Assignment # 03

IT-Universitetet i København
Introduction to Image Analysis and Machine Learning
(Spring 2017)

May 03, 2017

Purpose In this assignment you will explore various applications of stereo vision. You will also work on human face detection using Principal Component Analysis (PCA). You are going to do this assignment during the exercises classes, but expect also to use some time outside class to finish it. Please follow the plan closely otherwise you will be busy in the last week. Notice, this is the last mandatory assignment of the course.

It is therefore important that you don't fall behind schedule. It may be that you won't finish everything of this document this week, but get as far as you have time for. If you are stuck in any part of this assignment, please don't hesitate to ask for assistance (also outside class hours) so that you do not waste time on things we could easily help you with. We expect that you do your best in solving the majority of the assignment, but if you are unable to do this, please contact us for a *Plan B* and do NOT wait until a few days before deadline.

Reports The reports will be a part of the exam but we are also aware that it may be slightly challenging. The **report for this assignment** should be handed in by the same groups as in Assignment #01 and Assignment #02. The report should be self-contained and not just answers to the questions. In this assignment, you should possibly delegate the tasks among group members. The report should additionally be handed-in together with the other assignments by the end of the semester in learnIT. Make sure your report contains:

- □ Your **names** and **emails** on the front page;
- □ The filename should be as follows: `Assignment3_GROUPINDEX.zip`
- □ A link to where the code can be download (e.g. GitHub, Dropbox); and
- □ The printed program.

Your report should NOT be a list of answers to the questions given in the exercise sheet. The questions should be considered as a guide that can help you to write a good report. The report should then either directly or indirectly answer the questions. Each report should be self contained and you have to describe what is the overall method (perhaps supported with figures showing individual steps) and what your assumptions are. Remember to show that your developed code works (testing). This involves showing image sequences (multiple images), when the system works and when it does not work. So read through this document carefully and decide how you can work in the best possible way.

The **final report** (collection of all assignments) should be handed on learnIT at the end of the semester and it should contain: Please make sure that none of the examination contents is hosted other than ITU servers.

- □ Your written report, perhaps arranged nicely; :)
- □ A DVD with your report(s);
- □ Code; and
- □ Result videos and possibly test data.

Each mandatory assignment MUST be passed to be able to go to exam. Please see the guidelines for writing a report in learnIT. **Remember:** There is NOT only one solution to this assignment and we encourage you to be creative when solving it :)

Learning Goals The learning goals of the Assignment #03 are to explore some aspects of multiple calibrated cameras, e.g. Fundamental Matrix (F), Essential Matrix (E), Epipolar lines, rectification transforms and depth maps. You will also learn how to select a set of eigenvectors with highest eigenvalues (Principal Components) for dimensionality reduction in high feature spaces.

Expected outcome for this week Before next week, it is expected that your stereo vision system will be able to estimate every geometry components used by a stereo setup, to estimate and draw the epipolar lines and to create a 3D point cloud of your environment.

Get Code and Data Download `AssignmentMaterial_3.zip` from learnIT. Extract the zip file and copy the contents into the source folder of your project. The *Assignment3.py* file contains the code skeleton for the assignment. You are naturally free to make changes this file (or any class) as you prefer. These classes have various auxiliary methods that may become helpful in solving the assignment. Use the OpenCV 3.0.0 online documentation for additional information and help about the OpenCV functions.

Tasks, Exercises and Numbers The text contains several questions that will guide you through the assignment. *Tasks* are activities to test or improve your system. *Exercises* are mandatory activities related to development of this assignment. *Extra Exercises* are optional activities can help you to achieve a better solution if time permits. Activities marked with a (^{1}) are mandatory to develop during the next exercise class; higher numbers are optional for this week but you have to develop them in the next weeks. Individual questions marked with "$^{(Extra)}$" are not required (e.g. if time is short).

For developing this assignment, you will need to use at least two web cameras. You will be able to borrow the cameras from either Fabricio (fabn[at]itu[dot]dk, office 4D25) or Zaheer (zahm[at]itu[dot]dk, office 4D25).

Exercise 3.01. *Fundamental Matrix* – In this exercise, you will develop a method to estimate the fundamental matrix (`F`) based on 8 point algorithm, i.e. you are going to use 8 pairs of correspondent points selected by the user in both stereo images, as shown in Figure 1.

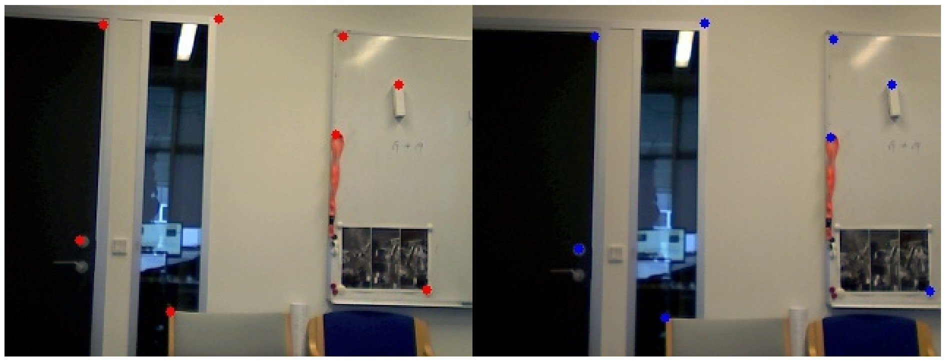

Figure 1: Example of 8 pairs of points selected by an user.

Connect both cameras in your laptop and run the script `fundamentalMatrix.py`. You must press the letter "*F*" of your keyboard to freeze the current stereo images. After that, it will be possible to select the pairs of points used for estimating the fundamental matrix. First, you must select a point in the left image and after selecting the correspondent point in the right image. The selected points will be saved in the global variable `queue` through the function `addNewPoint()`. When `queue` will be full, the script will estimate `F` through the function `fundamentalMatrix()`.

In the original code, the images are processed in their original size (640×480). The system combine the left and right image in only one image (see `np.hstack()`). For this reason, the output image is 1280×480 pixels. The easiest way to estimate the parameters used by the stereo cameras is using the images in their original size. Otherwise, if you want to use the smallest image, you have to adapt your code for this purpose.

For the report In the report, you should specify your assumptions on how you estimate a fundamental matrix based on pairs of correspondent points from stereo images. In the report you have to discuss under which circumstances your method fails/works and why?

(a) Develop the function `fundamentalMatrix()` to calculate the `F` based on a set of selected correspondent points in a stereo view.

(b) Slice the `points` variable to get the points selected in each image. The points selected in the left image are in the odd lines and the points selected in the right image are in the even lines.

(c) Calculate the Fundamental Matrix (`F`). Use the function `cv2.findFundamentalMat()` to calculate F. *Hint:* You have to use the selected points from the left and right images in this function.

Now, you will test the Fundamental Matrix (F) through projections of the epipolar lines over the points used to create F. *Hint:* You can find more information how to estimate epipolar lines in the reading material [SO], [TV] and [BR].

(d) Estimate the epipolar lines of the right image using the points selected in the left image.

(e) The epipolar lines are estimated in the general (or standard) form of the linear equation, i.e $ax + by + c = 0$. Before drawing the epipolar lines in the right image, you must calculate the initial point (x_i, y_i) and final point (x_f, y_f). Adapt your code to estimate the epipolar lines in the left image.

(f) Draw the epipolar line in the right image. Use the function `cv2.line()` to draw a line in the global variable `stereo`. *Hint:* Select different colors to draw the epipolar lines in the left and right images.

(g) Repeat the instructions of Exercise 3.01 (e) and (f) to estimate and draw the epipolar line of the left image using the points selected in the right image.

As result, you are going to see some lines as shown in Figure 2. Test your code using different points. Remember to use some pictures like that in your report.

□ **Output:** A image with the results of the epipolar geometry. You should name this image as: "*EpipolarResult.png*".

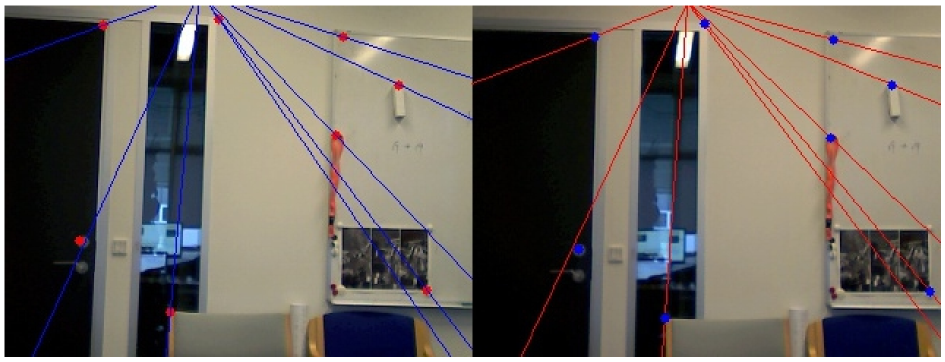

Figure 2: Epipolar lines estimate using a Fundamental Matrix (F).

Exercise 3.02. *Stereo Camera Calibration* – In this exercise, you will develop the calibration process of stereo camera. It is essential to use two cameras during this exercise. Remember to borrow the seconde camera from either Fabricio (narcizo[at]itu[dot]dk, office 4D25) or Zaheer (zahm[at]itu[dot]dk, office 4D25).

For the report In the report, you should specify your assumptions on how to calibrate two stereo camera. Show step by step the process for calibrating your own webcams.

(a) Print the chessboard provided by OpenCV (there is a file called *pattern.png* in *Images* folder with this Chessboard Pattern).

In the next steps, you will develop the calibration process of stereo cameras using some OpenCV functions. The calibration will be performed by the function `calibrate()` in the script `stereoCameras.py`. When a pattern (chessboard) is recognized in both input images (as shown in Figure 3), you have to press the letter "*S*" in your keyboard. The script will fill out three vectors (i.e. `leftCorners`, `rightCorners` and `objectPoints`). These vectors are used for estimating the camera matrix (`K`) and distortion coefficients of each camera and performing the stereo calibration process. The `leftCorners` and `rightCorners` contain the chessboard coordinates from the left and right image, and `objectPoints` is an outer vector contains as many elements as the number of the pattern views.

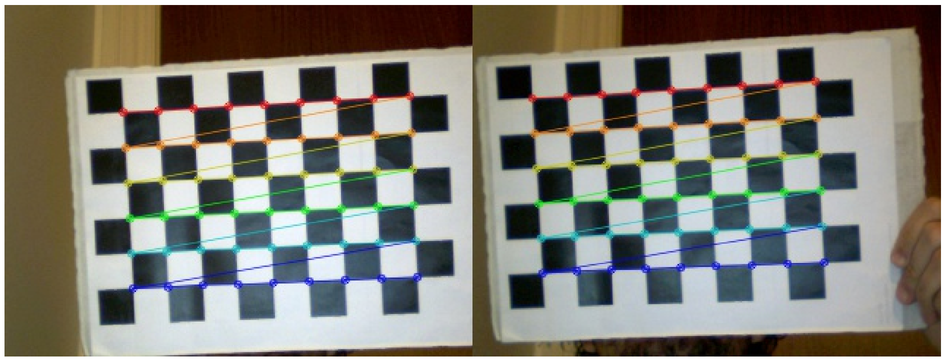

Figure 3: Example of identified chessboard during the stereo cameras calibration process.

Calculate the intrinsic and extrinsic parameters from each camera (i.e. *left* or *right* camera) based on several views of a calibration pattern.

(b) Use the function `cv2.calibrateCamera()` to calibrate the individual camera. You must calculate two variables, namely: (i) `cameraMatrix` (output 3×3 floating-point camera matrix); and (ii) `distCoeffs` (output vector of distortion coefficients).

For the calibration process of stereo camera, use the function `cv2.stereoCalibrate()` (read more information about this method here). After that, update the camera matrices and the distortion coefficients attributes of each camera. Although the function `cv2.stereoCalibrate()` returns the essential matrix (E) and the fundamental matrix (F), you can estimate these matrices without using OpenCV. You only need to know the rotation matrix (R) between the each stereo camera coordinate systems and the translation vector (t) between the coordinate systems of the stereo cameras.

(c) Estimate the skew symmetric matrix. Before calculating Essential Matrix (E), it is necessary to develop a method that returns the cross product matrix ($[a_x]$). *Hint:* You can find more information in the reading material [SO][BR] and in the lecture slides.

(d) Calculate the Essential Matrix (E). Use the estimated skew symmetric matrix ($[a_x]$ – a 3×3 matrix) and multiply it by the rotation matrix (R), according Equation 1:

$$E = [a_x]R \tag{1}$$

Hint: You can find more information in the reading material [SO] and [BR].

(e) Calculate the Fundamental Matrix (F). Use the parameters expressed in components of the two camera matrices, i.e. their relative translation (t) and rotation (R), and calculate F according Equation 2:

$$F = K_2^{-T} E K_1^{-1} \tag{2}$$

Hint: You can find more information in the reading material [SO] and [BR].

(f) Compare the results returned by the function `cv2.stereoCalibrate()` with the results returned by your own methods. Do they have the same values? Write in the report your conclusions about these tests.

Now, you will finish the calibration process of stereo cameras.

(g) It is necessary to compute the rectification transforms for each head of a calibrated stereo camera setup. Use the function `cv2.stereoRectify()` and the parameters you have estimated with the function `cv2.stereoCalibrate()`.

(h) The last step is to compute the undistortion and rectification transformation maps. Use the function `cv2.initUndistortRectifyMap()` to estimate the mapping for each camera. *Tips:* Record the returned maps in the global variables `map1` and `map2`.

As result, you will see an OpenCV windows with four images, as shown in Figure 4.

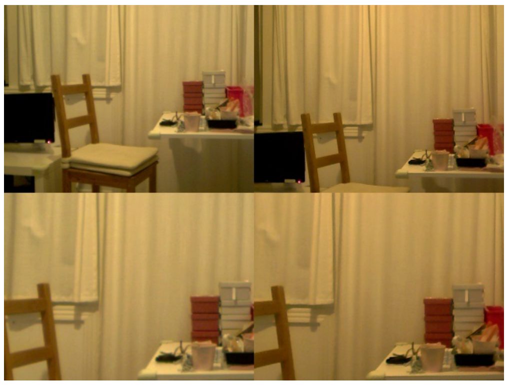

Figure 4: (Upper) The original stereo images; and (Bellow) The undistortion images after the rectification transformation.

Face Identification using Principal Component Analysis (PCA)

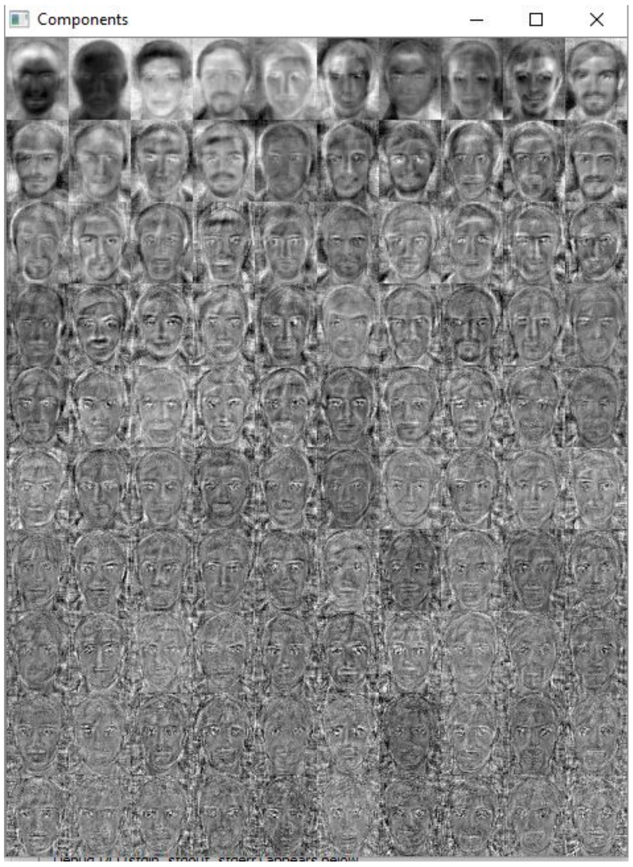

Figure 5: First 100 Eigenfaces

In the assignment resources, navigate to the folder `eigenfaces` and read through the `eigenfaces.py`. The following sections will explain the code step by step.

Datasets and PCA

After required `imports`, argument parser is constructed and command line arguments are parsed.

- `--dataset`: path to the dataset.
- `--num-components`: number of principal components to be selected from an ordered list of components.
- `--sample-size`: number of samples (faces) in the test set.
- `--visualize`: boolean value to visualize the montage of eigenfaces and mean face. Set a positive value to enable the visualization.

In order for face recognition to be successful (and somewhat robust), you should ensure you have multiple images per person you want to recognize. Essentially, this first step is to construct the training set of images. Loading the face dataset is accomplished by using the `load_caltech_faces` helper method. The face dataset resides in the folder `caltech_faces`. Set `--dataset` argument with the `path` to this folder. `load_caltech_faces` helper method loads a subset of the CALTECH Faces dataset, supplying `min_faces=21` to indicate that, for any given person to be included in the face recognition classifier, you must have at least 21 examples of their face. You will also use 75% of the data for training and the remaining 25% for evaluating your face identifier.
Each face is represented as a grayscale, `K x K` bitmap of pixels (images do not have to be square, but for the sake of this example, its easier to explain if we assume square images). In order to apply the Eigenfaces algorithm, you need to form a single vector from the image. This is accomplished by "flattening" each image into a K^2-dim vector. After each image in the dataset has been flattened, you form a matrix of flattened images (Figure 6) Where `Z` is

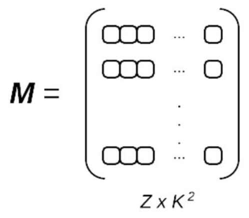

Figure 6: Entire Caltech faces dataset

the number of images in the dataset. The entire dataset is now contained in a single matrix `M`. Given this matrix `M`, you are now ready to apply Principal Component Analysis (PCA), the cornerstone of the Eigenfaces algorithm.
A complete review associated with the linear algebra underlying PCA can be seen at this link: Implementing a Principal Component Analysis (PCA), but the general outline of the algorithm follows:

1. Compute the mean μ_i of each column in the matrix, giving us the average pixel intensity value for every `(x, y)`-coordinate in the image dataset.

2. Subtract the μ_i from each column c_i this is called mean centering the data and is a required step when performing PCA.

3. Now that our matrix `M` has been mean centered, compute the covariance matrix.

4. Perform an eigenvalue decomposition on the covariance matrix to get the the eigenvalues λ_i and eigenvectors $\mathbf{X_i}$

5. Sort $\mathbf{X_i}$ by $|\lambda_i|$, largest to smallest.

6. Take the top `N` eigenvectors with the largest corresponding eigenvalue magnitude.

7. Transform the input data by projecting (i.e., taking the dot product) it onto the space created by the top `N` eigenvectors these eigenvectors are called our **eigenfaces**.

Again, for a complete review on manually computing the covariance matrix and performing an eigenvalue decomposition see link: Implementing a Principal Component Analysis (PCA). All seven of these steps are handled via Lines 35 and 36. You indicate `N`, the number of principal components you are going to use when initializing the `RandomizedPCA` class. The scikit-learn implementation of `RandomizedPCA` is used rather than simple PCA to speed up the process and make it more scalable. After the top `--num-components` are found, they are used to project the original training data to the Eigenface subspace. However, before you perform actual face identification using the Eigenfaces algorithm, lets actually discuss these eigenface representations: Each row in the matrix above is an eigenface with K^2 entries

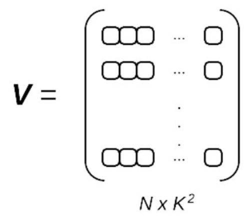

Figure 7: After applying an eigen value decomposition of matrix M

exactly like the original image. What does this mean? Well, since each of these eigenface representations is actually a K^2 vector, we can reshape it into a `K x K` bitmap.

Exercise 3.3.1: Set `--visualize` argument to some positive value to visualize the eigenfaces and the mean face.
Exercise 3.3.2: With default values you can visualize first 16 eighenfaces. Modify the code to visualize fist 100 eigenfaces.
Exercise 3.3.3: Can you explain why the eigenfaces down the list have less details?
Exercise 3.3.4: How can you relate the eigenfaces with the plot of eigenvectors you get in the beginning when you run your script?

Identification

Now our training data contains only the selected principal components. An SVM is trained using a `radial basis kernel` and fitted it to the training data. The classifier is evaluated which provides a classification report.

Exercise 3.3.5: Change selected eigenfaces to `5` and generate classification report. Discuss this classification report in your final exam report. Why does the identification accuracy deteriorate?

Given the eigenface vectors, you can represent a new face by taking the dot product between the (flattened) input face image and the `N` eigenfaces. This allows you to represent each face as a linear combination of principal components.

Exercise 3.3.6: Loop over the desired number of samples (faces) and classify them. Display the predicted face and label.

Exercise 3.3.7: Can we generalize this method for objects other than faces? If “yes”, give your arguments?