HW3 – submission 10.6.19, 23:55

Guide lines

1. Include all your personal details including name, id, and **e-mail address.**
2. You should submit all functions and script files written in **MATLAB or Python**. Your code should be well documented and clear. The code should run from **any** computer and include all path definitions (You should take care of this in the code).
3. Please divide the code by questions.
4. Final report – should include explanations on the implementation and the execution, answers to the questions, results, conclusions and visual results. Do elaborate on all parts of the algorithms/solution. The grades are highly depended upon the analysis depth of the report. **Please submit a PDF file and not a DOC file.**
5. Please post question regarding this HW on the QA forum in Moodle.
6. Eventually submit one zip file including the code + images PDF. HW can be submitted in pairs. In such case only one of the student should submit the solution. The zip file should be named HW#_id0_id1 (for example: HW1_123456789_012345678).
7. Please follow all the submission instruction as describe here.

Good luck!

Optical flow

In this assignment, you are given a short video "seq.gif". Your goal is to estimate the optical flow between every two neighboring frames, using the Lukas-Kanade optical flow algorithm. Remember that for Lukas-Kanade, for each flow vector that you estimate, you will be choosing a region over which to analyze the two frames.

For this assignment, use regions of size NxN pixels which are **not overlapping**, where N=16.

(Since the input frames are 512 by 512, you should have an array of 32 by 32 optical flow vectors at the end of your procedure.)

The Lukas-Kanade algorithm works by trying to find an optical flow vector with the components (u,v) for each region that minimizes the error of the following series of equations:

$$\begin{bmatrix} I_{x1} & I_{y1} \\ I_{x2} & I_{y2} \\ \vdots & \\ I_{xN^2} & I_{yN^2} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_{t1} \\ I_{t2} \\ \vdots \\ I_{tN^2} \end{bmatrix}$$

Recall that I_{x1} is an estimation of the image derivative in the horizontal direction at the first pixel in the image region (hence the superscript "1"). For the purposes of this assignment you can just take that to be the difference between that first pixel and the one immediately to its right. (NOTE: You will have to do something special for the image regions that are on the rightmost edge of the image, since those regions will not have a pixel immediately to the right of the rightmost pixel. Do whatever you like to deal with this. It won't have a major impact on the results.) The last index N^2 is the size of the region (16x16 here).

This equation can be rewritten in matrix form as

$$A\vec{\mathrm{u}} = b$$

Where

$$A = \begin{bmatrix} I_{x1} & I_{y1} \\ I_{x2} & I_{y2} \\ \vdots & \\ I_{x255} & I_{y255} \end{bmatrix}, \quad \vec{\mathbf{u}} = \begin{bmatrix} u \\ v \end{bmatrix}, \quad b = - \begin{bmatrix} I_{t1} \\ I_{t2} \\ \vdots \\ I_{t255} \end{bmatrix}$$

And the solution can be found using least square:

$$\vec{u} = \left(A^T A\right)^{-1} A^T b$$

For all the neighboring pairs of frames:

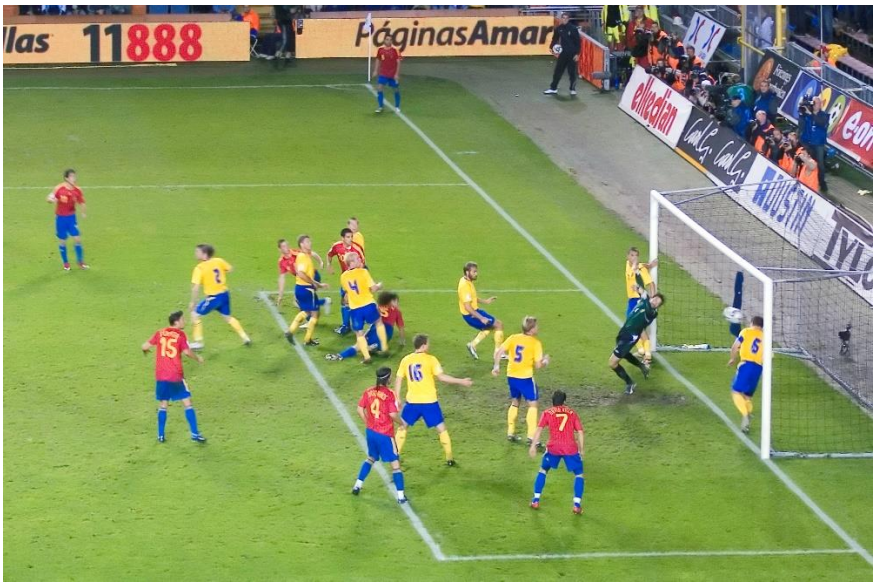
1. For each region in the image:
    a. Compute the matrix $A^T A$ and the vector b
    b. Compute the eigen-values of $A^T A$
    c. If the smallest eigan-value is larger than a threshold t=1, continue to (d), otherwise move to the next region.
    d. Estimate (u,v) for the current image region by solving $\vec{u} = \left(A^T A\right)^{-1} A^T b$.
2. From the estimations of (u,v) at each region, display the flow over the full frame using `quiver()`. This function plots a set of two-dimensional vectors (the flow field in our case) as arrows. **Please plot the flow on top of the frame**, using `hold all`**.**
3. Repeat for t=0.1
4. Repeat for N = 8, t=1,0.1
5. Describe in general effect of changing N and t.
   (There is no need to present the flow of all the frames. Select only a few examples for each case)

Note: it is important to convert the images into double (use `im2double`**).**

Camera Calibration

This question deals with a controversy from the qualification games of the European Championships in soccer, 7 October 2006. After a corner kick in the game between Sweden and Spain it was hard to tell if **the ball crossed the goal line or not**.

Here is an image from that scene (also included in the exercise files):



Your task is to find the camera location relative the field. To your help, we provide you a matlab script (called inl1.m) which gives you some measures of the goal area.

a. compute the camera matrix P which generated the image points x according to the camera equation:

$$x = PX$$

Use the linear DLT method which was presented in the tutorial.

b. Re-project all the real world point X to their estimated corresponded points $\tilde{x}$ on the image plane using the estimated matrix P. Define a way to calculate the estimation error based on $\tilde{x}$. Explain the error measure you defined. What are its units?

We will now determine the intrinsic parameters K and the extrinsic parameters R, t

c. Use the given function "rq.m" to reconstruct R, K.

1. Describe the goal of this function. What is the difference between the given function and the matlab function called “qr.m”? Why didn’t we use the qr function?
2. Explain the operation done at each of the function lines.

d. What can you say about the camera characters, from the matrix K ? Are the intrinsic camera parameters reasonable?

e. What can you say about the orientation of the camera from the matrix R ? Is that reasonable?

f. Calculate the translation vector t . This translation is given in the rotated camera space. To calculate the translation in the world space c use: $t = -Rc$. (see full explanation in the “Multiple View Geometry in Computer Vision” pa. 155-156). Is c reasonable?

g. Plot the camera location onto the given 3D field model.

h. Using all the above information, can you determine if the ball actually crossed the goal line? If yes, give an explanation and calculation. If not, explain why.

Depth with stereo

In this assignment, you will implement a stereo matching algorithm, to compute the pixel disparities between a stereo pair of images. The input will be a **rectified** pair of images, and the output is a matrix whose values indicate the disparity of the stereo correspondence for each pixel in the left image.

0. Explain (in only a few sentences) how does the disparity map connected to stereo depth estimation.

Window matching:

1. Implement a function that gets a template window of size `[w,w,3]` and an image of size `[N,M,3]` (`N,M≥W`), and returns the SSD between the template window to each of the image's windows of size `[w,w,3]`.
   It is highly recommended to use the matlab function `blockproc`. You may also use the function `im2col`.

Find corresponding pixels:

2. All the given images in the folder "Depth with stereo data" were already rectified. Explain how this simplify the stereo correspondence problem. Hint: what are the candidate corresponding pixels?

   In our algorithm, we will search for correspondence based on windows SSD values:
   - The pair of stereo images is `I_left` and `I_right`.
   - For each pixel in the left image, `x_left`, find its surrounding window of size `[w,w,3]`, `w_left`.
   - Search for `x_left` corresponding pixel in the right image: `x_right`. This will be done by finding the pixel in `I_right` which its surrounding window of size `[w,w,3]`, `w_right`, gives the minimal SSD value to `w_left`. Use the function from (1). ***This search will be done only on candidate corresponding pixels and not on the whole image.***

Constructing the disparity map:

3. Given two images `I_left` and `I_right` implement a function that finds, for each of the left image pixel's `x_left`, its corresponding pixel in the right image `x_right`, and construct the disparity map `x_left - x_right`. Make sure that the disparity values are reasonable:
   - If the disparity value is negative, assign it to zero.

- If the disparity value is larger than some threshold value `DisparityMax` assign it with this maximal value (what are the units of `DisparityMax`?).

Experiments:

4. Run the algorithm on all the stereo pairs given in this exercise. Use the following parameters:
   a. Max disparity: `DisparityMax = 60.`
   b. Window size: `W =[5,11,21].`
5. Compare the results to the given ground truth disparity maps using RMSE. Where does the algorithm preform best? Where less?