CSE527 Homework 2

Due. Thu Oct 9, 2014

In this homework you will experiment with SIFT features for scene matching and object recognition. You will work with SIFT tutorial and code from University of Toronto. In the compressed homework file (hw2_files.tgz), you will find the tutorial document (tutSIFT04.pdf), Matlab codes ("matlab" folder), and a paper from the International Journal of Computer Vision (ijcv04.pdf) describing SIFT and object recognition. You are STRONGLY encouraged to read this paper unless you're already quite familiar with matching and recognition using SIFT.

In Matlab, navigate to the "matlab" folder which contains all Matlab functions you can use during the tutorial. Run startup.m by typing "startup": the file paths will be set and you will be able to use the functions. (All needed functions are in the \utvisToolbox\tutorials\SIFTtutorial subdirectory, but do not navigate to this folder since the paths will be set automatically when you run startup.m.) Each function code contains descriptions about its input and output arguments, so take a look at them to figure out how they work.

Follow the steps below to compute your results. Read each step carefully to understand the required input and output for each step.

Problem 1: Match transformed images using SIFT features

{30 points} You will transform an image, and match it back to the original image using SIFT keypoints.

Step 1. Choose an image you want to use for problem 1. Read it into Matlab and, if it is color, convert it to grayscale (you can use rgb2gray() to convert RGB to grayscale). Then use the **sift()** function to extract keypoints from the image. Set the sift function's "interactive" argument to 2, and submit the figures that are displayed on screen.

NOTE: To test if your results are valid, you can use the "einstein.bmp" image in the folder and compare the outputs to the tutorial document. For actual submission, use your own image.

Step 2. You need to store your "reference" keypoints in a database in order to do the matching. Use the **add_descriptors_to_database()** function to do this. Note that you don't have to create any structure for the database yourself; you can just declare a variable and assign to it the output of the function. (You won't need to use the "handle" output argument.)

Step 3. Rotate your image by a certain degree with **imWarpAffine()** function. Extract SIFT keypoints for this rotated image and submit the last figure (rotated picture with keypoints scale and orientation overlaid) that is displayed on screen.

Step 4. Perform Hough transform to compute similarity transform between the rotated image and the original image. Use the **hough()** function to match the SIFT results from the rotated image (the output of step 3) to those of the original image (stored in the database in step 2). The hough() function will return a set of constraints with their corresponding SIFT descriptors (which means that each output argument of hough() will have the same number of rows (or cells) as the number of constraints).

Step 5. From the output arguments of hough(), choose the group of SIFT indices that correspond to the highest weight transformation. (Look at the WGHT output argument.) Then, for all SIFT descriptors

corresponding to these indices, retrieve the positions of original SIFT descriptors, the positions of rotated SIFT descriptors, and the scales of the original SIFT descriptors.

NOTE: SIFT indices will be returned in cells. To access a cell element in Matlab, use curly brackets. For example, idx{2} will give you the second cell element of idx.

Step 6. Compute the affine transformation from the rotated image to the original image. Use **fit_robust_affine_transform()**, with the values retrieved from step 5 as input arguments. This will give you the affine transformation matrix calculated from the SIFT features. Compare this matrix with the actual transformation matrix you used in step 3. Do they look similar? Turn in the values for both matrices.

Problem 2: Scene matching with SIFT features

{30 points} You will match and align between different views of a scene with SIFT features.

Step 1. Choose two pictures of the same scene but taken from different perspective. See p21 of the tutorial document for an example. You could find appropriate images on the web or could take photos by yourself. Crop them so that they match in size, and convert them to grayscale.

Step 2. Extract SIFT features for both images and go through the same procedures as you did in problem 1. Your goal is to find the affine transformation between the two images. Align one of your image to the other by using the **imWarpAffine()** function, and turn in the transformation matrix and the images that show the comparison between the two. (Again, look at p21 of the tutorial document. You should submit something like the images in that page.)

Problem 3: Object Recognition with SIFT features

{40 points} You will recognize objects by matching their SIFT features.

Step 1. Find several pictures of two different objects. For example, you could have 5 pictures of one object and another 5 pictures of the other object. In the tutorial, the objects are a phone and a Java book. Different object position, background clutter, partial occlusion or different lighting conditions will be okay. (I actually encourage you to include such images so that you can see the SIFT features at work.) If you have difficulties collecting images or taking your own photos, you can use the tutorial images in \utvisToolbox\tutorials\SIFTtutorial\images folder. They are in pgm format, which you can read into Matlab with **pgmRead()** function.

Step 2. For each of the two objects, choose one picture with a good view of your object and no occlusion. This will be your "model" for SIFT object recognition. Add the SIFT features from these two images to your database (see problem 1, step 2). To add SIFT features from multiple images to the database, you can just run **add_descriptors_to_database()** multiple times with the same database variable.

Step 3. For each of all other images, find the best matching object and the affine transformation matrix. (Note that if there are multiple images in the database, hough() function will return image indices as well.) Hand in the matrices and the images of matched results (something like p22 & p23 of the tutorial document).

Problem 4(extra credit): Object Recognition with HOG features

{30 points} You will use histogram of oriented gradients (HOG) to extract features from objects and recognize them.

This is a similar task to Problem 3, except that you'll be using histogram of oriented gradients (HOG) instead of SIFT. HOG decomposes an image into multiple cells, computes the direction of gradients for all pixels in each cell, and creates a histogram of gradient orientation for that cell. Object recognition with HOG is usually done by extracting HOG features from a training set of images, learning a support vector machine (SVM) from those features, and then testing a new image with the SVM to determine if there are objects.

You can use the VLFeat library (http://www.vlfeat.org/index.html) to use HOG and SVM (and a lot of other algorithms) in Matlab. Download the binary package and follow the install instructions for MATLAB. Also take a look at the HOG tutorial page (http://www.vlfeat.org/overview/hog.html).

Step 1. Find several pictures of two different objects. You can take the same images you used in problem 3. However, if you used only a small number of images for each object, try to add more. (I would recommend at least 10 images for each object.)

Step 2. Divide your image set into training and testing sets. For example, if you have 20 images (10 for each object), use 19 for training set and pick only one image for the testing set. (We will do a cross validation afterwards by changing the image in the test set.)

Step 3. Crop the training images around the actual object that you want to detect. Keep in mind that **the cropped image size should be the same across all training images, regardless of what kind of objects are in those images**. (Which means that we'll not consider multiple object scales at this point.) Try to find a cropping size that nicely wraps around most objects. You don't need to crop the test image.

Step 4. Extract HOG features from the cropped training images. You can use the VLFeat function **vl_hog()**. Submit what your HOG features look like on your images. Use **vl_hog()** to visualize the features. (The HOG tutorial page above mentions how to do this.) Be advised that vl_hog() function can only take images in SINGLE precision class. To convert your image to single precision class in MATLAB, use the following function:

```
Im_new = im2single( Im );
```

Step 5. The extracted HOG features will be in the form of three-dimensional matrices. Reshape them so that the HOG features from each image form a single column vector. Use the reshape() function in MATLAB to do this. For example, if your HOG feature for each image is a 60 * 80 * 31 matrix, reshape it to a 148800 * 1 vector. Then concatenate the vectors from all training images into a single matrix. If you had 19 images in the training set, your final matrix size would be 148800 * 19.

Step 6. Train an SVM on your features with **vl_svmtrain()** function. You need to pass the matrix you calculated in step 5, as well as a vector of labels that indicate which columns in that matrix correspond to which object. For example, if your matrix has 10 columns where the first 5 columns represent pictures of a book and the last 5 columns represent pictures of a phone, your vector of labels would be something like [1 1 1 1 1 -1 -1 -1 -1 -1]. For more details on how to use vl_svmtrain(), please look at the SVM tutorial page (http://www.vlfeat.org/overview/svm.html) and the function documentation page (http://www.vlfeat.org/matlab/vl_svmtrain.html).

Step 7. Extract HOG features from the test image. Implement a simple sliding-window algorithm where you calculate HOG features for different locations in the image. The size of the sliding window should be the same as that of the cropped training images. For each sliding window location, calculate the

classification score for the test image. You can follow the example codes at the end of the vl_svmtrain() documentation page mentioned in step 6. This score indicates the classification result for your test image. For example, if you labeled the pictures of a book as 1 and the pictures of a phone as -1 during the training phase, the scores would be close to 1 if SVM classified the test image as a book and -1 if SVM classified it as a phone. (They would be real values between -1 and 1.) Try to find the sliding window location that gives you the most confident scores. That location will be where your detected object is.

Step 8. Now do a leave-one-out cross validation on the images. Repeat steps 2 to 7, but pick a different test image for each iteration. Remember that you should not include this test image in the training set. In the end, you should have a classification score and the sliding window location of the detected object for each and every image. Submit all the scores, as well as figures showing the location of the detected objects in the images.

Submission

Submission will be via Blackboard. You will submit the following files for each problem. Please submit a compressed file which includes all of them.

Problem 1: Figures from step 1 and 3. Two affine transformation matrices from step 3 and step 6.
Problem 2: Computed affine transformation matrix. Reference image and the aligned image.
Problem 3: FOR EACH TEST IMAGE: Computed affine transformation matrix. Comparison between the warped test image and the matched object image.
Problem 4: FOR EACH IMAGE: Visualized HOG features, classification scores, and a figure showing the location of detected object.

Plus, a report file containing your answer or description for each question (PDF format preferred). Please clarify which files are for which question (by writing in the report, organizing the folders, etc.) You may include images as figures in the report document instead of independent files.