Assignment 2, Graphics and Image Analysis

April 25, 2014

Alexander Kirk Jrgensen (akir@itu.dk)
Daniel Lujan Villarreal (dluj@itu.dk)
Mikkel Bybjerg Christophersen (mbyb@itu.dk)

Source code for this assignment can be found at
https://github.com/DarkDecaydence/SIGB-Group67

Contents

1 Introduction

Our purpose in this assignment is to explore the uses of homographies, and apply this theory in order to perform various linear transformations. Projections of points from one 2d image onto another is the main focus.

2 Problem statement

The assignment is split in four parts. Throughout the report, we have made an effort to address all of these assignments.

To help us along the way, we have been given the *SIGB-Tools.py* library, containing useful functions to help us.

In part one, the assignment is to track a person who is moving around in the atrium of the IT-university. To solve this assignment, we have been given the datafile *trackingdata.dat*, containing the data relevant to the person walking in atrium.

The second assignment is to project a texture onto a series of image sequences using texture mapping.

In the third assignment, we are tasked with exploring **camera calibration**, and will be using a image sequence of a chessboard pattern to calculate the camera matrix of the camera that made the recording.

In the fourth assignment we will explore **augmentation**. Here, we will make use of a webcam, calibrate it and project a nonexistent cube onto a surface defined by a chessboard pattern.

3 Solution

Our description of our solution will be rooted in two major parts, as described in the introduction. The following part regards our texture mapping and morphology.

3.1 Floor Tracking

To get in touch with the concepts of morphology, our first assignment was to map the path of a person described by a video file. This mapping should be done, such that the path would correctly appear on an actual map.

The videofile we were provided came accompanied by data related to the movement of the person.

Solving this assignment required us to create the function *DisplayTrace(...)*, which processes the data and draws the trace onto the actual map. This function draws upon some properties of the *showFloorTrackingData()* function, and by its extension the *getHomographyFromMouse(...)* function.

In the *DisplayTrace(img, points, H)* function we start out by going through all points provided by the list "points". These are created in the *showFloorTrackingData()* function, so we start our research there.

The *showFloorTrackingData()* function finds the three boxes that contain the person which we are tasked to track. Each box determains a different part of the person; red is upper body, blue is complete body, and green is lower body. As such, we have chosen to track the "green box", which is the box that describes the feet of our target. The feet should be the best point to go for, since it will be the most precise. Should the camera have a very sharp angle of attack, plotting from the face or torso could give very inaccurate results. The homography will be created such that the floor coordinates get malformed.

Looking at figure 1, we can tell that the difference between the two points $p1, p2$ on the base plane represented by vp_1 is b_l when the angle of attack on vp_2 is sharpened. The difference b_l is dependant on a_l, given that $b_l = \sqrt{c_l^2 - a_l^2}$. $c_l = \frac{a_l}{sin(A)}$

We have chosen the bottom middle spot of the green box, to avoid these complications. This should be the most precise point when considering its proximity to the ground, along with its position horizontally. Given the green box $B_(x, y, w, h)$, we can find the best point $p_(x, y)$ with the following calculations:

$$p_y = B_y + B_h \quad | \quad p_x = B_x + (B_w/2)$$

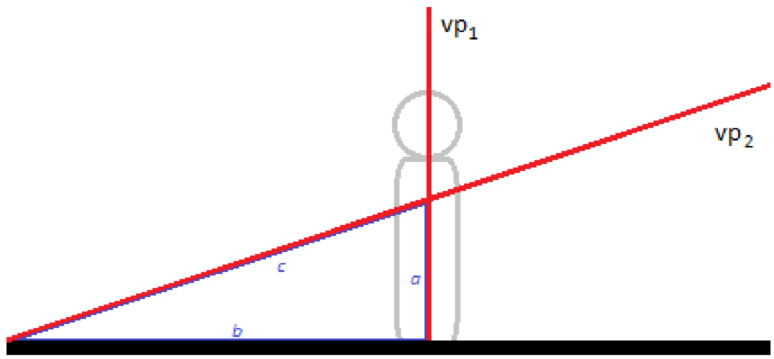

Figure 1: Differences in viewpoints.

Now that we have our point, we need to transfer it into a valid point on our map. We achieve this using a linear transformation in the form f a homography. We can do this because the transformation between the map and the floor is a simple perspective transformation.
In order to estimate the homography we require four corresponding points to be located in each image. This is necessary because of the amount of information found in the homography. Tthe homography is a $3x3$ matrix, and one number, the scaling parameter, is fixed to be 1. If the transformation preserved either angles or parallel lines, we would be able to make assumptions about the matrix and reduce the number of points required, but in this perspective transformation we cannot. This leaves 8 numbers to be identified, and four 2d points offers exactly that amount of information. This intuition is a little oversimplified, as it is not the amount of numbers needed found, but the *degrees of freedom* that matters. If one number in the matrix could be derived from another, those two would constitute only one degree of freedom.
We identify the four points on each image by hand, and then use the homography identified to map points from the path in the sequence to points on the map. Only a single homography is necessary, as the camera does not move throughout the sequence.
The result of this transformation is that the geometric representation of the path is more true to reality. In the image sequence the movement of the person would get smaller as the person moves further away from the camera, due to perspective distortions. Consequently measures of euclidean distance between two points on the path, and therefore measures such as total distance travelled and velocity, would be equally distorted if measured on the image sequence. After the transformation is performed, these measures closer approximate reality, and since the perspective distortion is tried removed, any remaining distortion is likely to e uniform across all points on the path.

With the traced path in hand and translated, we can go paint the path onto the map, in our case, something that is done with the CV2 library and its *cv2.circle(...)* function. The video file *Floor_Tracking.avi* shows the result of locating the path.

3.2 Texture Mapping

In this assignment we work with the projection of textures onto surfaces by use of homographies.
First, we project a texture onto an arbitrary area of the floor in the video sequence from the previous section. We accomplish this by estimating a homography from 4 points from each image, in the same fashion as in the previous section. In this case we are trying to map the entire texture onto the floor, so we can make the more relaxed assumption that the four corners of the texture function as points in the estimation. We therefore only have to choose the points on the floor that are to correspond to the corners of the texture. We then simply apply this homography to the texture and overlays the result on the floor.
Next we have to project the texture onto a chessboard pattern. We solve this

Figure 2: Projection of texture to arbitrary area of the floor.

task primarily by use of the existing method *cv2.findChessboardCorners(image, patternSize)*, which for a given image and a tuple describing the configuration of chessboard corner expected found locates the coordinates of the chessboard corner. In this context a corner refers to a point the lies adjacent to two white and two black squares, and as such what a person would identify as the corners of the board are not part of this set, as these points only touch a single square each. This set is then used to estimate a homography that places the texture so it exactly covers the chessboard corners. Any set of four points could be used for this purpose, but the outermost points are used to maximize precision.
The video file *GridTexture.avi* in appendix shows the result of this transforma-

tion.
Finally we are tasked with projecting the texture onto the floor in the image sequence previously mentioned, but to make the mapping more realistic by using the homography from the floor map to the recording identified in first part of the assignment. We are to assume that the texture if it was to be projected onto the floor and from there onto the map would be exactly rectangular and at right angles with the map. We solve this by first projecting the texture onto the map, and from there project it onto the floor using the homography identified in the first section. The homography for the first projection is found by choosing a point on the map, translating the textures center to this point, and then providing a proper scaling.

Figure 3: More realistic projection of the texture onto the floor.

3.3 Camera Calibration

With the help of an updated version of the *SIGB-Tools2.py* library the first task was to start learning the different functionalities of the file *Assignment_Cube.py*. Once we got to know the code we began with the camera calibration. For this task we used the *calibrateCamera* function in our tools library and a printed chessboard of size 6x9. This function uses several important variables like *pattern_points*, *img_points*, and *img_points_first*. *Pattern_points* are the position of the inner points of the chessboard squares in the calibration pattern coordinate space. *Image_points* are the corners of the chessboard found in the image being captured by the webcam and *image_points_first* contains the *image_points* for a picture taken in the first frame of the video. The calibration is done by taking 5 sample views and saves information like the camera calibration matrix, rotation vectors and translation vectors. By modifying the number of samples taken by

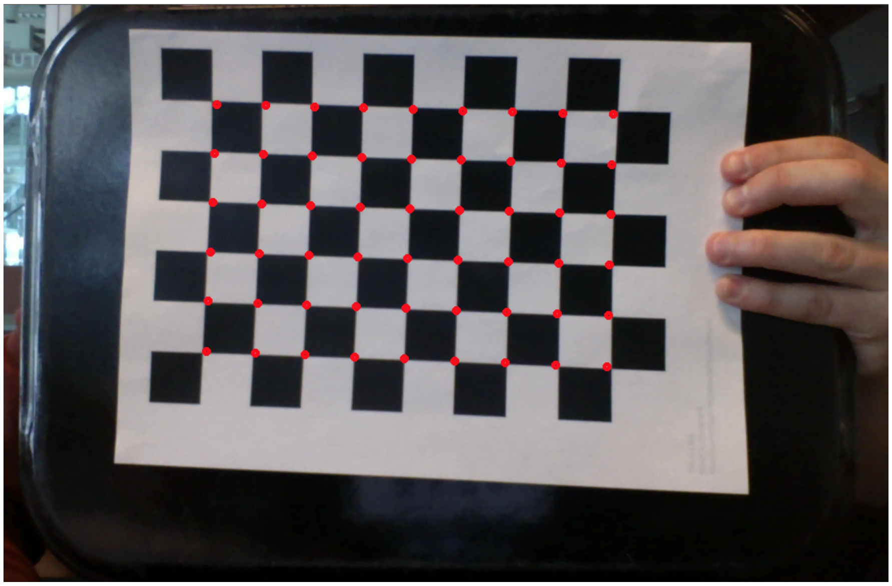

Figure 4: Projection of chessboard pattern points to first view image

the calibration function we noticed that increasing the number of samples also increases the accuracy of the calibration and viceversa.
After calibrating the camera the next step was to calculate the camera matrix $P_1 = K[\ R_1 — t_1\ ]$ for the first view. This camera matrix with its rotation and translation vectors helped us to project the points taken from the chessboard pattern during the calibration of the camera onto the first view image as seen in figure 4.

3.4 Augmentation

For the augmentation part in this assignment the goal is to calculate the camera matrix in each video frame taken from the webcam and project a 3D cube to the same view. The camera matrix is calculated using two different methods: through homographies and directly using extrinsic parameters.
In the first method two Homographies where used. The first Homography is from the chessboard calibration pattern to the first view image, $H_cp - fv$. The second Homography is from the first view image to the current view image from the webcam, $H_fv - cv$. In order to get the second homography we had to locate the 4 outer corners of the calibration patter in the playing video. Having these two homographies we could compute the homography form the chessboard pattern to the current view of the webcam by applying a dot product which gave us the homography $H_cp - cv$. This third homography was used to calculate the camera matrix *P2_Method1*. Several operations had to be applied to get to this result. First, the previous camera matrix was broken apart to get the

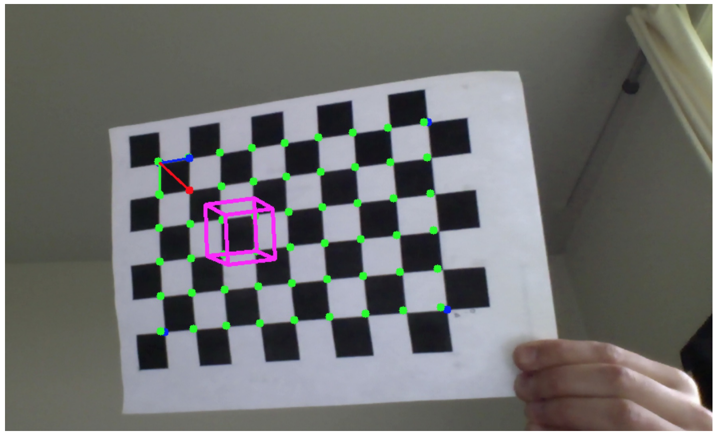

Figure 5: Projection of chessboard pattern points to webcam feed with P2_method2

intrinsic parameters' matrix K which were used to compute the new rotation vector. Second, a dot product between the previous camera matrix and the homography $H_c p - cv$ was applied and the result was complemented with the new rotation vector to conform the new camera matrix *P2_Method1*.
The second method uses the function *GetIObjectPos()* located in our tools library. This function finds the object pose from 3D-2D point correspondences and returns new rotation and translation vectors. The points used for this method are *obj_points* used in the first part and the corners of the current view from the webcam. The new vectors computed by this function are combined and multiplied by dot product with the previous camera matrix to compute the new *P2_method2* camera matrix.

After finding the new camera matrices we proceed to draw the world coordinate axes in the chessboard pattern and project the 3D cube as seen in 5.
The video file *CubeProjections.avi* shows the performance of the two homography methods. First *P2_method1* is applied, and shown in green circles, and then a switch is made to *P2_method2* shown in red circles.

4 Discussion

In this section we discuss and interpret the results of each exercise.

4.1 Floor tracking

We observe that the path located on the floor roughly corresponds to the position of the person in the video at all times. We observe a small jitter in the path, which can be explained by a similar effect in the data set.
As it is assumed that the person is walking on the plane of the floor at all times, we expected a noticeable distortion when the person walks onto the stairs, and thus is raised above the floor. This effect is expected to take place, but is too small to observe.

4.2 Texture mapping

When projecting a texture onto an arbitrary area of the floor we see results as expected. Due to the method of image overlaying employed the part of the image not covered by the texture is needlessly darkened. This could be prevented by using a different overlaying method.
Projecting the texture onto the tacked chessboard works rather well. We see some shaking in the texture when the camera moves, and in some frames the texture can not be tracked. Be attribute this to the camera motion applying a motion blur to the images; when this effect grows too large, the algorithm that identifies the chessboard can not locate the sharp edges expected between white and black squares.
The realistic projection accomplishes its job: the texture is placed on the floor in manner as if it was a rectangular section of the floor itself. The perspective transformation seems to follow the perspective of the camera as well.
We do observe higher degree of pixelation (more ”grainy” image quality) than the arbitrary projection. We attribute this to fact that the texture undergoes two linear transformations rather than one.

4.3 Camera calibration

We managed to identify the camera matrix for the image sequence rather successfully.
We observed when experimenting that using only a few frames for calibration (¡5) resulted in noticeable distortion effects when applying the calibration. The effects where most clearly visible around the edge of the camera. Choosing particular single frames for calibration could cause the sides of the screen to bulge in or out, and objects close to the edges to be extremely stretched. It was very hard to achieve a similar effect when choosing many (¿10) frames for calibration, even when these where picked in an attempt to distort. It was clearly visible that only a few calibration points places unjustified assumptions on the nature

of the camera, and using more points increases the probability of estimating a camera matrix that more closely resembles reality.

4.4 Augmentation

We have implemented both of the two methods in a way such that the movement of the cube follows the movement of the chessboard. As long as the chessboard is held still or moved slowly, and all squares are visible to the camera, the cube can be properly projected.
We have had some difficulties with getting the points using method 1 to align properly with the chessboard. It appears that the points have the perspective and rotation components correctly applied, but that translation and scaling are incorrect. We have not been able to identify the source of these problems.
Method 2 seems to work perfectly most of the time, although sudden quick movements seem capable of offsetting the alignment of points a little. We have not been able to figure out what causes this.
Due to these differences in accuracy we have not been able to properly compare the precision of the two methods, although we have observed that method 2 has a tendency to lower the frame rate to a degree that method 1 does not. Wee conclude from this that method 1 has better performance than method 2.

5 Conclusion

We have been able to solve all the mandatory parts of this assignment in a satisfactory way.
We have applied knowledge of homographies and linear transformations to solve the problems, and have gotten meaningful results in almost all cases.
We have experienced some technical problems, most notably in relation to tracking a chessboard pattern correctly.