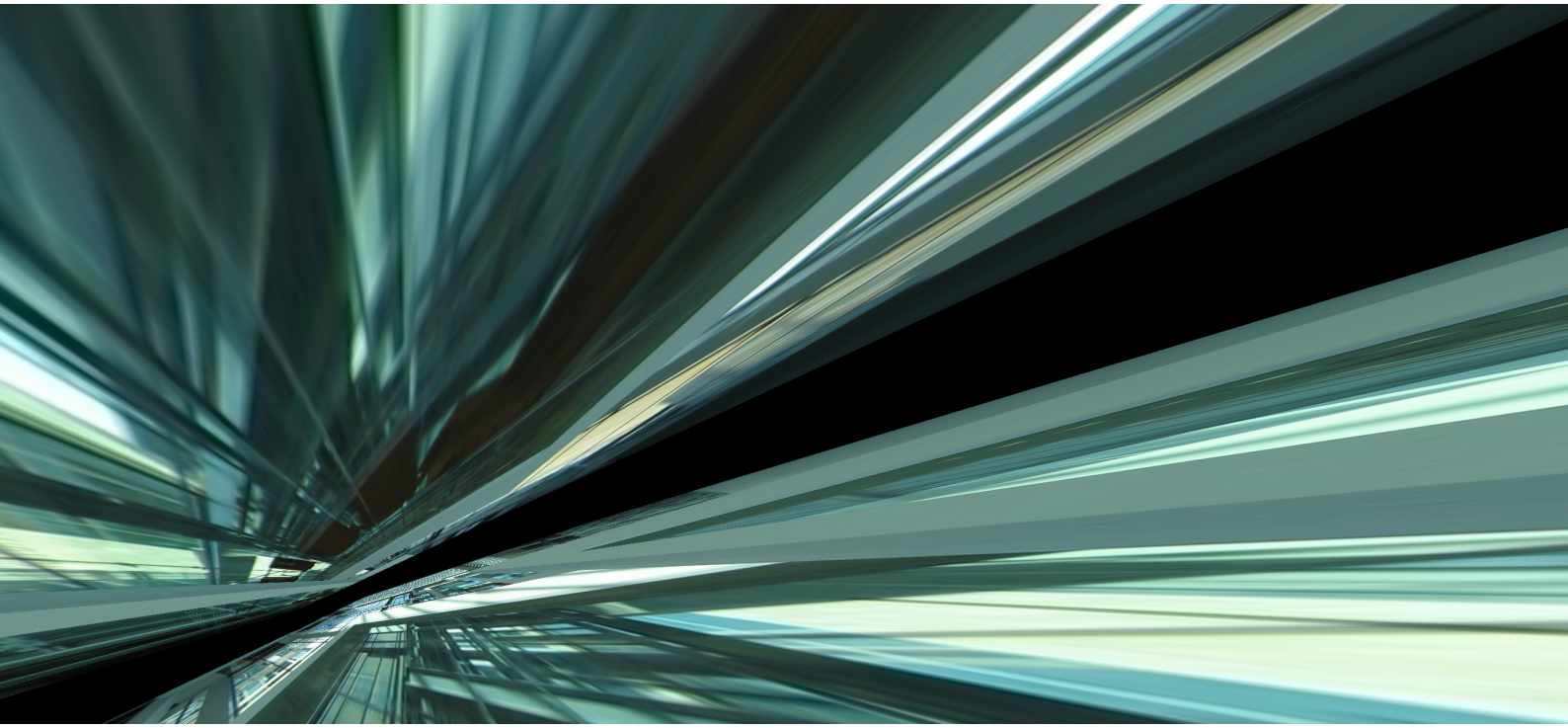

Computer Vision

A 12-week project

Renée Anderson
renee@itu.dk
081169-xxxx

Vedrana Andersen
vedrana@itu.dk
130274-xxxx

Lars Pellarin
pellarin@itu.dk
280979-xxxx

Project supervisor: Dan Witzner Hansen
IT University of Copenhagen, February – May 2006

Acknowledgments

Thanks to Dan Witzner Hansen, our project advisor, whose hard work, dedication, and tenacious love of the field of computer vision are nothing short of inspirational.

Cover-page: Warping the ITU building using a flawed homography, making the case that failure need not be ugly.

Contents

1 Homographies, Mosaic, and Warping **6**

1.1 Projective Geometry and Transformations of 2D 6

1.1.1 Homogeneous Coordinates 6

1.1.2 A Model for the Projective Plane 7

1.1.3 Homography Matrix 7

1.1.4 Homographies in Images 8

1.2 Homography Estimation 8

1.2.1 Overview 8

1.2.2 Direct Linear Transformation 9

1.2.3 Normalization 10

1.2.4 Implementation 10

1.2.5 Results 11

1.3 Image Mosaic 12

1.3.1 Overview 12

1.3.2 Backward Mapping 12

1.3.3 Bilinear Interpolation 12

1.3.4 Hat-Weighting Function 13

1.3.5 Implementation 13

1.3.6 Results 15

1.3.7 Automatic Stitching 17

1.4 Sequence-to-Map Homography 19

1.4.1 Implementation 19

1.4.2 Results 19

1.5 Warp Sequence 22

1.5.1 Implementation 22

1.6 Affine Rectification 22

1.6.1 Theory 23

1.6.2 Implementation 23

1.6.3 Results 23

2 Single-View Geometry, Stereo, and RANSAC **25**

2.1 Projection Matrix 25

2.2 Single View Metrology 26

2.2.1 Preliminaries 27

2.2.2 Geometric Representation 27

2.2.3 Algebraic Representation 28

2.2.4 Implementation 29

2.2.5 Results 29

2.3 Fundamental Matrix . . . 32
2.4 Fundamental Matrix Estimation . . . 33
2.4.1 Theory . . . 33
2.4.2 Implementation . . . 33
2.4.3 Results . . . 35
2.5 RANSAC . . . 36
2.5.1 Theory . . . 36
2.5.2 Implementation . . . 36
2.5.3 Results . . . 37
2.6 Camera Calibration . . . 38
2.6.1 Theory . . . 38
2.6.2 Implementation . . . 38
2.6.3 Results . . . 39

3 Tracking **40**
3.1 Models for Tracking . . . 40
3.1.1 Probabilistic Model . . . 40
3.1.2 Kalman Filter . . . 41
3.1.3 Constant-Velocity Model . . . 42
3.2 Simple Object Tracking . . . 43
3.2.1 Implementation . . . 43
3.2.2 Results . . . 44
3.2.3 Further Experimentation . . . 44
3.3 Eye Tracking . . . 46
3.3.1 Implementation . . . 47
3.3.2 Results . . . 47
3.4 Particle filtering . . . 50
3.4.1 Implementation . . . 51
3.4.2 Results . . . 52
3.5 Robust Background Subtraction . . . 55

Introduction

An introduction to a report is supposed to set the tone and provide an outline of what is to come. Very well, here it is. We have been participants in the project cluster entitled *Computer Vision*, and this report describes our work for the past 12 weeks, a period of time that passed astonishingly quickly. We found ourselves needing, at the end, to halt production in its tracks; otherwise, the report could easily have blossomed beyond its present borders, covering topics far and wide, over the line at infinity.

Our work in the project cluster included attending lectures and completing three mandatory assignments. The three major sections of this report mirror the environs of these assignments (but do not confine them overly strictly). Each concept is treated in three parts: theory, implementation, and results.

Chapter 1 keeps to the realm of 2D, presents the concepts of homogeneous coordinates, as well as homographies and how to estimate them from manually selected point-pairs. We carry out a simple experiment that maps a moving figure in an image sequence onto a 2D blueprint of the scene. Finally, we stitch images together to form mosaics and tilings.

In Chapter 2 we enter the realm of 3D, moving between the world and the image by means of the projection matrix. We take a look at single-view metrology and calculate unknown heights of objects in an image. We study the fundamental matrix, epipolar lines, and camera calibration. And we examine the RANSAC robust estimator, which best-fits a line (or homography, or fundamental matrix) despite the presence of outliers.

Chapter 3 is about tracking. We add the time dimension to our images and investigate some of the basic probabilistic tracking models, namely Kalman and particle filtering and the constant-velocity model, and we experiment with person-tracking and eye-tracking.

In short, we have studied some basic elements of Computer Vision, and tried our hand at many of its more well-known algorithms. We have done as much as we could given this 12-week time frame, but we fully appreciate that the field of Computer Vision is vast, brimming with useful and fascinating applications.

One note to the reader, *Computer Vision* was originally intended to be a regular course at the IT University of Copenhagen, our university. Due to a paucity of student interest that we cannot fathom, the course was changed at the last moment to a so-called project cluster—for which lectures are fewer than for a regular course, and exercises are to be carried out independently in our small group settings, whereas regular course students might have benefitted from organized exercise sessions complete with lab instructor. We hope that the reader will look patiently on the discussions that follow and bear in mind that this written work is both a combination of what we have learned from textbooks, articles, and lectures, as well as a strong dose of our own subjective reasoning and speculation.

Chapter 1

Homographies, Mosaic, and Warping

1.1 Projective Geometry and Transformations of 2D

In this section, we briefly introduce the major concepts of 2D geometry used in computer vision and in our first assignment for this project. These concepts can also be generalized to 3D space. We begin with an explanation of homogeneous coordinates, given their ubiquitous use in computer vision, and move on to introduce homographies. We describe the results from our first assignment for this project: the implementation of a function that estimates the homography between a pair of images. We put this function to use for the second two parts of the assignment, where we use homography for merging images into a mosaic, displaying tracking data and affine rectification.

1.1.1 Homogeneous Coordinates

A point in a plane is usually represented by a coordinate pair, so we identify the plane with $\mathbb{R}^2$. A (non-vertical) line in a plane can be represented by its slope and intercept, so any line can also be said to have two degrees of freedom. Instead of representing points and planes by pairs of numbers, however, we will use homogeneous coordinates for points and lines, which makes the representation of certain geometrical entities very simple.

Using homogeneous coordinates, a 2D point $(x,y)^\top$ can be represented by $\mathbf{x} = (sx,sy,s)^\top$, where s is an arbitrary scaling factor. Line $ax+by+c=0$ is represented by $\mathbf{l} = (a,b,c)^\top$, which is also equivalent to $(s'a,s'b,s'c)^\top$ for any scaling factor s'. We can thus use the following simple rules (using the symbol $\times$ to represent the cross product):

1. Point $\mathbf{x}$ lies on line $\mathbf{l}$ if and only if $\mathbf{x}^\top\mathbf{l} = 0$.
2. The intersection of two lines $\mathbf{l}$ and $\mathbf{l}'$ is the point $\mathbf{x} = \mathbf{l}\times\mathbf{l}'$.
3. The line through any two points $\mathbf{x}$ and $\mathbf{x}'$ is the line $\mathbf{l} = \mathbf{x}\times\mathbf{x}'$.

Parallel lines therefore intersect in the points $(x,y,0)^\top$, which do not correspond to any finite points in the plane $\mathbb{R}^2$, but which can be added to the plane as *ideal* points (points at infinity). Ideal points lie on the *line at infinity*, represented by $(0,0,1)^\top$,

which we also add to the real plane. The resulting plane is called the *projective plane* $\mathbb{P}^2$.

1.1.2 A Model for the Projective Plane

If we look at the homogeneous representation for a point $(sx, sy, s)^\top$, $s \in \mathbb{R}$, as a collection of points in $\mathbb{R}^3$, we see that it forms a ray through the origin (see Fig. 1.1). Intersecting this ray with the plane $z = 1$, we again have a point in the plane. Ideal points $(x, y, 0)$ are rays parallel to the plane $z = 1$, and therefore do not intersect it. Similarly, if we look at the homogeneous representation of a line in $\mathbb{R}^3$, it corresponds to a plane passing through the origin. We can again obtain the line by intersecting this plane with $z = 1$. The line at infinity corresponds to a plane parallel to $z = 1$.

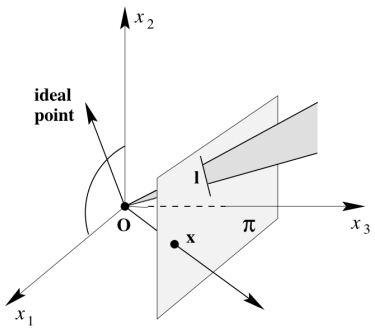

Figure 1.1: A model of the projective plane. Points and lines are represented by rays and planes, respectively, through the origin. Figure from Hartley and Zisserman [4].

1.1.3 Homography Matrix

Homogeneous coordinates allow for very elegant representation of projective transformations, also called *projectivities* or *homographies*. A homography is a transformation of $\mathbb{P}^2$ that maps straight lines to straight lines, and it can be represented simply as the multiplication of the homogeneous 3-vectors (representing the points) by a non-singular 3×3 matrix:

$$\begin{pmatrix} x_1' \\ x_2' \\ x_3' \end{pmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

or in matrix form, $\mathbf{x}' = \mathsf{H}\mathbf{x}$, where H is the *homography matrix*.

Matrix H is homogeneous—the transformation will not be changed if we multiply H with the arbitrary scaling factor. Additionally, matrix H is non-singular, so we can find an inverse mapping as $\mathbf{x} = \mathsf{H}^{-1}\mathbf{x}$. Another property of a homography is that under the point transformation $\mathbf{x}' = \mathsf{H}\mathbf{x}$, a line transforms as $\mathbf{l}' = \mathsf{H}^{-\top}\mathbf{x}$.

In their most general form, homographies will preserve only straight lines, but depending on the level of structure in H, some additional properties may be preserved—parallelism (affine transformations), ratios of lengths, angle (similarity transformations), length and area (Euclidian transformations).

1.1.4 Homographies in Images

The process of taking a photograph is modeled by a central projection of 3D world coordinates onto a 2D image coordinate system (as we will discus in Section 2.1). The central projection between two planes is a homography (it preserves straight lines), and therefore we will often use a homography when relating the planes in two images. For further detail on homographies and how they are estimated, please see Section 1.2, below.

In this report, we use two examples of projective transformations that arise in perspective images, both of which are illustrated in Figure 1.2.

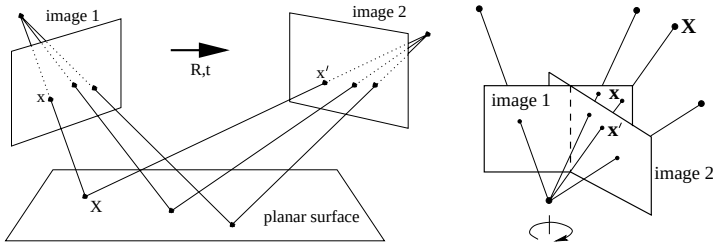

Figure 1.2: Homography examples. *Left:* A plane is captured by a camera from two different positions. The same three corresponding points are represented in each plane. *Right:* An object is photographed at different angles by a rotating but otherwise stationary camera. The corresponding points $\mathbf{x}$ and $\mathbf{x}'$ are represented in both Image1 and Image2.

On the left we have images of the same scene plane taken from different locations. The transformation between the images of the plane is a homography, because the acquisition of each image is a central projection (and therefore a homography for the given plane), and the concatenation of two homographies is still a homography. This is the setting we'll use for basic homography estimation (Section 1.2), an exercise that finds the homography of two different views of the same wall of lockers. We'll also use this setting for a sequence-to-map homography and warp-sequence exercise (Section 1.4 and 1.5), in which we map an image sequence to a 2D image. In this setting, the homography will only be valid for *one* world plane in the images at a time. If the object in the two images occupies more than one plane, for example, a wall *and* the floor, either one would need to use two homographies, one for each plane, or one would need to use the Fundamental Matrix, F, which we will discuss in section 2.4.

On the right in Figure 1.2, we have two images with the same camera center, with the camera rotating about its vertical axis. The corresponding points in two images are related by a central projection between two planes, which is a homography, so in this situation we can relate whole images with the homography. This is the situation at hand for composing an image mosaic, which we describe in Section 1.3.

1.2 Homography Estimation

1.2.1 Overview

As described in the preceding section, a homography is a projective transformation that can allow us to relate the features of two images. In two separate images, an object is captured from two different perspectives—an object such as a person, a landscape, a

set of points. With our eyes we can easily locate matching landmark points between the images. The homography is the interpretive tool the computer needs to perform this same task.

Our objective is now to implement the general homography estimation algorithm. For our assignment we were given two images of a wall of lockers (see Figure 1.3), captured from two different camera positions. The images provide a lot of corresponding points. Through manual corresponding-point selection, we were to calculate the homography for the two images such that, in the end, we could choose any point in one image and the function would return the corresponding point in the second image. The setting in this situation is as on the left-hand case of Figure 1.2, which means that the homography is limited to one plane, the front plane of the lockers.

The homography is a represented by a 3×3 matrix of numbers, which will be calculated on the basis of a few matching points (or lines) that we must first choose manually. We provide the first few examples, and the computer calculates the homography matrix, H, which relates them and all other corresponding points.

1.2.2 Direct Linear Transformation

We will use a simple linear algorithm, Direct Linear Transformation (DLT) for estimating the homography from point correspondences.

We know that for each pair of corresponding points $\mathbf{x} = (x, y, z)$ and $\mathbf{x}' = (x', y', z')$ we have $\mathsf{H}\mathbf{x} = \mathbf{x}'$. Having $\mathsf{H}\mathbf{x}$ and $\mathbf{x}'$ as (possibly different) homogeneous representation of the same point. Their cross product, therefore, must be 0, or

$$\mathbf{x}' \times \mathsf{H}\mathbf{x} = 0$$

This equation gives rise to three homogeneous linear equations in the elements of H, but only two of those are linearly independent. Those two independent equations can be written as follows:

$$\begin{bmatrix} \mathbf{0}^\top & -w'\mathbf{x}^\top & -y_i'\mathbf{x}^\top \\ w'\mathbf{x}^\top & 0^\top & -x'\mathbf{x}^\top \end{bmatrix} \begin{pmatrix} \mathbf{h}^1 \\ \mathbf{h}^2 \\ \mathbf{h}^3 \end{pmatrix} = 0$$

where $\mathbf{h}^{\mathbf{j}\top}$ are the rows of the matrix H.

There are 8 degrees of freedom for the homography, thus requiring a minimum of four non-collinear point pairs. Stacking the equations obtained from all available corresponding points, a homogeneous linear system is built, which can be written as

$$\mathsf{A}\mathbf{h} = 0$$

where A is a $2n \times 9$ matrix, the entries of which are obtained from the coordinates of n corresponding point pairs as explained in the two above equations, and $\mathbf{h}$ is a 9-vector of the elements of H.

Four point correspondences will lead to an exact solution of the system, while more than four point correspondences result in the over-determined system of equations. If the point coordinates of the over-determined system are noisy, there will not be an exact solution, but we can find the least-squares solution of the.

The solution to this system is found using Singular-Value Decomposition (SVD) of matrix A. The singular vector corresponding to the smallest singular value is the solution to $\mathbf{h}$. That is, if $\mathsf{A} = \mathsf{U}\mathsf{D}\mathsf{V}^\top$, the solution to the system is the column of V corresponding to the smallest entry in the diagonal matrix D.

1.2.3 Normalization

Hartley and Zisserman ([4], p. 108) emphasize,

> Data normalization is an *essential* step in the DLT algorithm. It must *not* be considered optional.

Normalization ensures that the result of the homography will be invariant with respect to scale and coordinate origin, along with improved accuracy of results. Of key benefit is that coordinate values will all be of similar magnitude. Without normalization, it is possible to have image coordinate values $(x,y,w)^\top$, where x and y can often be two or more orders of magnitude greater than w, which is usually 1. In later steps, multiplication with non-normalized coordinates will produce results that differ even more widely in magnitude, with xy, xx, and yy often being in the order of at least 10^4, compared with xw and yw, which would be on the order of 10^2, and ww products, which would be on the order of unity. This kind of divergence serves to amplify any errors brought on by the presence of noise in the images. Normalization eliminates these wide disparities and ensures a more accurate result.

The normalization scheme we used is the *isotropic* scaling: translating the point set to origin and scaling x and y coordinate equally, so that the average distance from the origin is equal. This results in point set where the average point is $(1,1,1)^\top$.

1.2.4 Implementation

The estimation of H comprises the following steps.

1. The user chooses at least four non-collinear matching point pairs.
2. Represent the 2D point coordinates using homogeneous coordinates. This is easily done by adding a third dimension to all points and setting that third dimension to 1 (see Section 1.1.1).
3. Normalize the point coordinates for each image independently. Normalization itself comprises two steps:
   (a) The points are centered the around the origin.
   (b) The points are scaled, so that the average distance between the point coordinates and the origin is equal to $\sqrt{2}$.

   The results of the normalization proces are two normalized sets of coordinates and two transformation matrices, T and T', one for each image.
4. Construct matrix A, as explained previously, using the normalized image coordinates.
5. Calculate the SVD of A. We construct a 3×3 matrix, $\tilde{\mathsf{H}}$, from the nine values in the last column of V. $\tilde{\mathsf{H}}$ is thereby the result of performing the direct linear transformation on the two sets of normalized coordinates.
6. Denormalization. We now derive the final homography H from $\tilde{\mathsf{H}}$ as well as the transformation matrices T and T', which are derived during the normalization process:

$$\mathsf{H} = \mathsf{T}'^{-1}\tilde{\mathsf{H}}\mathsf{T}$$

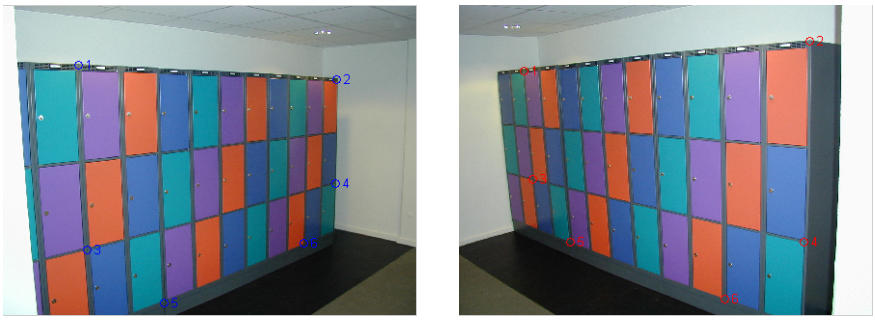

Figure 1.3: User-selected corresponding point pairs. The best choices for selection are corners or the small locks on the locker doors.

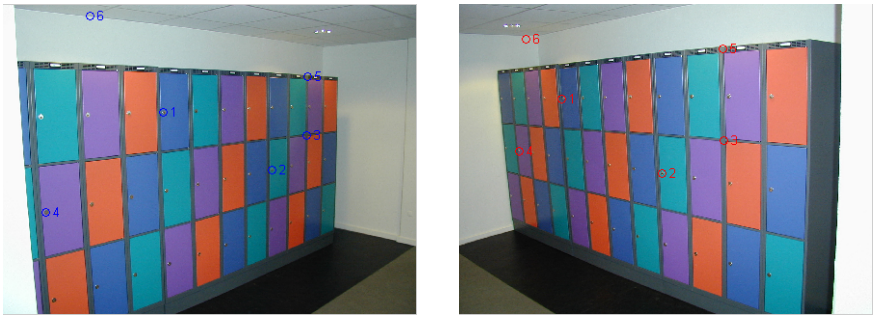

Figure 1.4: Testing the estimated homography. *Left:* The user clicks the image I at will to select points. *Right:* MATLAB uses the homography to find the matching points and plots them in I'. Note that Point 6, which is not on the same plane as the lockers, does not map correctly, though all other points are fairly accurate.

1.2.5 Results

After estimating our homography, we test it on the pair of images. First, we selected four point pairs (see Fig. 1.3) and computed the homography. Figure 1.4 shows the homography in action, as points are selected in the first image, I, and the corresponding points in the second image, I', is calculated and plotted automatically. Again, we note, the homography only applies to a single world plane at a time. The four point pairs must all be initially chosen from the wall of lockers, for example. One will then observe that when testing the homography by choosing a point on the ceiling, the homography will not return a properly matched point (e.g., point 6 in Fig. 1.4)

We note also that the accuracy of the homography depends heavily on the accuracy of the user hand-picking the original point pairs. One must take care to choose point correspondences that are 1) located on the same world plane, 2) spread around the whole image, and 3) as accurately placed as possible—obvious matching corners, for example. Picking more points will also improve the estimation of the homography.

To test the homography in the reverse order, that is, to find coordinating points in I given points from I', simply multiply the coordinates in I' with H^{-1}.

1.3 Image Mosaic

Here our task was to create a function that reads in a sequence of images, and then, based on their pairwise homographies, maps the images together into a single image, or mosaic (for a given base image). A number of options for interpolating and merging images are suggested for testing.

1.3.1 Overview

When a sequence of images is taken from a single location, the relation between images is a homography, as explained in Section 1.2 (see also Figure 1.2). By computing the homographies that relate the overlapping frames, we can handle the perspective transformation when going from one image to another. If we choose one of the images from a sequence as the base image, we can transform all the others to fit the same perspective. We can then "stitch" overlapping photos onto each other to form a single big image, such as a panorama of a scene.

In our implementation, the homography was estimated by first selecting point correspondences manually (by clicking in the image using the mouse) and thereafter computing the homography with the DLT algorithm as explained above. The images to be stitched onto the reference image were then transformed (warped) according to the homography, using back-mapping and two different interpolation settings. As for pasting the images into the corresponding positions, we tried both simple averaging and the 'hat function" technique.

1.3.2 Backward Mapping

Having to warp an image I onto the base image I_b, and having the homography H from the image to the base, it is feasible to visit all the pixels in I and transform them to the base according to the homography. This approach doesn't guarantee a good result, as some pixels in the base reference frame might not be evaluated; additionally, it doesn't allow for any interpolation. Instead, all warping is done using *backward mapping*—visiting all the pixels in the base reference frame and checking which pixels would be transformed to the position according to the inverse homography, H^{-1}.

Using this approach, we first needed to forwards-map the corners of the image I to determine the final size of the composite image, as well as the region of the composite image that is covered by I, so that our back-mapping wouldn't end up outside of image I.

1.3.3 Bilinear Interpolation

Another thing to consider when using backward mapping is interpolation. When we look back from a pixel position in the base image, we get non-integer values in the image I. In these cases, the most straightforward solution is to do simple rounding and adopt the nearest-neighbor pixel value, but the resulting image will be jagged.

A better solution is to use bilinear interpolation and find the blending ratio from the pixel values of the 4 nearest pixels, where the different pixels are weighted more the closer they are to the location we want to interpolate. Each intensity value of the RGB color image is computed as follows:

$$I(x,y) \approx (1-b,b) \begin{bmatrix} I(\lfloor x \rfloor, \lfloor y \rfloor) & I(\lceil x \rceil, \lfloor y \rfloor) \\ I(\lfloor x \rfloor, \lceil y \rceil) & I(\lceil x \rceil, \lceil y \rceil) \end{bmatrix} \begin{pmatrix} 1-a \\ a \end{pmatrix}$$

where x and y are non-integer local image coordinates, and

$$a = x - \lfloor x \rfloor$$

$$b = y - \lfloor y \rfloor$$

are the corresponding weight values in horizontal and vertical directions. The improvement achieved using bilinear interpolation, and its cost in terms of computing time are illustrated in Figure 1.6.

1.3.4 Hat-Weighting Function

When blending two images, the question arises of how to deal with transition boundaries—as certain areas of the base image will always be overlapped by another image, sometimes more than one, and the transitions between the two are often undesirably obvious. Initially, we used a simple averaging function for the overlapping parts of the image and rounding for interpolation.

We then augmented the blending method by incorporating a *hat function* to avoid transition boundaries. The idea is that we want to blend two images gradually from one to another. Therefore, when the two images are stitched, overlapping pixels will be weighted depending on their distance to the image center. In other words, a pixel "weighs" more the closer it is to the center of its image frame. In this way, a smoother transition is achieved. The weights used were linear in each direction and ranged from 0 at the image edges to 1 in the center, as shown in Figure 1.5), while in Figure 1.6 we show the improvement achieved by using the hat function.

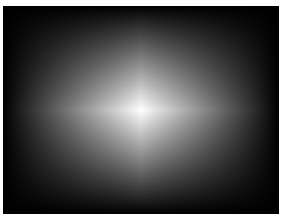

Figure 1.5: The hat-weighting function of an image area of 640x320 pixels.

1.3.5 Implementation

The following describes the steps we took to stitch two images into a mosaic. For a mosaic consisting of more than two images, these steps would need to be repeated for each subsequent image to be stitched.

1. The user selects point correspondences manually. A minimum of four point correspondences are needed to estimate the homography.

2. The homography is estimated using the normalized DLT algorithm.

3. Hat weights are computed for both images.

4. The corners of the image to be warped are forward-mapped to the base image, the sizes and position of the two images are evaluated, and the base image is

Figure 1.6: Comparison of stitching variants. *Row 1:* The original three images to be stitched. *Row 2:* Rounding used for interpolation, and averaging for the overlapping parts. *Row 3:* Rounding used for interpolation, and hat-function weights for the overlapping parts. *Row 4:* Bilinear interpolation used for interpolation, and hat-function weights for the overlapping parts. *Right:* Detail of images. The computation time for the three methods: 13.8 seconds, 14 seconds, and 73 seconds on the same machine.

Figure 1.7: Mosaic of four images, uncropped.

zero-padded to accommodate the new image. The homography is adjusted accordingly.

5. The new image is warped onto the base image using backward-mapping and bilinear interpolation. In the region where the two images overlap, the hat-weighting function is used to determine the blending ratio.

1.3.6 Results

Figure 1.6 compares the various stitching techniques that we tried: 1) rounding for interpolation with averaging for overlap; 2) rounding for interpolation, and hat-function weights for overlap; and 3) bilinear interpolation for interpolation, and hat-function weights for overlap.

The first method did not incorporate the hat function, and resulted in quite visible transition borders (row 2 of Fig. 1.6). The other two methods did utilize the hat-weighting function (rows 3 and 4 in Fig. 1.6), and indeed border transitions are no longer apparent. However, when simple rounding is used instead of bilinear interpolation, the edges of objects (such as the eaves of the roof in row 3) often appear jagged. Jagged edges are no longer evident when hat and bilinear interpolation are used together (see detail in row 4, right).

The computation time for the three methods was as follows: 13.8 seconds, 14 seconds, and 73 seconds on the same machine, respectively. We note that use of both hat and bilinear interpolation results in significantly longer computation time as compared with the other two methods, but for images of this size, bilinear interpolation with hat seems well worth the extra investment in time.

We note that, when stitching panoramic images together, the mosaic takes a characteristic bow-tie shape, an effect of the shift in camera angle relative to the scene, that results in the apparent flaring of the images as they get farther and farther from the center image. When presenting stitched images in a mosaic, one must decide whether to preserve the black border or crop, and this will always depend on the context and the

Figure 1.8: Tiling made of 9 images of the ITU building. Size of the small images 640×480; size of the final image mosaic 1627×971. Note obvious break in the stairwell, far left, which is a result of flawed image capture.

expected illustrative purpose of the mosaic. For Figure 1.7, we have retained the black border instead of cropping.

In addition to horizontally stitched mosaics, we also tried our hand at tiling images together. The mosaic shown in Figure 1.8 is the result of tiling nine images of the ITU (which we took ourselves). We first merged images vertically, producing three rows, and then merged those into a completed mosaic. To deal with the black edges, we changed the original implementation of the mosaic function slightly, which allowed for a mistake to creep in—the staircase at the far left reveals a very slight break in continuity. This was caused by the photographer not tiling the scene perfectly; the images in the left column are not vertically aligned, so the staircase is missing from the middle photograph. The stitching of the stairs, therefore, was forced to occur between the bottom-left and top-left image (i.e., non-neighboring images), and the homography (calculated from two homographies estimated between neighboring images) was not robust enough to span the gap.

1.3.7 Automatic Stitching

As an aside to the mandatory assignments, we experimented with the concept of automated stitching. The reader should take note that we are not offering a fully fledged implementation here, but simply presenting what we tried and observed. Using a video sequence panning over a scene, we developed the following suggested implementation:

1. Extract points from each image using a Harris corner detector.[1]
2. From this collection of points, find the putative correspondences from image to image by cross-correlation matching:
   - Sample a 7×7 area around each point and correlate with the 7×7 neighborhood around all the points in the other image.
   - Use the point with the highest correlation coefficient as the putative corresponding point.
3. Eliminate outliers using RANSAC. (We did not implement the final two steps; more detail on the RANSAC method in Section 2.5):
   - Build a model H for a sample of 4 putative correspondences using the DLT algorithm.
   - Find support for the model by counting the number of putative correspondences that are inliers. A putative correspondence $(\mathbf{m}, \mathbf{m}')$ is an inlier if $\|\mathbf{m}'\mathsf{H} - \mathbf{m}\| < t$, where the estimated threshold t is based on a table in Hartley and Zisserman ([4], p. 119).
   - Iterate until a suitable model is found.
   - Rebuild the homography from the new set of inliers.

Figure 1.9 illustrates the output of Harris corner detection. Figure 1.10 shows the set of putative correspondences connected with lines, so that inliers (horizontal lines) and outliers (non-horizontal lines) can be clearly distinguished.

[1] Harris corner detection is a filter detecting the maximum gradient variation for all directions. We used Peter Kovesi's Matlab implementation of the Harris corner detector [5].

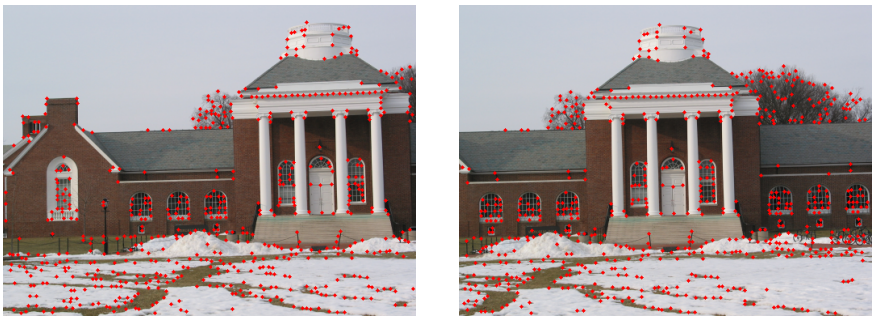

Figure 1.9: Points output from Harris corner detection on two overlapping images.

Figure 1.10: Putative matching points connected by lines. Points are from Fig. 1.9, and include those that gave the highest cross-correlation match. Outliers (non-horizontal lines) and inliers (horizontal or near-horizontal lines) can be easily distinguished.

We were hoping to be able to implement a fully automatic stitching algorithm that would take any sequence of images and put them together into one mosaic, with no intervention from the user. This approach would have had a few more problems than the difficulties outlined by Hartley and Zisserman for their automatic homography computation [4]. We cannot know the ratio of outliers/inliers of our points computed from neither the corner detector nor the cross-correlation, so we have to assume a 50% risk. Further more, it may even be higher. Also, the threshold for the Harris corner detector would have to be set manually from an estimation of how many points would be needed. For a fully automatic procedure, we suggest analyzing the images for the amount of variation and use that analysis for setting the corner-detector threshold.

We experienced that several points are repeatedly counted as matches when cross-correlating, which means that points are assigned to more correspondences than one—this should be prevented by enforcing a tighter constraint to the “winner takes it all” procedure Hartley and Zisserman use. But as we were not intended to do this for our assignment, we had to move on with a slight shrug.

1.4 Sequence-to-Map Homography

For this section, we have made a component that could be used in a tracking system. We are given an image sequence, captured by a still (surveillance) camera, in which a person is seen walking on the ground floor of the ITU building. We are also given the corresponding tracking data obtained by segmenting the person in the sequence. Additionally, we were given a 2D structural map of the ITU's ground floor. The task was to display the tracking data from the image sequence on the 2D map.

1.4.1 Implementation

The transformation from the ITU building's ground floor to the map is a scaling, so it is surely a homography. Additionally, the transformation of the ground floor plane to the camera's image plane is a central projection, which is also a homography. (This can be verified by the fact that straight lines remain straight.) Because the concatenation of two homographies produces another homography, we can confirm that it is a homograply that defines this transformation from the imaged ground floor to the map. In short, we are presented with a situation similar to that sketched in Figure 1.2, left.

To solve this task we first inspected the given data. We decided to crop the map of the ground floor so that it covered approximately the same area of the ground floor as seen in the image sequence.

The tracking data consisted of three bounding rectangles: one containing the upper body of the person, one containing the legs only, and one containing the whole figure, as illustrated in Figure 1.11. We decided to track the mid-point of the low side of the whole-figure rectangle, since that point is actually *on* the ground floor, directly under the person's center.

Implementation then proceeded according to the following steps:

1. The user manually selects corresponding points from the map and one of the image sequence frames.
2. The homography between the image sequence and the map is estimated using the DLT algorithm.
3. A set of points (one point per frame) is extracted from the tracking data.
4. Tracking points are transformed using the estimated homography.
5. The original sequence is displayed in parallel with the display of tracking data on the map (Fig. 1.11, bottom row).
6. A plot of the complete path of the walker is produced on the map (Fig. 1.12).

1.4.2 Results

When observing the data on the map, a number of additional pieces of information can be obtained that cannot be directly estimated from the image sequence. For instance, Figure 1.12 depicts the actual shape of the walker's path, with all angles and directions preserved. If we know the measurements of the ITU building from the map, we can make a scaling of the tracking data to derive information about the position of the person on the ground floor. Additionally, from observing the tracking data on the map over time, we could estimate the velocity of the person.

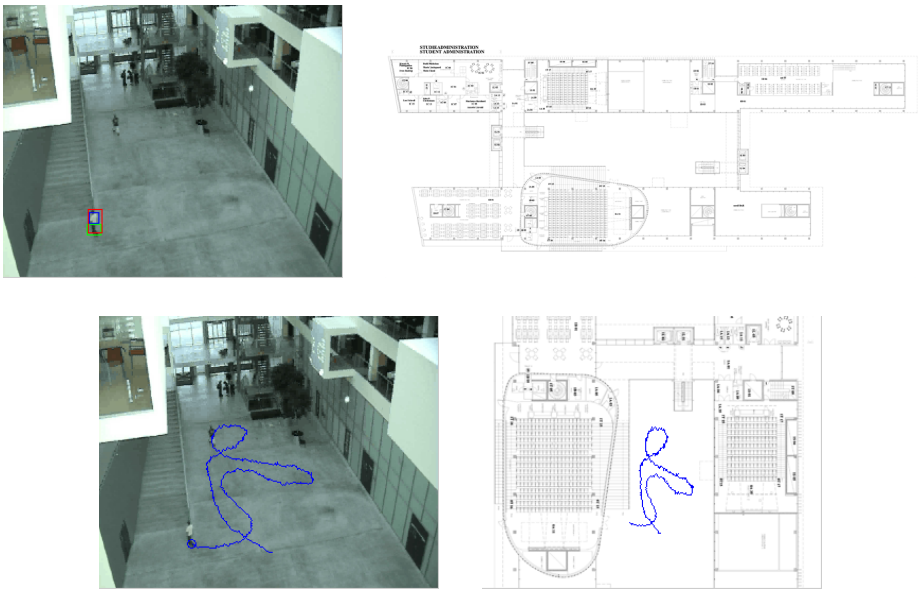

Figure 1.11: *Top row:* The input. *Left,* one frame from the image sequence, with all tracking data superimposed. *Right,* the original map of the ground floor. *Bottom row:* The output. *Left,* a frame from the image sequence with the tracking point plotted (corresponding to the center of the red rectangle's lower edge), and *Right,* the partial path corresponding to the frame displayed on map.

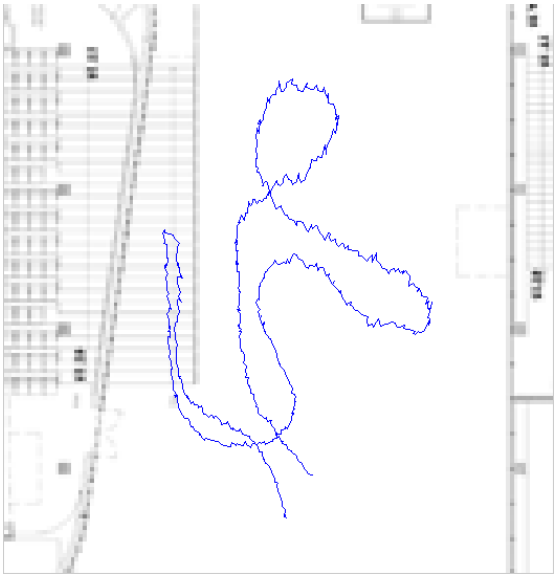

Figure 1.12: The complete path of the walker, plotted atop a portion of the 2D map. The slightly erratic nature of the path is caused by segmentation noise, not the homography.

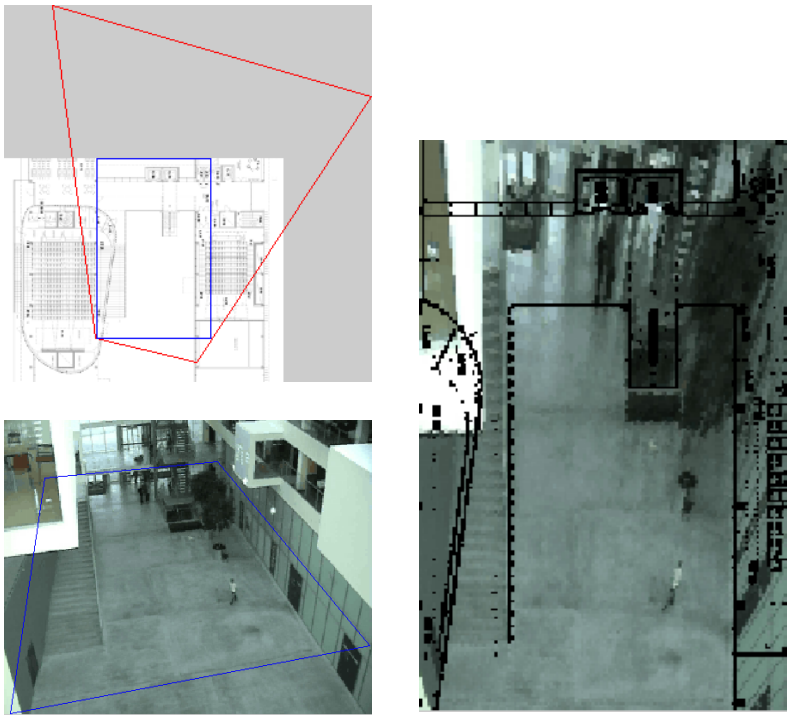

Figure 1.13: *Top-left:* The 2D map, with the part covered by original image sequence shown outlined in red, and the part covered by the final sequence outlined in blue. *Lower-left:* A single frame from the original image sequence, with the parts covering the final sequence outlined in blue. *Lower-right:* A frame from the final sequence obtained by warping the original sequence, with the outline of the map superimposed.

The accuracy of the mapping depends fully on the accuracy of the tracking data and the estimated homography. Tracking data are sometimes "shaky", resulting in a somewhat noisy-looking path. It is possible that the appearance of the path could be improved by smoothing the tracking data, which we indeed try later in this report using the Kalman algorithm (Section 3.2), which finds the likeliest path given noisy data.

It is difficult to assess the accuracy of the homography estimation: it looks rather good, but could likely be improved if we were able to find more point correspondences. It is important to note that the homography mapping is accurate only while the person is walking on the ground plane. When the walker ascends the stairs, he leaves the ground plane—the result of which is that the homography becomes more and more inaccurate as the distance between him and the ground floor increases.

1.5 Warp Sequence

This an extension of the sequence-to map task from the previous section. It involved making a function that would take the following input: an image sequence from a fixed camera, a 2D map, and the homography between the image sequence and the map. (The same image sequence and 2D map from Section 1.4 were used.) The function should then warp every image from the sequence onto the map, creating a new sequence.

1.5.1 Implementation

The given homography maps only the ground plane to the 2D map, so only those parts of the images in the sequence occupied by the ground floor itself will be correctly mapped. Meanwhile, only a fraction of the map is visible in the sequence. Therefore, the first and the biggest challenge was to decide on the dimensions of the final image and its location in the map. The second challenge was the implementation of the warping itself.

The part of the map visible in the sequence is shown in Figure 1.13 (top-left, red quadrilateral). The dimensions that we chose to use for the final sequence is marked by the blue rectangle in the same image. In the lower-left image, the part of the original sequence that is to cover our chosen frame is outlined.

Finally, after changing the coordinates, the homography from the final to the original sequences needed to be determined. Having all the dimensions and the homographies between them, we could start warping each image from the image sequence. We used backwards mapping, and because the original sequence covers the whole of the final one, we didn't need to concern ourselves with staying within the dimensions of the images. Simple rounding was used instead of some better interpolation method. After warping each image from the sequence, the outline of the map was printed on top of it. One image from the resulting sequence is shown in Figure 1.13 (right).

1.6 Affine Rectification

For this task, we were asked to make a function that performs an affine rectification of an image, as follows: 1) Calculate the vanishing line $\mathbf{l}$ of a plane in an image and form the projective transformation H that takes $\mathbf{l}$ to the line at infinity; 2) Affinely rectify the image according to H using backwards mapping; and 3) Compare the images both before and after affine rectification.

1.6.1 Theory

For a given imaged plane, we can construct a transformation that will recover the affine properties of that plane. This is called affine rectification.

The important invariant for affine transformation is *parallelism*. Therefore, one of the objectives of affine rectification is to recover parallelism in the chosen plane. To do that, a transformation is needed that will push the identified line at infinity of a chosen plane to its canonical position. In other words, we are searching for a transformation H that will map the line at infinity $\mathbf{l} = (l_1, l_2, l_3)^\top$ to $(0,0,1)^\top$. Or, written in matrix form, we are looking for the 3×3 matrix H, such that

$$\mathsf{H}^{-\top}(l_1, l_2, l_3)^\top = (0,0,1)^\top$$

It can easily be shown that the matrix

$$\mathsf{H} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ l_1 & l_2 & l_3 \end{bmatrix}$$

meets this criterion, but it is not the only matrix with this desired property. *Any* homography matrix with $\mathbf{l}^\top$ in its bottom row will map the line $\mathbf{l}$ to $(0,0,1)^\top$. We can say that the affine rectification is defined up to an arbitrary affine transformation.

Having affinely rectified an image, one recovers affine properties for a given plane and all planes parallel to it. Still, properties that are not invariant under affine transformations, such as angles, will not be recovered. It is possible to go one step further and perform the metric rectification of an image by finding a transformation that maps imaged circular points to their canonical positions. The additional constraints required for metric rectification can, for example, be obtained from two sets of orthogonal lines.

1.6.2 Implementation

We performed affine rectification on an image using the following steps, the results of which are depicted in Figure 1.14:

1. The user defines a plane by selecting two sets of imaged parallel lines.
2. The line at infinity for a given plane is calculated, and an initial rectification transformation matrix is constructed.
3. The corners of the original image are transformed to estimate the size and position of the rectified image. Scaling is determined, such that the rectified image has roughly the same size as the original. This scaling and initial rectification comprise the final rectification transformation.
4. The final rectified image is created using backwards mapping and simple rounding (for efficiency).

1.6.3 Results

Figure 1.14 shows two affine rectifications that we performed on a single image, selecting first one and then another imaged plane. As expected, the windows on a selected side of the building are transformed to parallelograms. Note how parallel lines in the original image, i.e., those that intersect at a chosen vanishing line, are transformed into true parallel lines after affine rectification, depending on the selected plane.

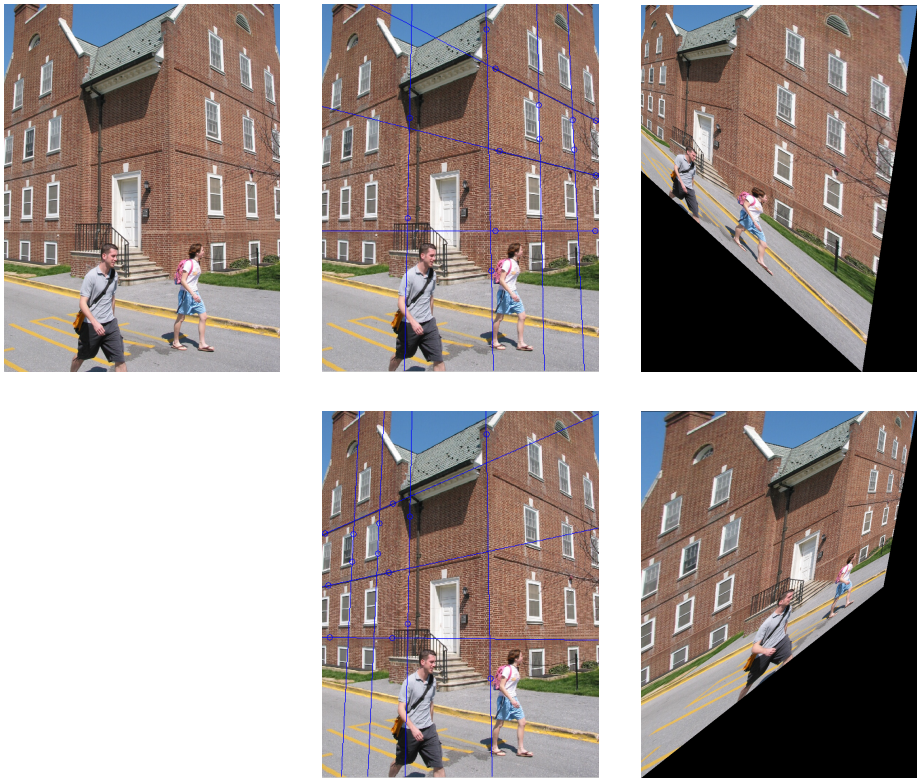

Figure 1.14: Two affine rectifications of a single image. *Top-left:* The original image. *Center column:* The selected planes. *Right column:* Affine rectification of the left and right sides of the building in the original image.

Chapter 2

Single-View Geometry, Stereo, and RANSAC

In this chapter, we study the concepts of single- and double-view geometry, providing a brief description of the *projection matrix* and the *fundamental matrix*. We look at the RANSAC method for finding the most probable linear solution given measurements of inliers and outliers. Lastly, we touch upon camera calibration.

2.1 Projection Matrix

The process of taking a photograph is, in short, a mapping of the 3D world onto a 2D surface, the image. We can analyze this mapping by developing a camera model, which typically constitutes a simple central projection.

For a convenient reference frame, we introduce a coordinate system whose origin lies at the camera center, with the *xy* plane parallel to the image plane, and the *z*-axis perpendicular to the image plane, as illustrated in Figure 2.1. The *z*-axis, also called the *principal axis*, intersects the image plane in the aptly named *principal point*. The distance f between the camera center and the image plane is called the *focal distance*.

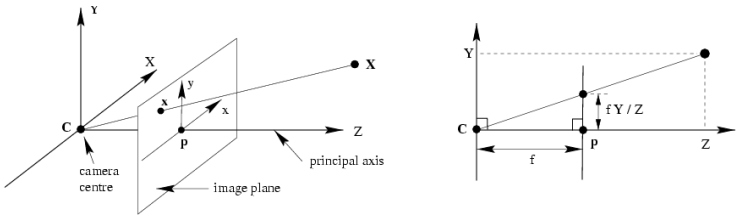

Figure 2.1: World coordinate **X** is projected onto the image plane at point **x**. The focal length f is the distance between the camera center and the image plane along the *z*-axis. Figure from Hartley and Zissermann [4].

If we now represent the 3D world coordinates by the homogeneous vector $\mathbf{X} = (X,Y,Z,1)^\top$, and the 2D image coordinates (ignoring for now the coordinate in the *z*-direction, $z = f$) by $\mathbf{x} = (x,y,1)^\top$, the central projection around the origin can be

expressed in terms of matrix multiplication:

$$\mathbf{x} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \mathbf{X}$$

To allow for an arbitrary coordinate origin in the image plane, (in practice, usually one of the corners of the image), we generalize by including a translation. The expression becomes

$$\mathbf{x} = \begin{bmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \mathbf{X}$$

where $(p_x, p_y)^\top$ are the (image) coordinates of the principal point.

The matrix

$$\mathsf{K} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix}$$

is referred to as the *camera calibration* matrix, and its entities are *internal* camera parameters. We can now write simply

$$\mathbf{x} = \mathsf{K}[\mathrm{I}\,|\,\mathbf{0}]\mathbf{X}$$

In Section 2.6, we touch upon the camera calibration matrix, and what is gained if the internal camera parameters are known.

It is generally undesirable to have a coordinate system that is defined by the position and orientation of the camera (which can also change from image to image). Instead we would rather use the *world* coordinate frame. The two coordinate frames are connected by a translation and a rotation, which lead to the expression

$$\mathbf{x} = \mathsf{K}[\mathsf{R}\,|\,\mathbf{t}]\mathbf{X}$$

where R is a 3×3 rotation matrix, and $\mathbf{t}$ is a 3×1 translation vector, while the coordinates $\mathbf{X}$ are now expressed in the world coordinate frame.

The 3×4 matrix

$$\mathsf{P} = \mathsf{K}[\mathsf{R}\,|\,\mathbf{t}]$$

is called the *projection* matrix, and parameters R and $\mathbf{t}$, which relate the camera's orientation and position to the world coordinates, are the *external* parameters of the camera. This is the model we will use, and in this model it is the projection matrix that relates the 3D world coordinates to the image coordinates.

An even more general expression can be obtained when introducing non-square pixels for charge-coupled device (CCD) cameras and a skew.

2.2 Single View Metrology

In their seminal paper, "Single View Metrology," Antonio Criminisi et al. [1] describe how aspects of the affine 3D geometry of a scene may be measured from a single perspective image without knowledge of the camera's internal calibration or position. It is, however, assumed that some geometric information can be determined from the image: the vanishing line of a reference plane, and a vanishing point for reference direction.

The following measurements can be extracted from an image: 1) distances between planes parallel to the reference plane (up to a common scale factor), 2) the area and length ratios on a plane parallel to the reference plane, 3) the camera's location. Criminisi et al. describe two approaches: a geometric approach, which is based on cross-ratios, and an algebraic approach, which (among other advantages) allows for metric calibration from multiple references.

In this assignment, we calculated distances between parallel planes using both the geometric and algebraic methods.

2.2.1 Preliminaries

To be able to use the methods described by Criminisi et al. [1], we needed first to determine the vanishing line of a reference plane, as well as a vanishing point for the reference direction, which could be any direction not parallel to the reference plane.

The reference plane in a scene is typically the ground plane, in which case its vanishing line is the horizon. The reference direction, usually orthogonal to the reference plane, is therefore usually the vertical direction. To simplify the discussion from here, we'll stick to the expressions ground plane, horizon, and vertical direction. With this setup, the computation of various distances between planes parallel to the reference plane is simply the computation of the heights of certain objects in the scene.

To obtain the horizon, we first determined the two vanishing points for the two sets of imaged parallel lines in the directions that define the ground plane. We then constructed the horizon as the line passing through these vanishing points. For the vertical direction, we needed a set of imaged vertical parallel lines. All of these lines could be rather easily obtained from images depicting structures such as walls, windows, or stairs.

In Figure 2.4 we see an example image showing the sets of user-selected lines, which have been plotted in blue. The horizon calculated on the basis of these lines is shown plotted in red. We also have an image where we can see the position of the distant vanishing points in relation to the finite image.

2.2.2 Geometric Representation

Our first step was determining the relationship between the heights of objects in the scene and the camera height. From the basic geometry of the camera illustrated in Figure 2.2, it is clear that the horizon is the intersection of the image plane with a plane parallel to the reference plane and passing through the camera center. This means that

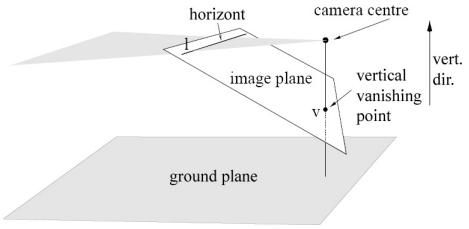

Figure 2.2: Basic geometry given a camera in a certain location relative to the ground plane and the vertical direction. Image taken from Criminisi et al. (with modifications [1]).

any scene point projected onto the horizon is at the same height (distance from the ground plane) as the camera was when the image was made.

If there is a vertically oriented object on the ground plane in the scene, its base point $\mathbf{X}_b$ and top point $\mathbf{X}_t$ are projected onto the image points $\mathbf{x}_b$ and $\mathbf{x}_t$, respectively, which are collinear with the vertical vanishing point $\mathbf{v}$ (see also Fig. 2.3). The line joining these three points intersects with the horizon $\mathbf{l}$ in a single point $\mathbf{c}$, which back-projects onto the scene point $\mathbf{C}$, which is at camera height.

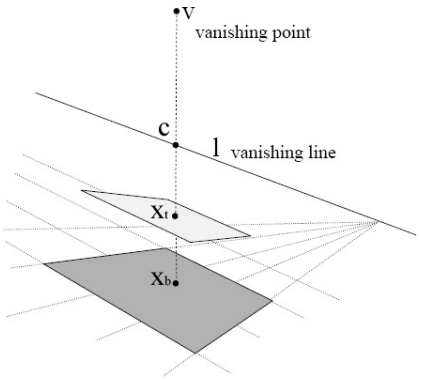

Figure 2.3: The height of the top plane relative to the camera height as viewed in the image. Image from Criminisi et al. ([1]).

For the situation illustrated in Figure 2.3, the four points $\mathbf{x}_b$, $\mathbf{x}_t$, $\mathbf{c}$, and $\mathbf{v}$ define a cross-ratio

$$\frac{d(\mathbf{x}_b,\mathbf{c})d(\mathbf{x}_t,\mathbf{v})}{d(\mathbf{x}_t,\mathbf{c})d(\mathbf{x}_b,\mathbf{v})} = \frac{d(\mathbf{X}_b,\mathbf{C})d(\mathbf{X}_t,\mathbf{V})}{d(\mathbf{X}_t,\mathbf{C})d(\mathbf{X}_b,\mathbf{V})}$$

which is basic invariance under projectivity.

Using the fact that $\mathbf{V}$ is a point at infinity, and defining camera height Z_c and object height Z, this reduces to

$$\frac{Z}{Z_c} = 1 - \frac{d(\mathbf{x}_t,\mathbf{c})d(\mathbf{x}_b,\mathbf{v})}{d(\mathbf{x}_b,\mathbf{c})d(\mathbf{x}_t,\mathbf{v})}$$

Note that the signs in this expression depend on the ordering of the cross-ratio, so one should take care or use signed distances.

The ordering ambiguity of the cross-ratio can be avoided using an alternative approach (described in Hartley and Zissermann [4], p. 220–222). We can look at the line containing the four points $\mathbf{x}_b$, $\mathbf{x}_t$, $\mathbf{c}$, and $\mathbf{v}$ as a 1D affine rectification problem. It is enough to find the homography that will push the point $\mathbf{v}$ to its position at infinity; the affine properties of the line (length ratios) are thereby restored.

Having established the connection between the scene height Z and the camera height Z_c, it is enough to have one known reference height (either scene or camera) to compute all other scene heights using Z_c as a link.

2.2.3 Algebraic Representation

For algebraic representation of single view metrology, a scene coordinate system is defined with x- and y-axes spanning the ground plane, and the z-axis representing the

vertical direction. This choice of the coordinate frame constrains the parametrization of the 3×4 projection matrix P as

$$\mathsf{P} = [\mathbf{v}_x \quad \mathbf{v}_y \quad \alpha\mathbf{v} \quad \bar{\mathbf{l}}]$$

where $\mathbf{v}_x$ and $\mathbf{v}_y$ are (unknown) vanishing points in the x- and y- directions, $\mathbf{v}$ is the vertical vanishing point, $\bar{\mathbf{l}} = \mathbf{l}/\|\mathbf{l}\|$ is the (normalized) horizon, and α is a scale factor. Only the last two columns of P have an influence on the measurements of heights.

Our base and top points are now represented as $\mathbf{X}_b = (X, Y, 0, 1)^\top$ and $\mathbf{X}_t = (X, Y, Z, 1)^\top$, and their images are $\mathbf{x}_b = \mathsf{P}\mathbf{X}_b$ and $\mathbf{x}_t = \mathsf{P}\mathbf{X}_t$, respectively. Exploiting various properties of the projection matrix, it is possible to obtain the relation

$$\alpha Z = -\frac{\|\mathbf{x}_b \times \mathbf{x}_t\|}{(\bar{\mathbf{l}} \cdot \mathbf{x}_b)\|\mathbf{v} \times \mathbf{x}_t\|}$$

which gives the connection between the scene heights and the scale factor, because the values on the right side are calculated from the image. With one known reference height, one can calculate α, which can then be used for calculating other scene heights.

If more than one reference distance is known, then an estimate of α can be derived from an error minimization algorithm, stacking the linear equations in a matrix, and finding the solution by means of single-value decomposition. This is refered to as metric calibration.

2.2.4 Implementation

We implemented both the geometric and the algebraic approaches. Since the algebraic approach allows for metric calibration, that was our preferred method and is described here.

- The user defines the ground plane and the vertical direction by selecting lines in the image.
- The user selects reference heights (known heights) by clicking the line segments on the image and inputing the known measurements. To ensure that the selected line segments are vertical, after the user clicks the top point $\mathbf{x}_t$, the line connecting $\mathbf{x}_t$ and the vertical vanishing point $\mathbf{v}$ is plotted. The user should then click the base point $\mathbf{x}_b$ on the plotted line.
- The user selects unknown heights in a similar way. (See Fig. 2.4, right.)
- The scale factor α is calculated from the metric calibration algorithm.
- Unknown heights are computed.
- A new figure is generated, depicting the reference heights and calculated heights superimposed on the image.

2.2.5 Results

In Figure 2.5, we display some of the obtained measures. The top-left image is an example of scene heights obtained from *one* reference height. In the top-right image, three reference heights were used for metric calibration. In both images, the estimated door height falls in the range 297–203 cm. The estimated chair height varies

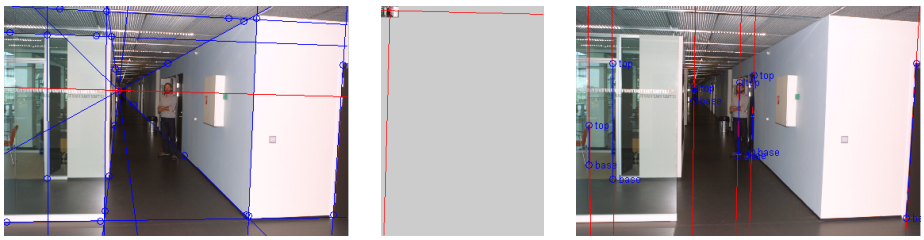

Figure 2.4: Preliminaries. *Left:* Sets of user-selected reference lines, shown in blue. The user-selected lines are used to calculate the vertical vanishing point and the horizon (shown in red). *Center:* Depiction of the position of the distant vanishing points in relation to the finite image. *Right:* Plot of top and base points and height representations (blue) of objects in the scene (cf. Fig. 2.6).

Figure 2.5: SVM results. *Top left:* Four heights that were calculated from a single reference height. *Top-right:* Three reference heights used for metric calibration. *Bottom row:* Further examples of height estimations. Reference heights are shown in blue, calculated heights are shown in green, and the horizon is given in red. The camera height is given in green, and can be found near the middles of the horizon lines.

Figure 2.6: Comparison of and algebraic (middle column) and geometric (right column) method. *Top row:* Well chosen top and base points. *Middle and bottom rows:* Poorly chosen top and base points.

much more; because the chair has no vertical edges, it is difficult to choose well-corresponding base points for any given top point.

The measurements shown in Figure 2.5, bottom-row middle and bottom right, were obtained using the same settings: the same horizon and vertical vanishing point. Among other distances, we estimated the heights of the metal tree cylinders (which we knew to be 90 cm tall); these estimations fell in the range of 88–91 cm. The estimated height of the woman varies as much as 6 cm, again due to the difficulties of having to choose corresponding top and base points.

Comparing the geometric and algebraic methods, we often noticed rather large fluctuations between consecutive estimations using the same method, but we also noticed significant differences between the results obtained from the two methods. This was most evident in one of the images and some of our observations are shown in Figure 2.6. In the first row, we have an example of well chosen corresponding top and base points (collinear with $\mathbf{v}$), and in that case the results obtained by the two methods were very similar to each other. In the middle and bottom rows, however, we see the results of poorly chosen top and base points (*not* collinear with $\mathbf{v}$), and the results from two methods differed greatly. Clearly, these are poorly defined problems and we cannot expect correct results. However, we did observe that the two methods each handle this difficulty in ways that are very different from each other. The algebraic method seems to be more sensitive to this type of problem, whereas the geometric method seems more

robust. It would therefore be advisable to enforce the collinearity of the points $\mathbf{x}_t$, $\mathbf{x}_b$, and $\mathbf{v}$ as an integral part of the implementation. This could be done, for example, by fitting a line through $\mathbf{v}$ that minimizes the sum of square distances from $\mathbf{x}_t$ and $\mathbf{x}_b$. Note, however, that this recommendation is based on a very limited number of observations.

2.3 Fundamental Matrix

When images of the same object are taken from two different views, the resulting situation is illustrated in Figure 2.7. A point $\mathbf{x}$ from the image I is back-projected as a ray through the camera center $\mathbf{C}$. In another image I', this ray projects to a line $\mathbf{l}'$. The line $\mathbf{l}'$ is called the *epipolar* line for $\mathbf{x}$, and it is on this line that we must search for point $\mathbf{x}'$.

Regardless of which point $\mathbf{x}$ we choose to start with in the image I, its epipolar line in another image will always contain the image of the camera center $\mathbf{C}$. Hence, all epipolar lines intersect at a single point called the *epipole*.

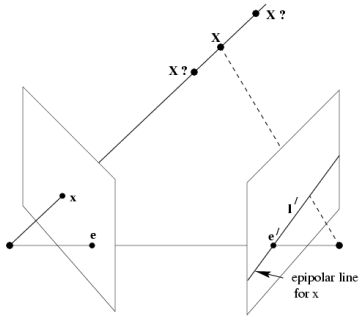

Figure 2.7: A point $\mathbf{x}$ in the image to the left becomes a line $\mathbf{l}'$ in the image on the right. Figure from Hartley and Zisserman [4].

The geometry between these two views is independent of scene structure, depending only upon the internal parameters of two cameras and their relative positions. However, we can find the relation between two images without knowing those parameters explicitly.

Aligning the world coordinates with the coordinate frame of the second view, the projection of the world points $\mathbf{X}$ onto images I and I' can be expressed as

$$\mathbf{x} = \mathsf{K}[\mathsf{R}|\mathbf{t}]\mathbf{X}$$

$$\mathbf{x}' = \mathsf{K}'[\mathrm{I}|\mathbf{0}]\mathbf{X}$$

where the rotation matrix R and the translation vector $\mathbf{t}$ relate the orientations and positions of the two cameras. Introducing a skew-symmetric matrix T, which defines the cross product with $\mathbf{t}$, and using various properties of symmetric and skew-symmetric matrices, a relation

$$\mathbf{x}'^{\top}\mathsf{F}\mathbf{x} = 0$$

can be obtained [6], where $\mathsf{F} = \mathsf{K}'^{-\top}\mathsf{T}\mathsf{R}\mathsf{K}^{-1}$ is the *fundamental* matrix that encapsulates the two-view geometry.

Matrix F is homogeneous and singular, because T is singular. As a consequence, matrix F maps points onto lines, and because the point $\mathbf{x}'$ lies on the epipolar line $\mathbf{l}'$,

the mapping defined by F is the mapping between points in one image and the epipolar lines in the other:

$$\mathbf{l}' = \mathsf{F}\mathbf{x}$$

It can additionally be shown that $\mathsf{F}^\top$ is the fundamental matrix of the pair in the opposite order.

2.4 Fundamental Matrix Estimation

Estimation of the fundamental matrix F was not a mandatory part of the assignment, and we have implemented it primarily for our own information. The first part of the task was to make a function for which, given a set of point correspondences, returns the fundamental matrix. Secondly, a function should be made that displayed points in one image and the corresponding epipolar lines in another image.

2.4.1 Theory

The fundamental matrix F must satisfy the condition

$$\mathbf{x}'^\top \mathsf{F}\mathbf{x} = 0$$

and this means that it is possible to estimate F, given enough point correspondences. Indeed, the algorithm for estimating the fundamental matrix from point correspondences is similar in nature to the algorithm for estimating a homography from point correspondences (see Section 1.2), as well as a host of similar estimation problems. We have described the calculation for homography estimation in some detail; we will provide comparatively less detail here.

Each point correspondence gives rise to one linear equation; therefore, in the minimal case one would need 7 points to estimate F, since the last 2 constraints are found in the fact that F is homogeneous and singular.

The singularity constraint is not linear in the elements of F and is therefore cumbersome to enforce [6], so we ignored the singularity constraint and used 8 points to find the initial linear solution $\tilde{\mathsf{F}}$. We then enforced the singularity constraint by replacing $\tilde{\mathsf{F}}$ with the closest singular matrix, F, where $\det \mathsf{F} = 0$, using SVD [4] and by finding the smallest singular value of $\tilde{\mathsf{F}}$ and setting it to 0. As usual, normalization is required for improved accuracy of the result; in fact, in the case of estimating the fundamental matrix, it is considered the key to success. Normalization is discussed in more detail in Section 1.2.

Hartley and Zisserman [4] advise that the 8-point computation for the fundamental matrix is therefore done in two steps: 1) derivation of the linear solution $\tilde{\mathsf{F}}$, and 2) enforcement of the singularity constraint and the replacement of $\tilde{\mathsf{F}}$ with F, the closest singular matrix to $\tilde{\mathsf{F}}$.

2.4.2 Implementation

Our implementation therefore comprised the following steps:

1. The user selects at least 8 point correspondences.

2. The point sets are normalized (translated and scaled).

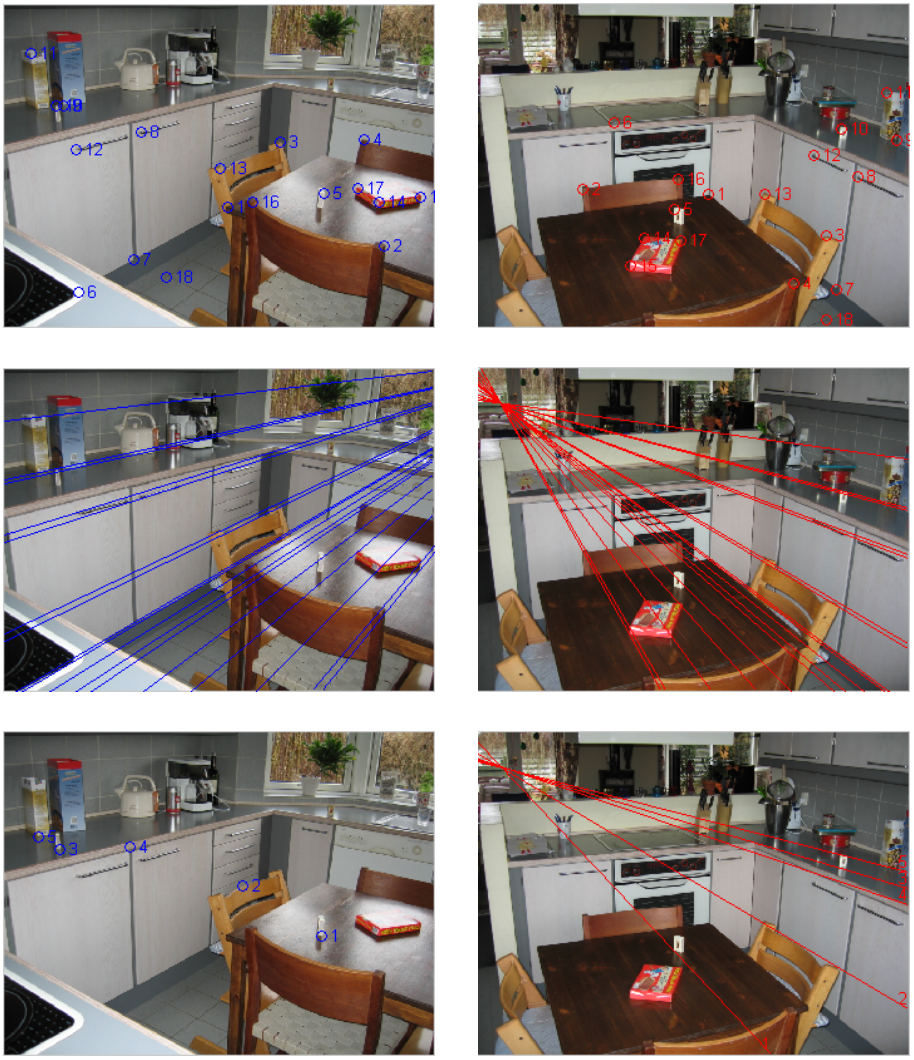

Figure 2.8: Estimation of the fundamental matrix using the 8-point algorithm. *Top row:* Point correspondences used for estimation of F. *Middle row:* Epipolar lines determined by the corresponding points. The epipoles (locations where epipolar lines meet) reveal camera locations: the epipole in the *left* image is where the camera was when the *right* image was captured, and vice versa (the left epipole is outside the boundary of the image). *Bottom row:* Testing points and the corresponding epipolar lines.

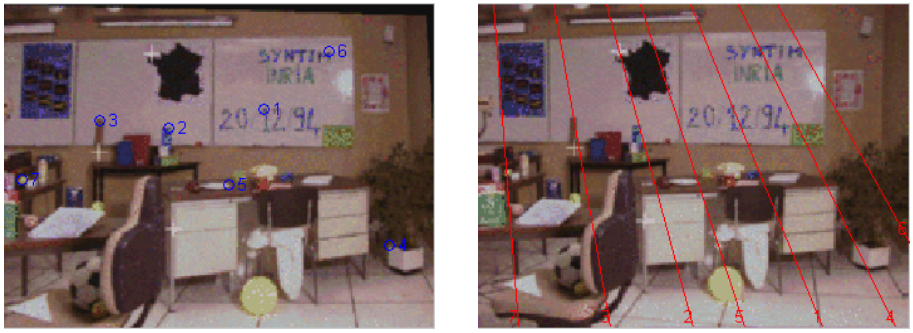

Figure 2.9: Further testing of estimations of the fundamental matrix for a pair of images.

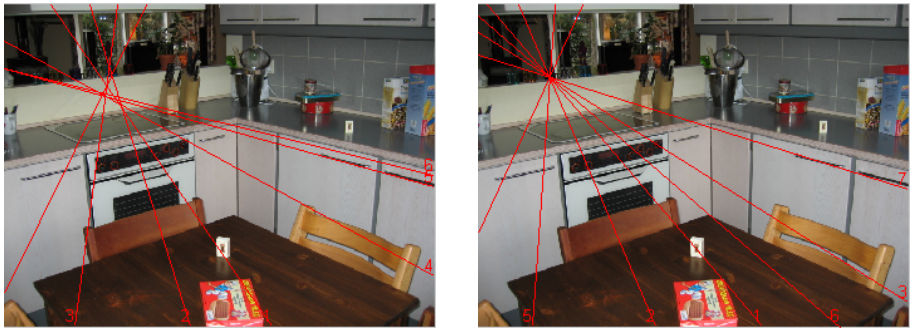

Figure 2.10: *Left:* The effects of a non-singular fundamental matrix, i.e., if Step 5 of the 8-point algorithm is omitted. Epipolar lines do not meet in a single point. *Right:* Results of enforcing singularity using the SVD method.

3. A linear system is composed by adding a row to the matrix B for each point correspondence.
4. The solution of the linear system is found as the smallest singular value of B (please see Hartley and Zisserman [4], or our implementation, for an explanation of how B is calculated).
5. Using another SVD, the initial solution is replaced by the closest singular matrix (i.e., replacing the last column of V by zeros).
6. The final solution that matches the original data is obtained by de-normalization.
7. Plot the epipolar lines in both images.
8. Plot the points in one image and the corresponding epipolar lines in a second image.

2.4.3 Results

Figure 2.8 displays two images of a scene, each taken from rather different positions and angles. The top row shows the point correspondences that were used to estimate

the fundamental matrix. In the second row, we have plotted the corresponding epipolar lines in both images, and in the third row are the results of the test of the estimated fundamental matrix, with points and corresponding epipolar lines plotted together. Epipolar lines meet in the epipole, and by looking at the figure we can verify that the epipole is the image of the camera center of the other view.

Figure 2.3 is another example of two images taken from slightly different positions and angles, for which we calculated and tested the fundamental matrix.

Figure 2.10 demonstrates the effects of a non-singular fundamental matrix. The epipolar lines do not meet in a common epipole when Step 5 of the algorithm, replacing the initial solution with the closest singular matrix, is omitted.

2.5 RANSAC

We implemented the RANSAC algorithm for estimating a line. The RANdom SAmple Consensus algorithm is a robust estimator, one that can find the best estimate of a line from a set of points, even given a large proportion of outlying points.

2.5.1 Theory

When fitting a straight line through a set of 2D points, the least-squares solution (for example, SVD) will provide the best-fit line, such that the distances between all points in the set and the line are as small as possible. These methods are especially convenient when the set of points already suggests a line. But when the set of points includes obvious outliers, and we can intuitively presume (or know already) that the outliers represent some error or belong to some other data set, it becomes clear that least-squares, executed over the entire data set, will return a solution that can be quite different from the apparent one. We want a method that will ignore the outliers, and return a solution that approaches or meets our expectations.

RANSAC, being a robust estimation algorithm, best-fits a line by testing all or a subset of all possible lines (with large data sets, it is time-consuming and unnecessary to check every possible point pair), and then returns the line with the most *support*, that is, the line with the greatest number of corroborating inliers. Since outliers will normally be cast among the solutions with the least support, it is not probable that the algorithm will return a best-fit line that includes them.

In Computer Vision the RANSAC algorithm is often used for automatic computation of homographies and fundamental matrices, since automatically determined set of corresponding points usually contains outliers. We present our attempt of automatic homography estimation in section 1.3.7.

2.5.2 Implementation

The RANSAC algorithm is presentedin Hartley and Zisserman [4] for RANSAC robust estimation, which was adapted from Fischler and Bolles [2]. The algorithm asks for the threshold on the number of inliers, and then iterates until the suitable line is found. We simplified our implementation by setting the number of trials as input.

1. Determine number of iterations n, and the distance threshold t.

2. For n iterations do:

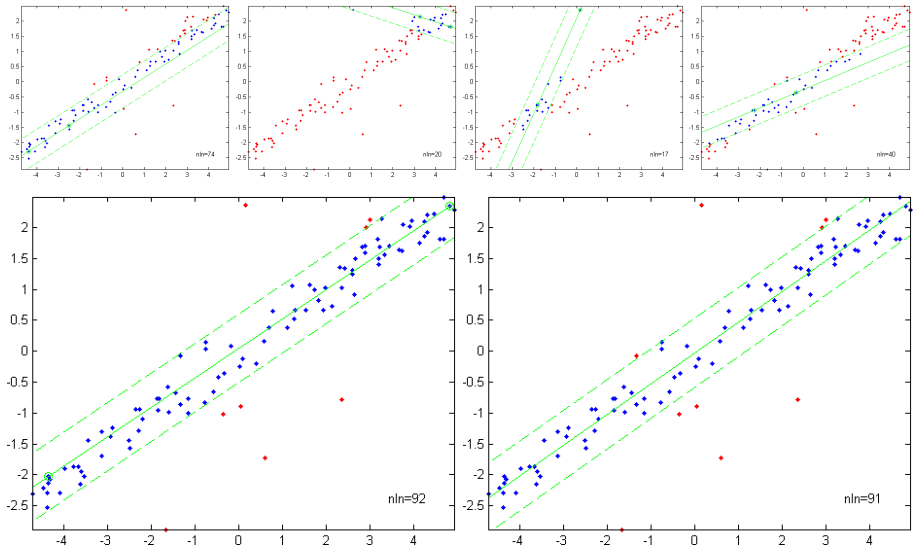

Figure 2.11: Plots illustrating RANSAC execution. *Top row:* Four random samples. *Second row, left:* Best sample (out of 20); *Right:* SVD-fitted line through the inliers of the best sample.

- Randomly choose two different points from the data set, and estimate the line that connects them.
- Count the number of points that lie within the threshold distance from the line—the inliers (support) for the sample.
- If the support is the highest achieved, save the line.

3. Return the best line.

In sum, RANSAC performs an initial estimate for the data set, essentially dividing the set into two subsets, inliers and outliers. The best-fit is then made on the basis of the inliers only, and this can be done by means of least-squares, SVD, Huber, or some other robust norm. The model is evaluated by consensus, and depends heavily on the criteria initially declared for the distance threshold.

In our implementation SVD-fitted the final line trough the set of inliers of the best sample returned by RANSAC.

In step 1 of the algorithm the number of iterations has to be set. How can one know when sufficient number of samples has been tested? A usual practice is to test so that there is a 99% chance of finding at least one sample with all inliers. The needed number of iterations can be calculated from the probability that any data point is an inlier.

2.5.3 Results

Figure 2.11 displays RANSAC in action. In this example, we chose to run 20 iterations over the 100-point data sets. In the top row we display few of the evaluated samples, with different number of support. In the bottom row we have a best sample as returned by RANSAC, and also the SVD-fitted line trough the inliers of the best sample. Note

that the line fitted trough the inliers has less support than the best sample. (It is also possible that the line fitted trough the inliers has *more* support then the best sample.)

2.6 Camera Calibration

For this exercise, we were given the following calibration matrix:

$$\mathsf{K} = \begin{bmatrix} 640 & 0 & 320 & 0 \\ 0 & 640 & 240 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

and our task was to make a function that displays the normal to a plane in the image.

2.6.1 Theory

The camera's internal calibration matrix K gives the transformation between the image point $\mathbf{x}$ and the direction $\mathbf{d} = \mathsf{K}^{-1}\mathbf{x}$ of a ray defined by $\mathbf{x}$ and the camera center.

The direction $\mathbf{d}$ is expressed in terms of the camera's coordinate frame, so it is not particularly useful in and of itself if we don't know the orientation of the camera. However, with two directions available, we can measure the angle between two rays passing through the camera center, and then we can also measure the angle between two scene lines from their vanishing points. This allows for the formulation of a number of orthogonality relationships among vanishing points and lines.

To solve the task we used the following relation:

> If a line is perpendicular to a plane then their respective vanishing point $\mathbf{v}$ and vanishing line $\mathbf{l}$ are related by $\mathbf{l} = \omega\mathbf{v}$ and inversely $\mathbf{v} = \omega^*\mathbf{l}$ ([4], p. 219),

where ω and ω^* are the image of the absolute conic and the dual image of the absolute conic, respectively, which relate to K by $\omega = (\mathsf{K}\mathsf{K}^\top)^{-1}$ and $\omega^* = \mathsf{K}\mathsf{K}^\top$ ([4], p. 210).

2.6.2 Implementation

To simplify the selection of a plane, we limited the problem by stipulating that planes would be defined by a scene rectangle. This can easily be extended to any plane for which the vanishing line can be calculated.

1. ω^* is calculated using $\omega^* = \mathsf{K}\mathsf{K}^\top$.
2. The user selects a plane by clicking on the corners of the rectangle.
3. The vanishing line $\mathbf{l}$ of a plane and the center of the rectangle are calculated.
4. The vanishing point for the normal direction is calculated using $\mathbf{v} = \omega^*\mathbf{l}$.
5. The normal is plotted as a line connecting the center of the rectangle and the vanishing point for the normal direction.

Additionally, a function that estimates the camera calibration matrix purely on the basis of the image's dimensions was made, so as to enable the use of images with other dimensions.

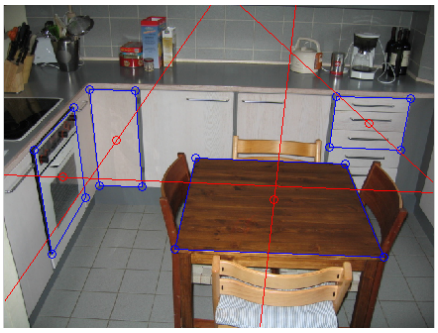
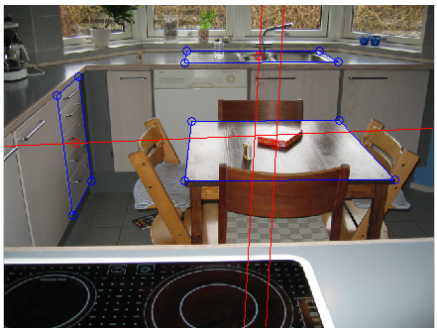

Figure 2.12: Normals to the plane from camera calibration. The user chooses planes by clicking corners to make rectangles (blue), from which the normal to the plane can be calculated (red). Vertical normals were generally well calculated, but note that in the left image, for example, the normal to the cabinets and the normal to the drawers do not meet at the vanishing point where the lines defined by the sides of the table meet. This is probably caused by the poor estimation of the calibration matrix.

2.6.3 Results

Results can be seen in Figure 2.12. One should keep in mind that these results were obtained without a correct calibration matrix. Still, we can see that the direction of the normal was quite well calculated.

Chapter 3

Tracking

3.1 Models for Tracking

Tracking models are often integrated within one another. The Kalman filter and particle filtering are both included among the general set of probabilistic tracking models and the constant-velocity model can be integrated into either Kalman or particle filtering. We will discuss each of these models in turn, and present implementations for both Kalman filtering and particle filtering.

3.1.1 Probabilistic Model

Tracking is about inferring the motion of an object throughout an image sequence. A moving object can be said to have a certain underlying state—which usually consists of such variables as its position and velocity, but often other variables as well. From frame to frame, we obtain measurements, which tell us something about the state of the object.

Two of the most common problems in tracking are uncertainty, wherein the accuracy of observed results is a function of probability, and the presence of noise, wherein errors are introduced before the result can be observed and measured.

In the probabilistic model, the state of the object at the ith frame is the random variable $\mathbf{X_i}$, the measurement obtained in ith frame is the random variable $\mathbf{Y_i}$, and we use the notation $\mathbf{y_i}$ for the *value* of the variable.

Supposing we know what the measurements tell us about an object's state, there are two issues we must concern ourselves with:

Prediction: When there is a sequence of previous measurements $\mathbf{y_0}, \ldots, \mathbf{y_{i-1}}$, how can we predict the state for the next frame? That is, we wish to represent $P(\mathbf{X_i}|\mathbf{y_0}, \ldots, \mathbf{y_{i-1}})$.

Correction: Having also obtained a measurement for the current frame, how do we estimate the current state? That is, we wish to represent $P(\mathbf{X_i}|\mathbf{y_0}, \ldots, \mathbf{y_i})$.

We simplify the problem considerably by assuming that only the immediate past matters, and that the measurement depends only on the current state. Those independence assumptions make it possible to use mathematical induction and Bayes's rule to

obtain the following expressions for prediction,

$$P(\mathbf{X_i}|\mathbf{y_0},\ldots,\mathbf{y_{i-1}}) = \int P(\mathbf{X_i}|\mathbf{X_{i-1}})P(\mathbf{X_{i-1}}|\mathbf{y_0},\ldots,\mathbf{y_{i-1}})d\mathbf{X_{i-1}}$$

and correction,

$$P(\mathbf{X_i}|\mathbf{y_0},\ldots,\mathbf{y_i}) = \frac{P(\mathbf{y_i}|\mathbf{X_i})P(\mathbf{X_i}|\mathbf{y_0},\ldots,\mathbf{y_{i-1}})}{\int P(\mathbf{y_i}|\mathbf{X_i})P(\mathbf{X_i}|\mathbf{y_0},\ldots,\mathbf{y_{i-1}})d\mathbf{X_i}}$$

We now need to find a representation of the tracking problem, one that is strong enough to do the tracking, but simple enough for the above expressions to be evaluated.

If we consider only linear dynamic models and linear measurement models, both with Gaussian noise, all densities will be Gaussian, and their integrals are quite well behaved. We can then obtain the analytical solution for the representation, which we present in Kalman filtering. In case of non-linear models there is no general solution, but one approach is to integrate using sampling of the probability distributions, as we present it particle filtering.

3.1.2 Kalman Filter

Our dynamic model now consists of a state and a measurement, both random variables with a normal probability of a certain mean and covariance. If we denote the value $\mathbf{x}$ of a Gaussian random variable with mean μ and covariance matrix Σ as $\mathbf{x} \sim N(\mu;\Sigma)$, we can write our dynamic model as

$$\begin{aligned}\mathbf{x_i} &\sim N(\mathsf{D}_i\mathbf{x}_{i-1};\Sigma_{d_i})\\ \mathbf{y_i} &\sim N(\mathsf{M}_i\mathbf{x}_i;\Sigma_{m_i})\end{aligned}$$

In 1960, R.E. Kalman published a solution to this problem, known as the discrete-data linear filtering problem [8]. We can also make use of his algorithm to produce not only probable but reasonably accurate results for use in tracking.

The discrete Kalman filter operates within the framework of optimal estimation theory. It is an algorithm that recursively estimates a moving object's *state* and the *uncertainty*, returning the best estimate in the assumptions of linear systems.

Four equations make up the basic algorithm, which essentially takes the estimated state of the object from the previous frame in the sequence and estimates the object's state (position) in the next frame—the *prediction* update. Once the measured state is known, the algorithm recalculates the estimation taking the measurement into account—the correction update.

Following are the Kalman equations, which serve as a general recipe for implementation. We also present a simple example from our assignment in Section 3.2. Assuming we know the initial state $\mathbf{x}_0^-$ and covariance matrix Σ_0^-, the prediction-update equations are

$$\begin{aligned}\hat{\mathbf{x}}_i^- &= \mathsf{D}_i\hat{\mathbf{x}}_{i-1}^+\\ \Sigma_i^- &= \Sigma_{d_i} + \mathsf{D}_i\Sigma_{i-1}^+\mathsf{D}_i^\top\end{aligned}$$

and correction-update equations are

$$\begin{aligned}\mathsf{K}_i &= \Sigma_i^-\mathsf{M}_i^\top(\mathsf{M}_i\Sigma_i^-\mathsf{M}_i^\top + \Sigma_{m_i})^{-1}\\ \hat{\mathbf{x}}_i^+ &= \hat{\mathbf{x}}_i^- + \mathsf{K}_i(\mathbf{y}_i - \mathsf{M}_i\hat{\mathbf{x}}_i^-)\\ \Sigma_i^+ &= (\mathsf{I} - \mathsf{K}_i\mathsf{M}_i)\Sigma_i^-\end{aligned}$$

(from Forsyth and Ponce [3]). These equations provide a recursive solution. First, based on the previous state we *predict* the new state $\hat{\mathbf{x}}_i^-$, where the model of *uncertainty* is the state covariance matrix Σ_i^-. After we obtain the measurement for the current frame, we *correct* the state estimation $\hat{\mathbf{x}}_i^+$, again with uncertainty in the covariance Σ_i^+.

The corrected state estimation is the combination of two terms. The first term is the prediction $\hat{\mathbf{x}}_i^-$, and the second term is the *innovation*, $(\mathbf{y}_i - \mathsf{M}_i\hat{\mathbf{x}}_i^-)$, which is based on the current measurement $\mathbf{y}_i$. It is the *gain* matrix K_i that serves as a kind of weighting function between the two terms, which in essence evaluates whether the estimated state relies more on the prediction or the (generally noisy) measurement. We can see that with a very noisy measurement (large entries in the measurement covariance matrix Σ_{m_i}), the gain K_i would have small entries, making the correction rely predominantly on the prediction.

The Kalman filter general solution is such that the state and measurement covariances and matrices could change from frame to frame. We will focus on a much simpler case.

3.1.3 Constant-Velocity Model

The constant-velocity model is the model we needed to build for the assignment. We consider that one feature point moves with constant velocity between frames. We include both the position and the velocity of the object in the state vector

$$\mathbf{x} = (x, y, \dot{x}, \dot{y})^\top$$

In doing so, we implicitly save all information needed about the past states in terms of velocity.

With the units of velocity expressed in terms of frame rate, we describe the linear dynamic model as

$$\mathbf{x_i} = \mathsf{D}\,\mathbf{x_{i-1}} + \mathbf{d}$$

where D is the *state transition matrix*

$$\mathsf{D} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

and $\mathbf{d}$ represents zero-mean Gaussian system noise, with covariance matrix Σ_d.

At every frame we are provided with the estimation of the point's position, so we describe the measurement model by

$$\mathbf{y_i} = \mathsf{M}\mathbf{x_i} + \mathbf{m}$$

where M is the *measurement matrix*

$$\mathsf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

and $\mathbf{m}$ is zero-mean Gaussian measurement noise, with covariance matrix Σ_m. In this model, the measurement matrix M maps a four-dimensional state to a two-dimensional position, so even though we only see the position of the object, we can still estimate the whole state vector.

We can now use the Kalman filter to track the moving object. There are still a few things to consider, however. We need to supply the initial state and covariance estimates. More importantly, we often need to *estimate* both system and measurement noise. Those settings will greatly influence the estimation.

3.2 Simple Object Tracking

As an assignment, we were given a function that returned a set of measurements ($\times$ symbols in Fig. 3.1, left), roughly tracing the outline of a circle in a counter-clockwise direction. If we assume that this path was made by an object that does not make such erratic movements, we can therefore assume that errors were introduced somehow into the measurements. (Note, due to random generation, measurements were different for each execution.) Our task was to estimate the underlying path of the moving object, and plot the estimation (as in Fig. 3.1, right). Later in the assignment, we would be expected to apply our working Kalman implementation in some eye-tracking exercises.

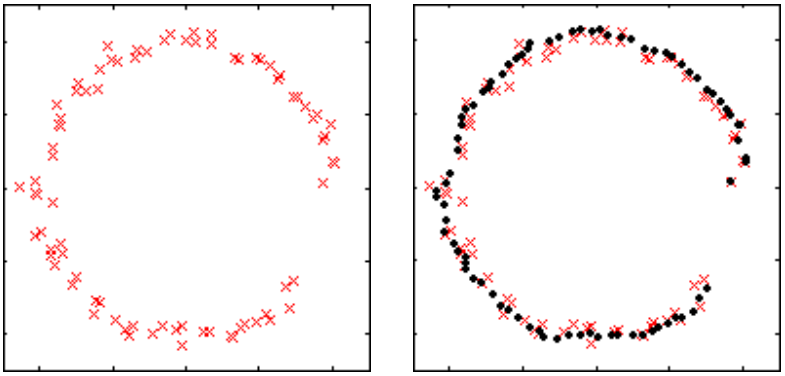

Figure 3.1: *Left:* Given path, corrupted by noise. The $\times$ symbols represent discrete measurements. *Right:* Noisy path shown with estimated underlying states (black dots).

3.2.1 Implementation

To develop our Kalman filter, we used the model described in the previous section and implemented it directly. Apart from functions that implement the equations of the Kalman algorithm, only additional functions are those for visualizing the results: plotting predictions, estimations and uncertainty regions.

We assumed that the movements in the x- and y-directions were independent, so the noise covariance matrices would be diagonal matrices. We also wanted to supply some information about the size of the uncertainty region; we simply extracted the uncertainty in the x- and y-directions to form the diagonal elements of the state covariance matrix (as mentioned in Trucco and Verri [7]), and plotted it as an uncertainty rectangle. The diagonal entries in the covariance matrix were variances, which we expressed in terms of standard deviation, and usually ploted the region spanning ± 3 standard deviations. A general approach is to find the eigenvectors of the covariance matrices and form the *uncertainty ellipse*.

Implementation steps are therefore:

1. All settings are chosen: dynamic model, system and measurement noise, scaling for uncertainty region.

2. State and covariance are initialized.

3. Kalman filtering, for each frame:

   - Prediction: predicted state and covariance matrix are obtained from the Kalman filter.
   - A "measurement" is obtained from the provided `TrackGenerator` function.
   - Correction: the estimated state and covariance matrix are obtained from the Kalman filter.
   - Two plots are made—one with the predictions, the corresponding uncertainty regions, and the actual measurement, the other with the state estimation, the corresponding uncertainty regions, and the actual measurement.

3.2.2 Results

We tried to track the dot with different estimations of the system and measurement noise. Results are shown in Figure 3.2. The setting that best met our expectations is in the top tow, with rather small system noise (the point is moving nicely and uniformly in the circle), and rather large measurement noise (causing the erratic observed path). We set the initial position in the origin, and we see that the system finds the correct path after a few frames. Uncertainty is initially large, but it soon becomes stable. The other rows of the figure demonstrate how different noise settings influence the behavior of the Kalman filter.

In the second row we set the system noise to an even smaller value (in hopes of getting a nicer circle). As a result, the Kalman filter estimated the states mainly on the basis of prediction; it followed the noise-free path, not really caring for the measurements. So, we did get a nicer circle, just not the circle we wanted.

In the third row, the system and the measurement noise were 1000 times greater than in the first row, but the *ratio* between the two noises is the same. We see here that it took more frames for the tracking to recover from bad initialization, but after that, the overall appearance of the track was more or less the same, meaning that (once the system stabilized) it was mainly the ratio of the two noises that determined the weighting when estimating the states. The size of the uncertainty region was, however, greatly influenced by the very large noises—the uncertainty regions were so big that they didn't even fit into the plot.

In the last row we set the system noise to be large and measurement noise to be small; the Kalman filter behaved exactly as expected. We immediately saw that the state estimation basically coincided with the measurement, and the uncertainty region for state estimation was very small—the filter relied on (not very noisy) measurements. It was, on the other hand, difficult to predict the movement of such a noisy system, and as a result, the uncertainty regions for predictions were large.

3.2.3 Further Experimentation

The task was to try to use the Kalman filter on the tracking data provided in previous assignments (Sections 1.4 and 1.5).

We simply passed the noisy tracking data through the Kalman filter, and we observed various degrees of smoothing, depending on the choice of the noise parameters. We had to choose settings that would successfully smooth the path, but not deform it. The result that we consider best is shown in Figure 3.3. The system noise here was

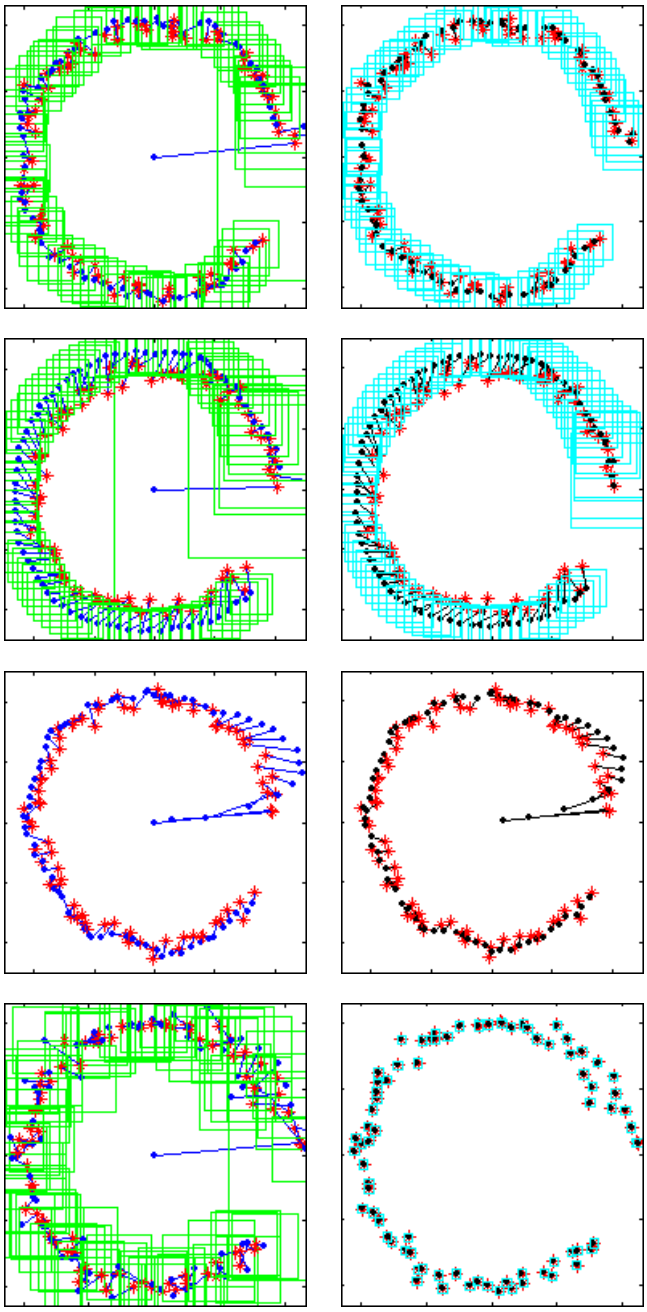

Figure 3.2: Comparison of Kalman implementation of the same noisy path, with varying system and measurement noise. *Left column:* Predictions, their respective uncertainty regions and prediction errors. *Right column:* Corrected state estimations, uncertainty regions for estimations and the difference between the measurement and the state estimate. *Top row:* Small system noise, $\Sigma_d = 0.03\mathbf{I}$, and large measurement noise, $\Sigma_m = 2\mathbf{I}$. *Second row:* Even smaller system noise, $\Sigma_d = 0.03\mathbf{I}$, and $\Sigma_m = 2\mathbf{I}$. *Third row:* System and measurement noise 1000 times greater than in top row ($\Sigma_d = 30\mathbf{I}$ and $\Sigma_m = 2000\mathbf{I}$). *Bottom row:* Big system noise, $\Sigma_d = \mathbf{I}$, and small measurement noise, $\Sigma_m = 0.05\mathbf{I}$).

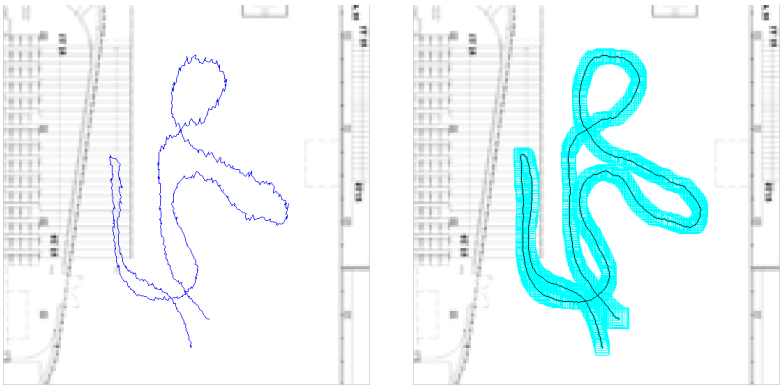

Figure 3.3: *Left:* The noisy path of the walker from the exercise in Section 1.4. *Right:* Smoothed path estimated using Kalman filter and the uncertainty region (cyan). System noise $\Sigma_d = 0.003\mathbf{I}$, measurement noise $\Sigma_m = 10\mathbf{I}$, and the radius of uncertainty region $s = 2$ st.d.

very, very small, but that can be explained by the very small differences between the frames.

We also include a small experiment, in which we kept increasing the measurement noise until the estimated path was so smoothed that it didn't even resemble the original. The same result can be obtained by decreasing the system noise.

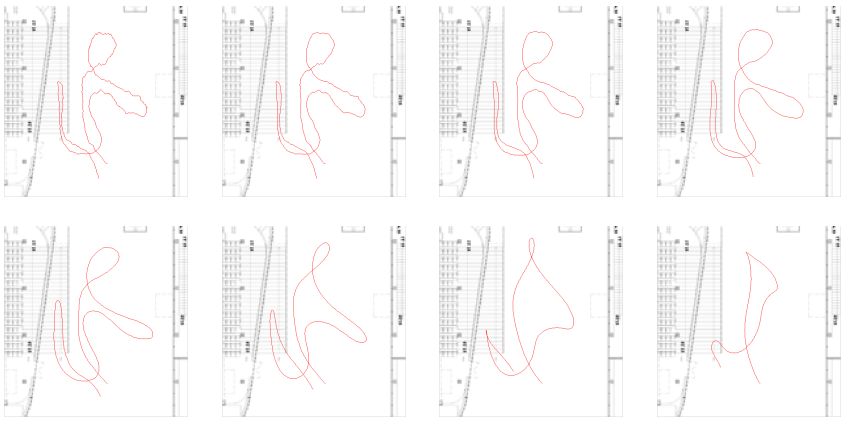

Figure 3.4: More experiments with varying noise. In all, system noise $\Sigma_d = 0.003\mathbf{I}$, while measurement noise Σ_m varies from $0.1\mathbf{I}$ to $10^6\mathbf{I}$.

3.3 Eye Tracking

Our task was to make a Kalman filter for tracking eyes in image sequences. This task is an extension of the previous, so we'll not repeat the issues we already discussed in the previous task. We were given four image sequences, each of which featured a human subject in close-up, slowly rolling their heads. Each image sequences, therefore, showed the two eyes of the subject tracing certain paths, which we were to calculate.

Having the Kalman filter working, we only had to decide on the method that would return an estimate for the location of the eye from each frame. We thus decided to correlate the uncertainty area around the predicted position with the template of the eye, and find the position that yielded the maximum correlation for each frame. To obtain the template and initial location of the eyes, the user was to click on the eyes in the first frame of the sequence (clicking as close to the center of each iris as possible).

One challenge was that as the subjects' faces turned, their eyes would actually change shape—when faces were tilted backward, for example, the eyes looked comparatively narrower than when the subjects faced front. Likewise, the ellipsis shape of the eyes appeared to rotate in one direction, and then back again, over the course of a full revolution of the head.

3.3.1 Implementation

1. All settings are chosen: system and measurement noise, scaling for the uncertainty region, and the size of the template.
2. The first frame of the sequence is displayed, and the user clicks on the eyes. Templates of the determined size are taken out of the image, and the initial position is saved.
3. The state vector and state covariance matrix are initialized.
4. Kalman filtering is performed for each frame of the sequence:
    - Predictions and covariance matrices are estimated from the prediction equations. Uncertainty regions are estimated from the covariance matrices and scaled by the scaling factor.
    - Uncertainty regions are correlated with the templates. The pixel with maximal correlation is taken for the estimated measured position of the eye.
    - Underlying states are estimated from the correction equations.
    - The positions of the predicted states and uncertainty regions, measured states obtained from correlation, and corrected estimated states are plotted on top of the image.
5. The final path is plotted on top of the images from the sequence.

3.3.2 Results

The weak point of this tracking scheme was the correlation. The appearance of the eyes changes a lot across the frames, as described above, and there was a substantial risk that the correlation function would pick up some other facial feature. When that happened, the Kalman filter would be unable to return to the right track—jumping to another feature is definitely not a characteristic of linear measurement with the Gaussian noise. Still, we managed to successfully track eyes in a couple of sequences. We discussed the possibility of using a tracked eye from the previous frame as a template, to deal with the changes in the eyes' appearance, but then we would also need to deal with the propagation of the error.

In Figure 3.5 is an example of a successful track. We had to experiment a lot with different settings, changing the noise matrices, uncertainty-region scaling, and the size of the template. It was very important to adjust the scaling of the uncertainty region

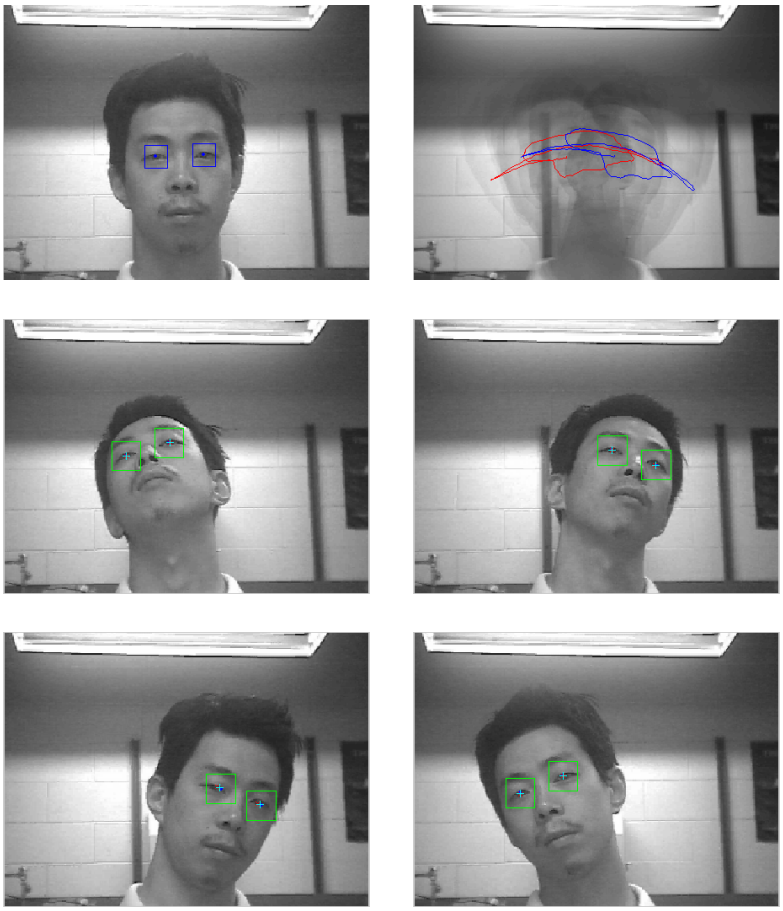

Figure 3.5: An example of successful tracking. *Top-left:* Manual selection of the template and initial positions. *Top-right:* The result of tracking, the complete paths of both eyes. *Rows 2 and 3:* Other example frames from the tracking sequence.

so that it was small enough so as to exclude any other dark facial features, such as nostrils or eyebrows, that might be mistaken for eyes—but also big enough to contain the entire eye. In Figure 3.6 (last row), we show an example of an uncertainty region that is too small. The eye is too close to the edge of the uncertainty region, and because of the zero-padding, it was not found as a point of maximal correlation. In Figure 3.7, we give an example of an uncertainty region that is too big, such that in one frame, the correlation becomes stranded on the nostril. The Kalman filter returns an estimated position that is between the correctly predicted position and the wrongly measured position, which moves the prediction for the next frame to a nostril and can never correct itself.

Figure 3.8 provides details of two frames. The blue dot represents the predicted position; the red $\times$ represents the measured estimated position (from correlation), and the cyan + represents the estimated corrected state. On the left side, everything coincides neatly; on the right, however, is an example of the tracked object suddenly changing direction—the prediction keeps moving in the former direction, the correla-

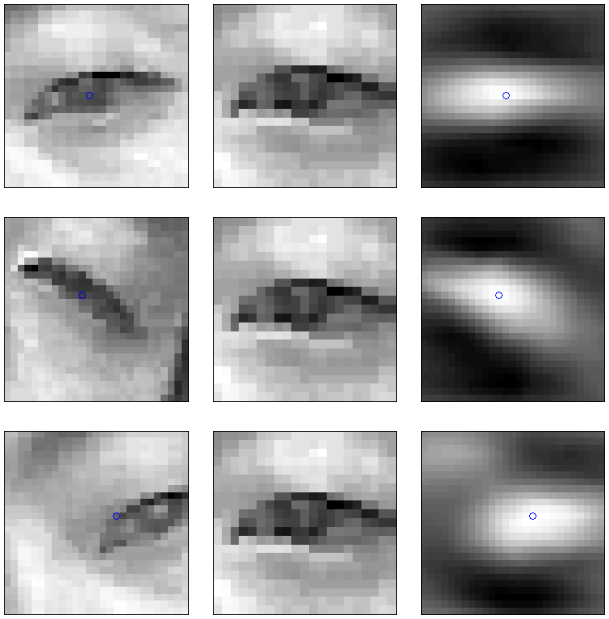

Figure 3.6: Correlation. *Left:* Example uncertainty regions, the position of maximal correlation plotted on top. *Center:* The template from the first frame of the sequence. *Right:* The result of correlation with the position of maximal correlation plotted on top. *Bottom row:* Example of failure; uncertainty region too small.

Figure 3.7: An example of unsuccessful tracking, caused by the uncertainty region being too large.

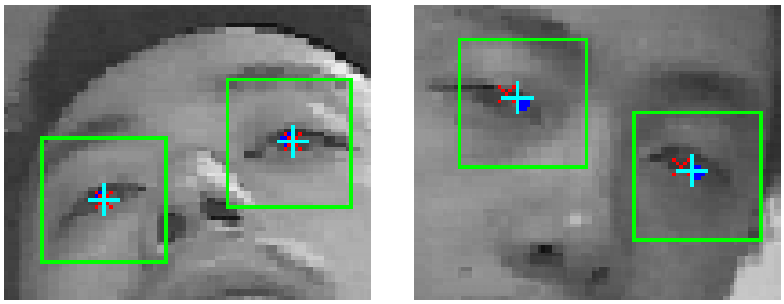

Figure 3.8: Details of example frames from tracking sequence. Tracking can be improved by forward-backward filtering. Blue dots, prediction of position; red $\times$, correlation maximum (measurement); cyan +, estimate of the state.

Figure 3.9: Other eye-tracking examples. *Left:* The left eye is lost and subsequently recovered.

tion correctly measures the positions as going backward, and the estimated corrected state falls in between. The performance of the Kalman filter in such situations can be improved by forward-backward smoothing, where an estimate that fits both the past and the future behaviors of the point is found.

Figure 3.9 shows two other examples of successful tracks. On the left is a track that actually went astray and then recovered itself.

3.4 Particle filtering

Many natural dynamics models are non-linear, making it difficult to represent the probability $P(\mathbf{X}_i|\mathbf{y}_0,\ldots,\mathbf{y}_i)$. Furthermore, the likelihood function $P(\mathbf{Y}_i|\mathbf{X}_i)$ can have multiple peaks, and the largest peak might not correspond to the *right* peak. An example is a measurement that is not only noisy, but sometimes also jumps to the wrong feature. We therefore need the representation of $P(\mathbf{X}_i|\mathbf{y}_0,\ldots,\mathbf{y}_i)$, which allows us to keep track of multiple peaks until it is resolved which one is the right one. One approach is particle filtering—a method that maintains a radically different representation of the relevant distributions than Kalman filter does ([3], from Forsyth and Ponce's notes on their website at http://www.cs.berkeley.edu/ daf/book.html).

The reason why we need the representation of the probability distribution is to compute expectations, which are integrals. It is possible to represent the probability distribution with a random set of weighted samples, and approximate the expectation

with the sum over the samples. This approach of integrating by sampling is called *Monte Carlo integration* and is exploited in particle filtering.

We can do our steps of prediction and correction once our probabilities are represented by samples, called *particles*. For the prediction, we start by having a representation of $P(\mathbf{X}_{i-1}|\mathbf{y}_0,\ldots,\mathbf{y}_{i-1})$ as the set of weighted samples. Moving the samples using the chosen dynamic model, we can generate a representation of $P(\mathbf{X}_i|\mathbf{y}_0,\ldots,\mathbf{y}_{i-1})$ as another collection of samples ([3]). For the correlation, we take the samples at the predicted position and adjust their weights.

For a workable particle filter, we also need to resample the particles at each step. We draw the samples from original set of samples using their weights as the probability of drawing a sample. The new set would predominantly contain particles (probably not distinct) that appeared in the old set with high weight.

3.4.1 Implementation

Following the theory, we implemented an algorithm for a particle filter. Our implementation is simply based on a difference image obtained by background subtraction (more in Section 3.5). We chose to use the simplest dynamic model—random motion.

Roughly speaking, we spread out a population of particles over the the image and let them wander with a certain amount of randomness. For every frame, we evaluated each particle's significance by sampling the slightly smoothed difference image at the sample's position (which should leave a high value for a desired feature, and close to zero otherwise). This evaluation is defining the individual weight of the particles. By combining these weights and the particle positions, we can get a weighted-mean position of our feature in the image. We resample our particle set without changing the total number of particles. It means that we let the heaviest survive (and multiply), while the particles with the small weights are likely to be discarded. And as the last thing before moving onto the next frame, we propagate the particles by simply adding random noise to each one. The propagation step can incorporate a certain dynamic model, where all the particles are first moved according to some deterministic dynamics (*drift*), and then individually perturbed (*diffuse*). We elected to use only the random, diffuse movement for propagation.

In short, the outline of our implementation is as follows:

1. Determine the needed settings: number of particles, noise for propagation.
2. Initialize particles randomly across the image, or the part of the image.
3. Perform the particle filtering, for each frame:
   - Evaluate the particles using the difference image.
   - Calculate the weighted mean of all particles.
   - Plot particles and the weighted mean on top of the image from the sequence.
   - Resample particles so that the number of particles doesn't change.
   - Propagate particles.
4. The final path is plotted on top of the image from the sequence.

3.4.2 Results

We used our particle filter on three image sequences. The fist sequence is an easy task of tracking a single person walking in a empty room. The difference image for this sequence gives a good segmentation, so we used it for learning about the behavior of the particle filter. The filter performed well (see Figure 3.10). In a few empty frames before the person entered the room, the particles were randomly moving around, sometimes finding some noisy pixels. When the person entered the room it took a few frames for the first particle to land on the high-valued pixel, but once that happened the result was a nice track, with high-weighted particles staying within the body of the person and low-weighted particles diffusely floating around. In the plot of the final path, the first part of the path corresponds to the noisy movement before the person entered the room. Figure 3.11 shows a detail of a tracking frame and the difference image which was a base for tracking. We can verify that the particles are assigned high weight on the positions where the difference image has high values.

The second sequence is our much-used film with the person walking on the ground floor of the ITU building. We initially spread the particles only in the area around the person, to increase the chances of the particles finding him quickly. We experimented by changing the number of particles and the randomness of the particle propagation. If the random propagation of particles had a relatively large standard deviation, we observed that the particles jumped from one moving person to another, sometimes splitting in two sets (see Figure 3.12), and often continuing to track the person with darker clothes. When we decreased the randomness of propagation, particles stayed with the object, and we managed to successfully track him through the sequence. A frame from the successful track, the difference image that was the basis for the tracking, and the final tracked path are shown in Figure 3.13. These should be compared with the tracking paths that were provided for the previous tasks and can be seen in the figures of Section 1.4.

We also tried, with less success, to apply the particle filter for tracking eyes. As the basis for this tracking, we used the difference image between two corresponding sequences—one in which the person has dark eyes, and one where the eyes were made lighter. This difference image was rather noisy, and if we increased the randomness of the propagation, the particles were repeatedly getting stranded on the noise (see Figure 3.14). Decreasing the standard deviation of the random movement would result in particles that were unable to follow the eyes. Frames in Figure 3.14 are sorted chronologically, and we can verify that the particle filter is capable of returning from distraction. There were unfortunately too many such distractions for this tracking.

Compared with our success in using the Kalman filter on the same data, the results of particle filtering were a bit disappointing. However, one has to bear in mind that the particle filter was using random motion instead of some better dynamic model. Furthermore, particle filtering was based on the difference image, whereas we used correlation with Kalman filtering.

In the frames where we managed to successfully track the eyes using the particle filter, we still could not plot the path of the movement, having the the weighted mean located in the area between the eyes, jumping from one eye to the other. This made us discuss the possible ways of grouping the particles. We considered using the *k*-means clustering algorithm to divide the particles into two groups. The location of each eye would then be calculated as the weighted mean of the cluster, and whether the eye was left or right would be determined by comparison with the previous frame. We roughly implemented the algorithm, but have never used it due to the poor tracking results.

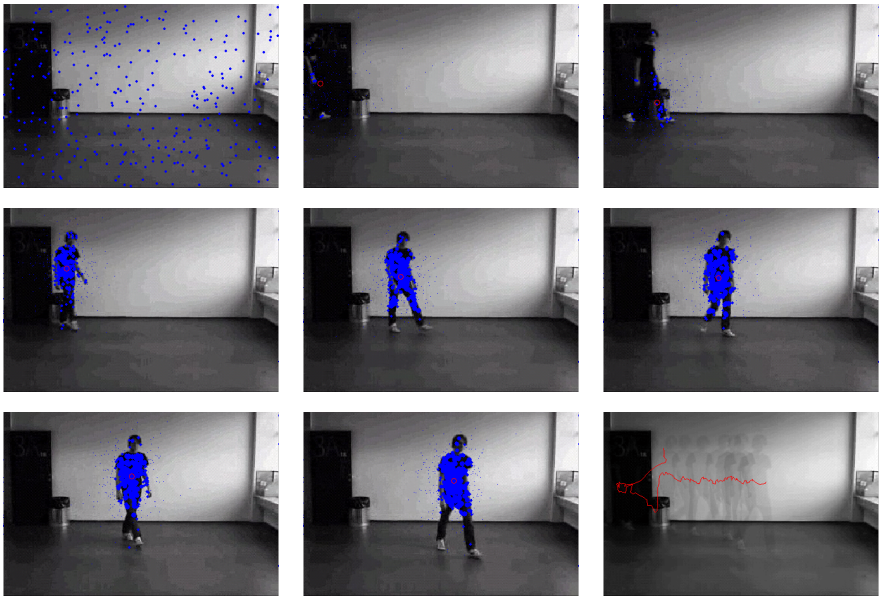

Figure 3.10: Tracking the person entering the empty room. We used 300 particles, and the standard deviation of 30 pixels for the random movement. *Top-left:* Initialization over the whole image, particle weights still not assigned. *Middle:* Tracking, plotted particle sizes correspond to the weights. Weighted mean plotted as a red circle. *Bottom-right:* The resulting path. Note that the noisy part of the path corresponds to frames before the person entered the room.

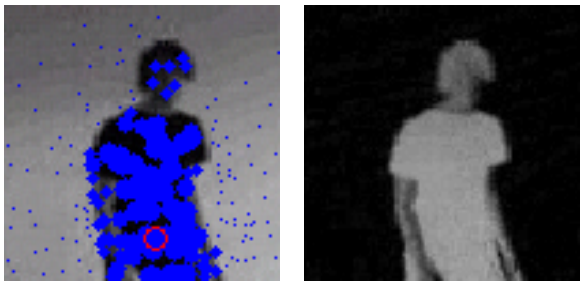

Figure 3.11: Details of the particle filtering frames. *Left:* Tracking frame with the particles of different weights plotted on top of the image. *Right:* Corresponding difference image, which was a base for evaluating weights.

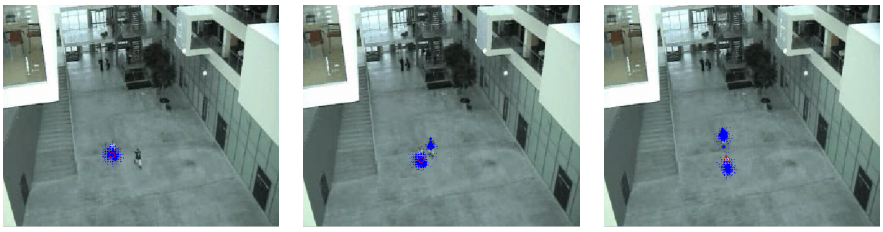

Figure 3.12: Three equally spaced frames illustrating the split of the particle set when tracking a person walking in ITU building. Settings used: number of particles $N = 1000$, standard deviation for random propagation $\mathbf{\Sigma} = 5\mathbf{I}$.

Figure 3.13: Successful track of the person walking. *Left:* A frame from the tracking sequence with the particles plotted on top. *Middle:* Corresponding difference image, which was a base for evaluating the particle weight. *Right:* Final path; should be compared with the figures in Section 1.4.

Figure 3.14: An attempt of tracking the eyes using particle filtering with some relatively successful and some unsuccessful frames, with partial or complete distraction by noise. Frames are sorted chronologically.

3.5 Robust Background Subtraction

We extended simple background subtraction, which we used for the basis of our measurements for particle filtering, with a more robust model. From previous experiences with tracking techniques, we decided to use the watershed algorithm for defining the foreground in an image from a simple absolute difference image, where an "empty" scene (containing no objects of interest) is subtracted from all following frames in the image sequence. All non-foreground parts of each image are then stored in a buffer of a certain length, and we compute the histogram and choose the most frequent value for each pixel over the length of the buffer—in our case $n = 20$ frames.

Following is our implementation of robust background subtraction:

- For every frame, calculate the foreground with the following steps:
  - Compute the absolute difference image by subtracting the background frame from the current frame of the sequence.
  - Blur this with a small Gaussian kernel to suppress noise.
  - Compute the $\nabla x \nabla y$ image, and set all gradients less than a threshold to 0.
  - Run the watershed algorithm on the gradient image.
  - Morphologically close and dilate the watershed image for all values greater than 1 (the extra dilation is used just to make sure the whole foreground is included).
- Find the bounding box of the foreground from the maximum and minimum foreground coordinates greater than zero. All surrounding image information is stored in the background buffer at position i, and the bounding-box area is filled from a reference background image (where i is the position in the buffer, and loops between 1 and n).
- For each coordinate in the image, do a histogram count over the length of the background buffer, and select the maximum count value for the final background image used for the subtraction.

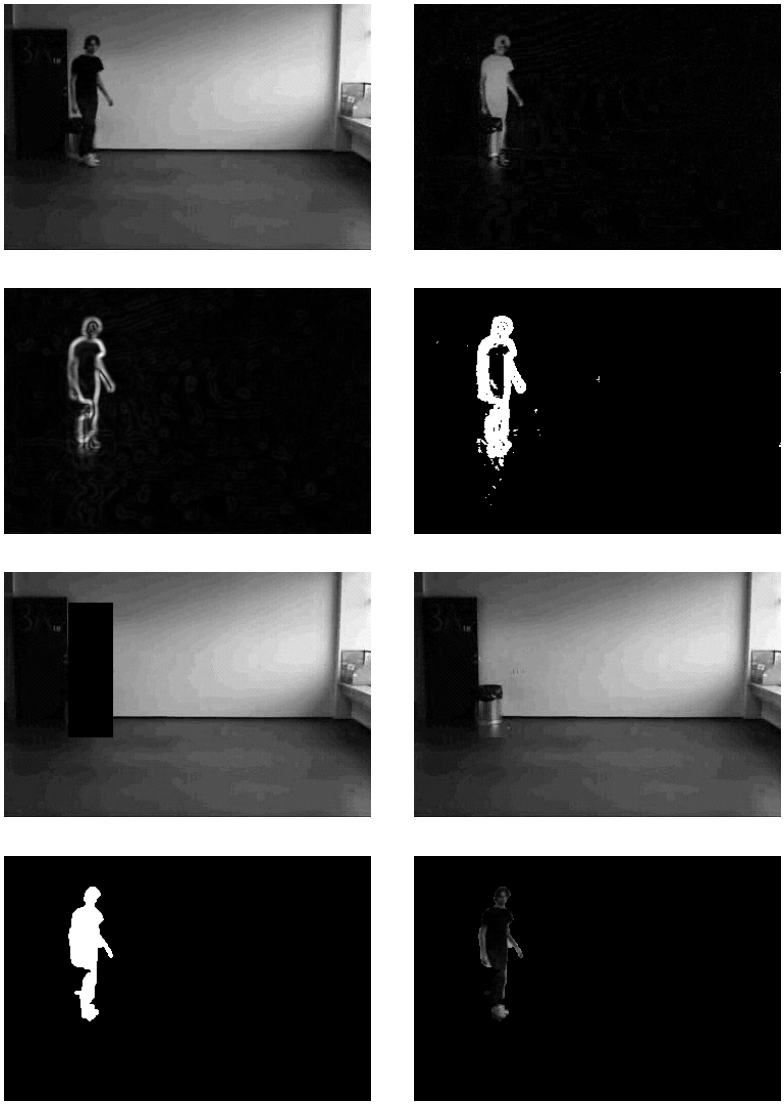

Figure 3.15: *Row 1, left:* Original frame. *Right:* Absolute difference. *Row 2, left:* Gradient image of the absolute difference. *Right:* Thresholded gradient image (set to 1 where bigger than 3). *Row 3, left:* Background area from frame i. *Right:* Foreground area filled from buffer frame $(i-1)$. *Row 4, left:* Watershed, 8 connected components, thresholded to avoid the background level, and dilated with a single pixel. *Right:* Final segmented foreground image.

Conclusion

Conclusions are supposed to reiterate the tone that was set in the Introduction and provide a small recap of what the reader has supposedly just finished reading. We will not waste the reader's time in that way, but merely say, "Well, there it was."

We will also say that this project provided a great many challenges, some of which were enjoyable and ultimately rewarding, such as image mosaic and RANSAC—some of which were not so enjoyable, such as having to wait for large images and long image sequences to load. With nearly every task, we looked for ways to automate the process, which offers a tantalizing direction for us to follow once this project has concluded.

Above all, this project has given us the chance to look at our world with new geometric vision; vanishing points and parallel planes seem to spring into view at every street corner. We visualize the path of a bus changing lanes in traffic. Our gaze naturally identifies landmarks and other structures that would make for an interesting image mosaic. We wonder how we would code our everyday observations and how to create functions that other people would find fascinating and useful. And of course, we are always thinking about the work we have already done and how we might improve it.

"I bet I could do that better if I....What if we try...."

Perhaps soon.

Bibliography

[1] Antonio Criminisi, Ian D. Reid, and Andrew Zisserman. Single-View Metrology. *International Journal of Computer Vision*, 40(2):123–148, 2000.

[2] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Comm. Assoc. Comp. Mach.*, 24(6):381–395, 1981.

[3] D.A. Forsyth and J. Ponce. *Computer Vision: A Modern Approach*. Englewood Cliffs, NJ: Prentice Hall, 2002.

[4] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[5] P. D. Kovesi. MATLAB and Octave functions for computer vision and image processing. School of Computer Science & Software Engineering, The University of Western Australia. Available from: <http://www.csse.uwa.edu.au/∼pk/research/matlabfns/>.

[6] Henrik Aanæs. An Introduction to Multiple-View Geometry. The Technical University of Denmark, Lyngby, 2003.

[7] E. Trucco and A. Verri. *Introductory Techniques for 3-D Computer Vision*. Englewood Cliffs, NJ: Prentice Hall, 1998.

[8] Greg Welch and Gary Bishop. An Introduction to the Kalman Filter. Technical Report, The University of North Carolina, Department of Computer Science, 1995.