle computationally. 100 to 500 is a reasonable value.

Note: Not all the points you get will seem "interesting". Many of them may be noise, specially at the boundaries (both in space and time) of the images. For an idea of the keypoints you should get look at fig. 1, which depicts spatiotemporal interest points and optical flow obtained with this simplified implementation.

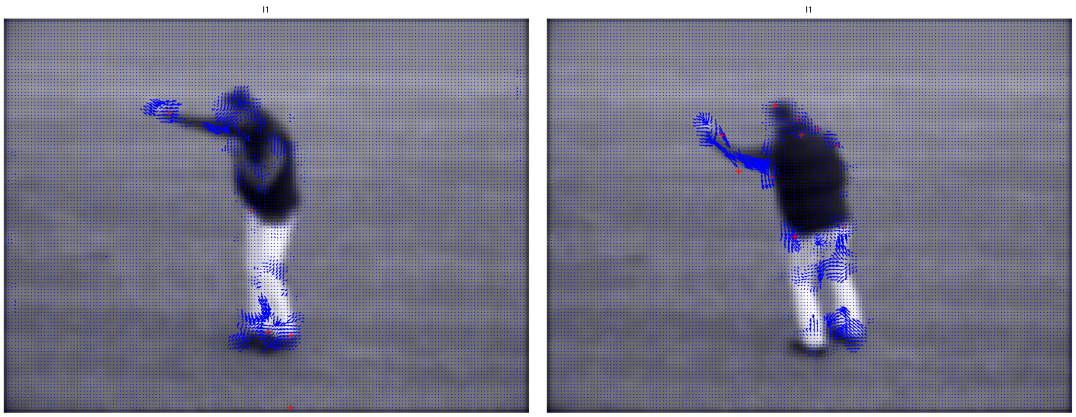

Figure 1: Interest points (red crosses) and optical flow (blue arrows) for two frames of a boxing sequence.

4.4 Computing OFH descriptors

Q7 (15 points)

The OFH descriptor consists of a histogram of the optical flow vectors computed in an area around each keypoint. It is a simple version of the "Local position independent

browse the toolbox's code as an example of well-written, efficient Matlab code, but don't use it for the assignment unless explicitly allowed.

histogram of optic flow" descriptor described in the second paper cited above.

You should implement a function with the header

```
function descriptors = computeDescriptors(Vx, Vy, points)
```

where `Vx` and `Vy` describe an optical flow field, and `points` are the interest points obtained from `computeHarris3D`. `descriptors` is an $N \times 64$ array where N is the number of interest points (rows) in `points`[6] and 64 is the dimensionality of the descriptor.

To save you time we have provided the pixelwise optical flow computed for each frame, downloadable from the same place as the images. As mentioned earlier, the filename for the flow of the ith sequence is stored in `nada_kth_seq.flow_fnames{i}`. The file is a `.mat` file that can be read like so:

```
of = load(nada_kth_seq.flow_fnames{i});
```

Then the velocity field (v_x, v_y) will be stored in the two 3D arrays `of.Vx` and `of.Vy`, which have the same size as the corresponding image sequence. In other words, the velocity at row i, column j and frame t will be `[of.Vx(i,j,t), of.Vy(i,j,t)]`. Note that the parameters for optical flow computation were chosen so the fields are relatively sparse.

To compute the descriptor for a keypoint with coordinates `[i, j, t]`, compute a 32-bin histogram of `Vx` accumulated over a 3D-window with size $(17, 17, 11)$ centered on `[i, j, t]`. Normalize the histogram to sum to 1. Then do the same for `Vy` and concatenate the two, resulting in a 64-bin vector.[7]

You can use the Matlab function `hist`. The papers don't say anything about the histogram edges, so we've heuristically chosen 32 centers equally spaced in the interval $(-4, 4)$. Feel free to experiment with this.

4.5 Creating the dictionary

Q8 (5 points)

As in the last homework, we will create a dictionary with k-means. You can use Matlab's implementation:

```
[ignore, dictionary] = kmeans(trainDescriptors, K, 'EmptyAction', 'drop')
```

where `trainDescriptors` is an array containing the descriptors of all the videos in the training dataset and `K` is the number of visual words. This will probably be the slowest part of the pipeline. With around 130000 input descriptors and $K = 400$, this took about 20 minutes on my computer.

4.6 Quantizing into Bags of Words

Q9 (5 points)

Now write a function with the signature

[6] The number of rows in `descriptors` may actually be less than the rows in `points` if some of the points are discarded for being too close to the edges.

[7] You may discard points with windows that fall outside the boundaries of the input sequence.

```
function qdesc = quantizeDescriptors(descriptors, dictionary)
```

which will assign each OFH descriptor in `descriptors` (of size $M \times 64$, where M is the number of descriptors) to one of the keywords in `dictionary` (of size $K \times 64$) and return a normalized bag of words histogram `qdesc`, of size $1 \times K$. Feel free to reuse code from your previous homework for this. You can use the functions `pdist2` and `histc`.

4.7 Classifying

Q10 (5 points)
Write a function with the signature

```
function predLabels = classifyBoW(trainBow, trainLabels, testBow)
```

Which uses a nearest neighbor classifier to predict a label for each bag of words histogram in `testBow`. You can use any of the distance or similarity metrics you used for the last assignment.

4.8 Putting it all together

Q11 (5 points)
Write a script `evaluateRecognitionSystem.m` that performs the following steps:

1. Compute Harris 3D points for each sequence in the training set.
2. Extract OFH descriptors for each point.
3. Cluster the descriptors to create a dictionary.
4. Compute bag of words histograms for each sequence in the training set.
5. Compute Harris 3D points, OFH descriptors and corresponding bag of words histograms for each sequence in the testing set.
6. Classify each sequence in the testing set.
7. Compute a confusion matrix and net accuracy for your predictions.

This script should also save the dictionary as a variable named `dictionary` and a confusion matrix as a variable named `confusionMatrix` into a file named `vision.mat`.

Remember, all the information you need is in `nada_kth_seq.mat`. The whole pipeline should take around 30 minutes, depending on your parameters and computing power. We suggest subsampling the data when first implementing and testing the algorithm.

As for the accuracy, with the suggested parameters our implementation obtains 75%. Naturally, yours may vary, but it should not be significantly inferior.

Q12 (5 points) Writeup Finally, put together a small writeup (one page is enough) describing what parameters you chose, a couple of frames with detected Harris 3D points, and the confusion matrix with the results.

4.9 Extra credit

QX2 (10 Points)

You may have noted that even this simplified pipeline has several parameters. The values we suggested were determined by hand and are probably suboptimal. Evaluate the sensitivity of the algorithm (in terms of accuracy) to at least 3 of the parameters. Include plots or tables and briefly describe why you think the parameter changes affect the results they way they do.

4.10 Further reading

While the papers cited above are seminal work in action recognition, there has been considerable progress in the area. There has also been a move towards more complex, realistic datasets. A somewhat more recent paper that compares various local detectors and descriptors for action recognition on various datasets is

> *Evaluation of local features for action recognition* (2009), H. Wang, M. Muneeb Ullah, A. Klaser, I. Laptev and C. Schmid, BMVC 2009. `http://lear.inrialpes.fr/people/klaeser/research_descriptor_evaluation`

Another interesting recent approach uses densely sampled trajectories (as opposed to keypoints) extracted with a state-of-the-art optical flow algorithm for action features:

> *Action recognition by dense trajectories* (2011), H. Wang, Klaser, A., Schmid, C., Cheng-Lin Liu, CVPR 2011. `http://hal.inria.fr/inria-00583818/PDF/wang_cvpr11.pdf`