Assignment 2 - part 1

Purpose In this assignment you will explore various applications of homographies. The use of homographies are related to both computer vision and computer graphics applications and is in general very useful for many applications.

Each section specifies what kinds of images or plots you should generate. The report should be self-contained and not just answer the questions. You should use the same groups as in assignment 1. Again for this assignment, you should possibly delegate the tasks among group members.

Plan

The assignment spans several exercises and each week will usually build upon previous weeks. It is therefore important that you don't fall behind schedule. It may be that you will not finish everything in this part of the assignment this week, but get as far as you have time for. **If you are stuck in a part of the assignment please don't hesitate to ask for assistance (also outside class hours) so that you do not waste too much time on things we could easily help you with.** The reports will be a part of the exam but we are also aware that it may be slightly challenging. 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.

After this week you should have finished the questions related to "person map location" and texture mapping.

Submission The deadline for the report is **16th April**. Please try to finish one part every week as the subsequent parts are also meant to last (at least) one week. The report should additionally be handed-in together with the other assignments by the end of the semester to the examination office. Make sure the report contains:

- Your written report with names and emails on the front page
- Result videos and possibly test data.
- A link to where the code can be downloaded
- 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 implementation works (testing). This involves showing image sequences (multiple images), when the tracker works and when it does not work. As with any other group work, it is important that you distribute the workload among the group members. 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 in to the examination office at the end of the semester. The final report should contain:

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

Learning goals The learning goals for this part of the assignment is using homographies for point transfers and texture mapping.

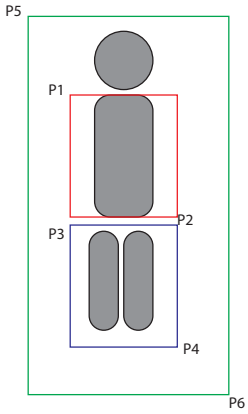

Figure 1: Regions

Person map location[1]

In this part you will make a component to show where on an overview map a person is walking. You do not need do the tracking your self, as the tracking data has been provided. But you have to show the location of a person walking on the ground floor in an overview map. However, please feel free to experiment with background subtraction and connected components or any other of the techniques that you have heard about in the course to improve the results.

Outcome The outcome of this part of the assignment is a figure or movie where it is possible to see where in the overview map the person is walking.

Data The image sequence ('SunClipDS.avi'), the corresponding tracking data ('trackingdata.dat') and the map of ITU building ('ITUMap.bmp') can be found in the *assignment2.zip* on learnIT. Some functions have been provided in *Assignment2.py*. You can start with making changes to this file, but you may want to make additional file to get a better structure of your code.

The file 'trackingdata.dat' can be loaded in python through `loadtext('trackingdata.dat')`. Each row in the resulting matrix contains the coordinates of three rectangles in the corresponding image in the sequence (row 1 contains the coordinates of the rectangles in images 1, and row 2 the rectangles of image 2, etc.). Each rectangle corresponds to a subregion of the person being tracked where a set of coordinates pairs (x_1, x_2) and (x_2, y_2) are the corners of one rectangle. Hence, the rows in 'trackingdata' contain three rectangles (e.g. 12 numbers) - See figure 1. The function *showFloorTrackingData* in *Assignment2.py* shows how to load and display the tracking data.

The image sequence ('SunClipDS.avi') will in the following be denoted S, the ground floor in S is denoted G and the map of ITU, M (see figure 2).

1. Run the function *showFloorTrackingData* in *Assignment2.py*. You should now see 3 boxes surrounding the person walking on the ground floor.

2. The transformation from the ground floor, G in S to the overview map M is defined by a homography – why?

3. How many point correspondences are needed to estimate a homography?

Initially you have to calibrate the system so that points on the ground floor, G can be mapped to the overview map via a homography, $\mathbf{H}_G^M$ - see figure 2. That is, the points in the ground floor and the overview map that correspond to the same points in space should somehow be defined. The corresponding points can be used to estimate the homography between planes in the two views. You will in the following be selecting the corresponding points manually via mouse clicks.

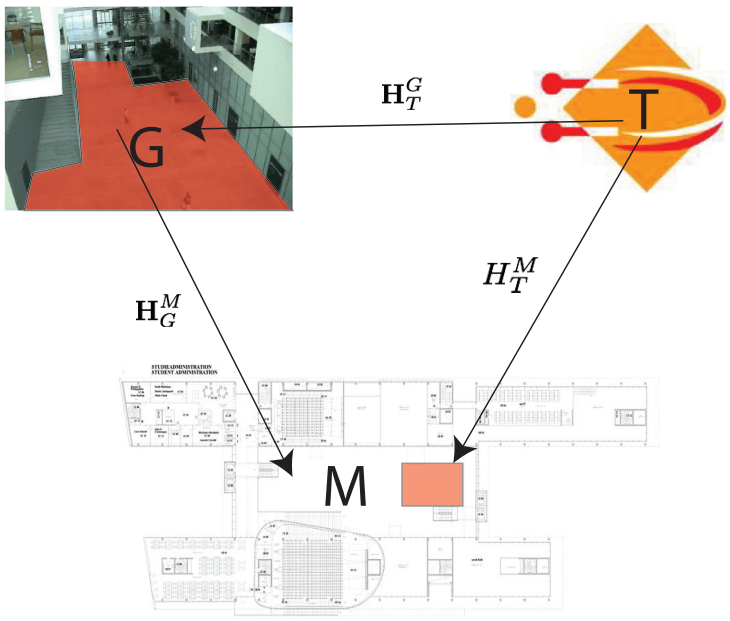

Figure 2: Naming of planes and transformations

4. Read the help for the function `getHomographyFromMousePoints` in a python prompt.
5. Have a look at the code for `getHomographyFromMousePoints` to get an overview of what it does. You will perhaps later need code fragments of the function in your own implementation.
6. Use the first image (or any image you like favorite one) in the image sequence S and M (map of ITU) to find the homography,$\mathbf{H}_G^M$, from ground floor, G to M. You can use *getHomographyFromMousePoints(I1,I2,N)* to select the corresponding points in the two images.
7. Why is only one calibration with four points needed for the entire sequence?
8. What happens if the camera or ground floor move relative to each other?
9. Make a function `DisplayTrace` that uses the estimated homography to display the trace of the person in the overview map. Notice, you should only map points that are on the ground floor to the overview map (e.g. the feet and not the head). The example function *showImageandPlot(N)* should give you sufficient information on how you can display and save the image data.
10. Extend `DisplayTrace` so that it saves the result into an image. The image should be used in the report.
11. Extend `DisplayTrace` so that it also saves the homography in a file *H_G_M*. Hint use `save` and `load` from Numpy to save and load numpy arrays.
12. * Since tracking data on the map is free from perspective distortions you can better describe the movements of the person. What are the benefits of using the map data when analyzing the tracking data rather than in the image view? When does the method fail?.
13. Make a function that for each image in the sequence shows the current position of the person in the overview map e.g. like a real-time system.
14. Make a movie showing the current frame in the sequence and and the current location of the person in the overview map. Add this video to the final report for the examination.

Linear Texture Mapping[1]

Ground Floor

The function `simpleTextureMap` is an example of how linear texture mapping can be done using Opencv.

15. run `simpleTextureMap` and use it to texture map the large area in front of the auditorium in the overview map with the image of the ITU logo.
16. Make a function `texturemapGroundFloor` that places your favorite texture on the ground floor, G, for each image in the image sequence S. Notice when setting the N parameter to negative values (e.g. -4) in *getHomographyFromMousePoints(I1,I2,N)* that the method assumes the corners of the I1 image are used as input. In this way you only have to select points in the destination image to perform the texture mapping.
17. * Change the weights when adding the textures so that the the texture appear as nicely as possible

Moving object

In the following you will experiment with texture mapping on image sequences where the objects move. The directory *GridVideos* contains a few video sequences in which you there is a grid pattern which should be texture mapped with the the ITULogo.jpg. The function `texturemapGridSequence` shows how to detect the grid in an image sequence.

18. Perform automatic texture mapping on the image sequences in the directory.
19. In some of the frames it appears that the texture map fails yet it does so in a consistent way. What happens?
20. * Implement a way of solving the problem so that the texture is mapped even during rotations.
21. * The homography maps points to the grid. But how can this homography be used to map to the corners of the paper?
22. * Implement texture mapping so the entire paper is covered by the texture.

Ensuring a correctly placed texturemap

When you select four points in the target image you cannot be sure that the texture looks "realistic" compared to the geometry of the texture - in fact you can easily select the four destination points such that the texture map looks highly distorted compared to the geometry of the ground floor (see e.g. fig.3). In this exercise, you will implement a method which given a texture, a desired scale, σ, of the texture in the overview map and a point p in the overview map, M, will place the texture on the ground floor. The mapped texture on the ground floor (G) should therefore look more realistic.

23. How can the homography $\mathbf{H}_T^G$ from T to G via M be obtained when $\mathbf{H}_G^M$ is known?
24. Implement a function `realisticTexturemap(H,scale,point,map)` that given the homography $\mathbf{H}_G^M$ (from question 6) and a scale of the texture, allows you to select a location in the overview map using a single mouse click. The output should be a texture on the ground floor $\mathbf{H}_G^M$.
25. How can the homography $\mathbf{H}_T^G$ from T to G via M be obtained when $\mathbf{H}_G^M$ is known?
26. Implement a function `realisticTexturemap(scale,point,map)` that given a scale of the texture loads the homography $\mathbf{H}_G^M$ (from question 6) and allows you to select a location in the overview map using a single mouse click. The output should be a texture placed nicely on the ground floor.
27. * The mapped texture on the ground floor may cover the person walking on the ground floor. Implement a method such that it appears (to the extend possible) that the person is actually walking on the texture. There are many different approaches to go about this e.g.
    - Perform a simple weighting of the pixels within the bounding boxes (highest in the center)
    - Perform additional image analysis of the images within the bounding boxes.

    Ask Dan or Diako if you want some ideas to improve the results.

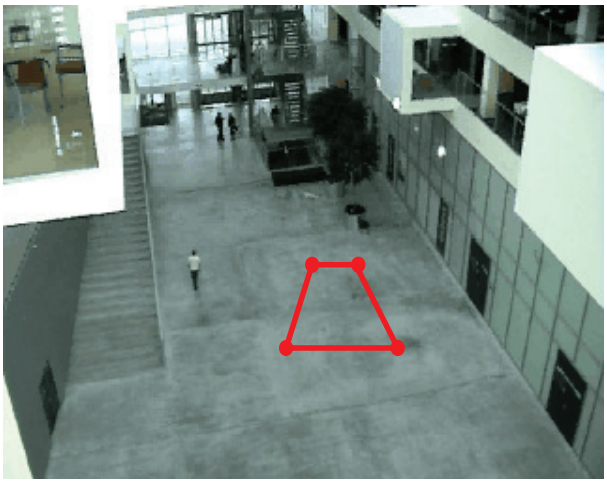

Figure 3: Quadrilateral of points selected by mouse

Ideas for Extra Credit

You can work on the following exercises and document them in you report to get extra credit. Feel free to talk to Dan if you want to implement your own ideas.

Image Augmentation on image reality*

Now you should have a better feel for how to use homographies and to do linear texture mappings. In the following you will extend with texture mapping on objects that move.

28. Run the function *texturemapObjectSequence* within *assignment2.py*.

The method *attempts* to locate a colored rectangular regions in the image sequence, but it is not particularly successful in doing so. Your assignment is in the following steps to improve the detection results using the techniques that you have learned so far in the course. When the rectangular regions are detected in most of the images, then you can proceed with texture mapping the detected region.

29. Change the function *DetectPlaneObject* so that it detects one of the objects on the table more reliably. You may chose any of the objects on the table as you like. A simple extension of the existing pixel classifier should be sufficient, but you may experiment with your preferred method.
30. Once the object rectangle has been located you can perform texture mapping. Extend *DetectPlaneObject* so that it first warps the texture, *texture* to *imgOrig*.
31. Make a sequence showing the results of your method
32. The next step for you is to combine the two images. You can be inspired by the *image_in_image* example on the lecture slides on how to merge the two images.
33. Evaluate your method on the other sequences found in the same directory.
34. [2] Change *DetectPlaneObject* so that the texture mapping becomes more transparent and less dominant. Which values of transparency gives the most realistic results?

Implementing Linear texture mapping

You now have to implement a method that maps an image I onto the planar surface in another image using bilinear interpolation . In particular you are going to make a texture map onto the ground floor of the image sequence. You will have to make your own implementation and NOT use the built-in functions in OpenCv or python. In the first steps you will map the *ItuLogo.jpg* (called I) to the video sequence. Later on you can experiment with other images or videos.

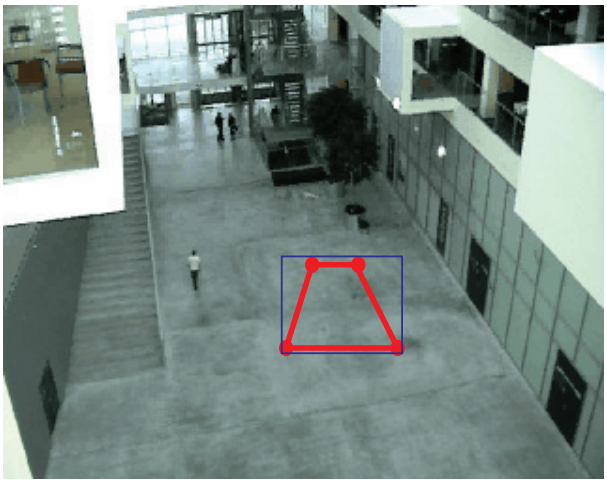

Figure 4: (Red) selected points (blue) bounding box

1. Make a function *forwardMappingITU(I)* which shows the image I and estimates the homography from I onto the ground floor of ITU. You only need to click 4 times on the ground floor since the coordinates of the corners of I are known (e.g. the red region in figure 3).
2. Extend *forwardMappingITU(I)* so it uses forward mapping to map the texture I onto the ground floor.
3. What happens if the region in ground floor becomes very large?

In the following steps you will use backward mapping to improve the results (see e.g. lecture slides). When backward mapping from the four points (red region in 4) try to make the region follow the ground plane (e.g. make it a general quadrilateral). When doing the backward mapping it is usually easier to define a bounding box around the points (blue rectangle) and then do backward mapping. Points that are outside red region will be mapped to a point outside I (simple if-statement to check this). Remember to check that all indices are inside the image.

4. Make a function *backwardMap(I,I2,pts,method='NN')* that uses backward mapping (nearest neighbor) to texture map the image I onto a region specified by pts in $I2$. The *method* parameter specifies which method to use (for now nearest neighbor).
5. Extend *backwardMap* so that it is able also to do bilinear interpolation in the backward mapping. Does this improve the results – how?
6. Map your favorite image onto the ground floor in 'SunClipDS.avi' using backward mapping. Your implementation will most like not be as fast as the ones in OpenCV. You may speed up the process by only performing the texture map on every N'th frame in the sequence (e.g. every fourth image or so).