UNIVERSITÄT OSNABRÜCK

Department of Computer Science
Department of Cognitive Science

Rasmus Diederichsen

Designing and Implementing a Rephotography Application for iOS

Bachelor's Thesis

First Supervisor: Prof. Dr. Oliver Vornberger
Second Supervisor: Dr. Thomas Wiemann

ABSTRACT

Rephotography is the process of recreating a historic photograph by finding the exact pose and ideally the exact camera parameters to then take a picture from the same spot. The original and new images can be used to document the passage of time and the changes which a static scene has undergone, for instance by blending the two images together. Traditionally, the exercise is carried out by photographers via careful examination of the current camera picture and comparing it with the original image, gradually moving the camera until an optimal registration is achieved. Besides being very laborious, this approach is also quite error-prone, motivating the desire for computerised assistance.

The ubiquity of camera-enabled mobile devices which—contrarily to cameras—can be programmed allows such assistance to be provided, but few aids are available. Two existing mobile applications simplify the procedure, yet still the photographer is required to determine the necessary motion on their own. This thesis presents an attempt to reproduce a more sophisticated system which was prototyped for a laptop with connected camera as a mobile application. This approach makes use of image processing in order to tell the user how to move the camera to recover the original viewpoint.

The theoretical and practical challenges in computing a necessary motion are explored and the system implemented as an iOS application. A detailed evaluation of the results is performed, concluding that the reproduction was partially successful, but some aspects of the pose recovery require further work.

ZUSAMMENFASSUNG

Refotografie bezeichnet das Wiederfinden von Aufnahmepose und Kameraparametern einer möglicherweise historischen Fotografie, um eine Aufnahme vom selben Standort aus zu machen. Das Original und das neue Bild können verwendet werden, um Veränderungen einer Szene über einen längeren Zeitraum zu dokumentieren, indem die Bilder beispielsweise übereinander gelegt oder Teile des einen in das andere Bild geblendet werden. Normalerweise wird die Prozedur von Fotografinnen mittels Ansicht eines Ausdrucks vom Originalbild und viel Geduld durchgeführt. Die Kamera wird hierbei so lange graduell bewegt, bis die aktuelle Einstellung möglichst genau der des Originals entspricht. Der visuelle Vergleich zwischen Vorlage und Kamerabild ist zeitaufwendig und fehleranfällig, was den Wunsch nach Computerunterstützung motiviert.

Die Verbreitung von mobilen Geräten mit integrierten Kamera, die anders als kommerzielle Digitalkameras programmierbar sind, erlaubt solche Unterstützung. Bisher existieren allerdings kaum Ansätze. Zwei bereits existierende mobile Anwendungen vereinfachen das Refotografieren mittels eines Overlays des Originalbildes über das Kamerabild, doch die Nutzerin muss die nötige Kamerabewegung nach wie vor selbst schätzen. Diese Arbeit stellt einen Versuch vor, ein leistungsfähigeres System zu untersuchen und zu implementieren, welches zuvor für einen Computer mit angeschlossener Kamera entwickelt wurde. Die Anwendung nutzt Algorithmen aus Bildverarbeitung und maschinellem Sehen, um der Nutzerin die nötige Bewegung zu kommunizieren.

Die theoretischen und praktischen Herausforderungen bei der Berechnung der nötigen Bewegung werden untersucht und das System für iOS implementiert. Eine detaillierte Evaluation der Ergebnisse zeigt, dass die Reproduktion teilweise erfolgreich ist, wobei einige Aspekte beim Wiederfinden des Originalaufnahmeortes weiterer Arbeit bedürfen.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

This chapter will introduce the notion of rephotography, elaborate on the process of how to make such a photograph and survey existing approaches to simplify it. These include two applications for mobile operating systems which will be briefly discussed. Furthermore, a summary of more sophisticated work by MIT researchers will be given, leading to the problem statement and the goal of this work.

1.1 REPHOTOGRAPHY

Rephotography or repeat photography denotes the retrieval of the precise viewpoint used for taking a—possibly historic—photograph and capturing another image from the same spot, ideally with the same camera parameters. This allows for documentation and visualisation of changes which the scene has undergone between the two or more captures. For instance when documenting urban development, one can present progress of construction, restoration efforts or changes in the surroundings in a visually striking manner, e.g. by blending the photographs together. Figures 1.1 and 1.2 show examples.

When done manually, the photographer must attempt to find the original viewpoint usually by visual inspection of the original image and trying to match the current camera parameters—camera position, camera rotation, focal length, possibly principal point—to the original. The procedure is often carried out by placing the camera on a tripod and comparing a printout of the original image with what can be seen through the viewfinder or the camera screen. The number of parameters to match as well as the difficulty to estimate them purely from comparing two-dimensional images makes the process error-prone and tedious. Visual acuity and experience of the photographer thus place limits on the accuracy with which the camera pose of the reference image can be reconstructed. Some corrections can be done by post-processing the images and warping the rephotograph with a homography to better match the original, but it would be preferable to achieve a good result in-camera.

At the time of writing, few computerised aids are available to the photographer (see Subsection 1.2.1). The advancement of mobile phones and tablet computers with integrated cameras and larger screens presents the opportunity to develop applications which can assist in this endeavour, moving away from the traditional trial-and-error approach. On current digital cameras[1] this is impossible due to their closed infrastructure not permitting to run user programs.

1 At the time of writing, no commercial manufacturer produces a camera with user-modifiable firm- or software. A project at Stanford by Adams et al. (2010) was discontinued (Levoy, 2014).

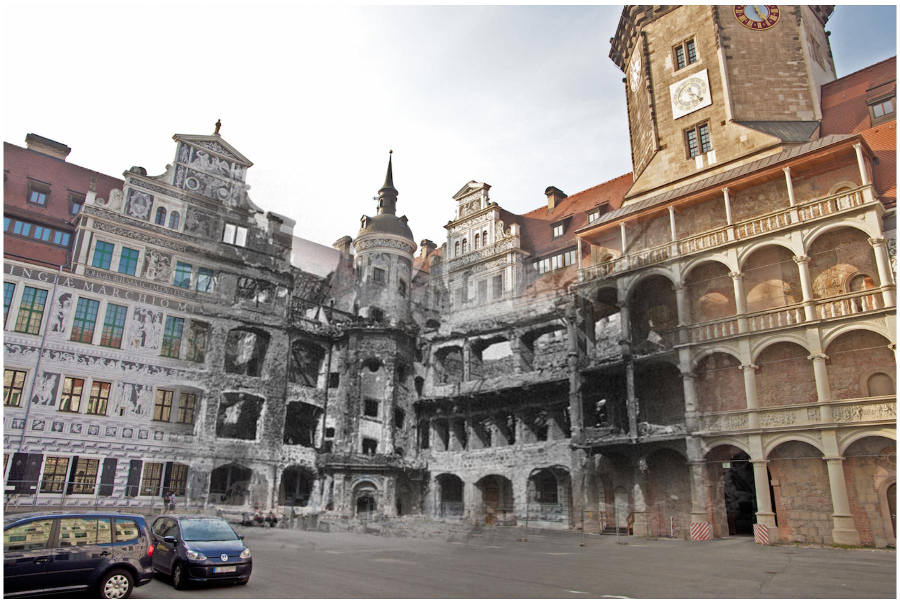


Figure 1.1: Rephoto of the Dresden Castle, destroyed during World War II, © Sergey Larenkov, printed with permission

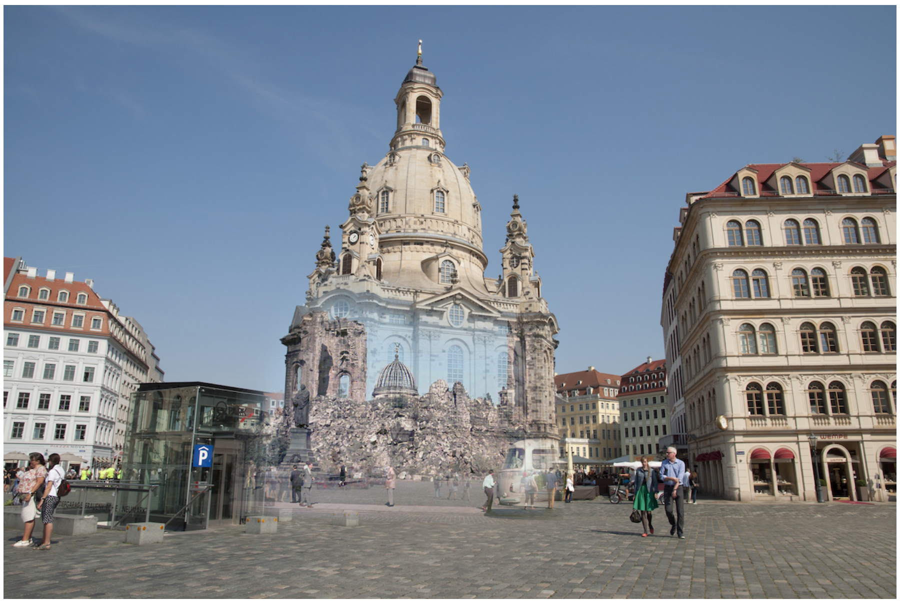


Figure 1.2: Rephoto of the Dresden Frauenkirche, destroyed during World War II, © Sergey Larenkov, printed with permission

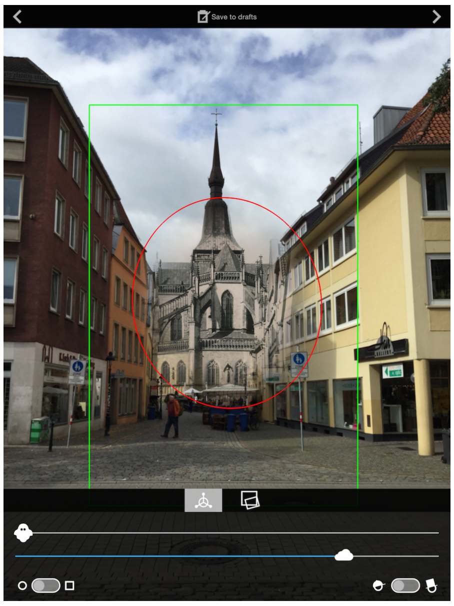


Figure 1.3: Overlay customisation with Timera

1.2 PREVIOUS APPROACHES TO ASSISTED REPHOTOGRAPHY

1.2.1 *Mobile Applications*

Two applications have been developed to assist a photographer in taking rephotographs. For smartphone operating systems, *rePhoto*[2] and *Timera*[3] exist, both available for Android and iOS devices. These applications support the user by placing a transparent version of the original image over the current camera image, allowing for easier alignment. Both apps present the captured rephoto with the original image blended in, but only Timera allows for customisation (see Figure 1.3)

What is characteristical about both of these applications is that the user must still determine on their own how to actually move the camera. An overlay simplifies the procedure, eliminating some of the inaccuracy introduced into the manual approach by the necessity to move the eyes from printout to camera, but it is still the user's responsibility to determine the necessary motion between the current camera position and the goal position (that of the original image).

1.2.2 *Computational Re-Photography*

A more sophisticated automated approach was presented by Bae et al. (2010). They found in preliminary studies that neither a side-by-side view, as would be used in the manual approach, nor a linear blend provided by the above applications result in accurate rephotographs. Their design applies image processing and geometrical reconstruction of the scene in order to guide the user into the right direction.

This subsection will give a brief overview, while a more in-depth discussion of the relevant concepts is deferred to Section 3.4. In this set-up, the relevant parameters of a historic image's camera are reconstructed, including the focal length, principal point and the six degrees of freedom in camera pose with structure-from-motion (SfM) techniques. Five problems are addressed by Bae et al. (2010).

2 `http://projectrephoto.com/`
3 `http://www.timera.com/Explore`

1. It is difficult to communicate a necessary motion to the user if it has six degrees of freedom.

2. While it is possible to compute the direction of translation between two camera frames given some corresponding points, its scale is unknown so that one does not know how far to move.

3. As the translation between two camera frames becomes small, the estimate of relative translation becomes unstable.

4. Historic images are visually very different from current ones, so that automatic feature detection will not work to obtain corresponding points.

5. The historic images' camera is unknown, but its calibration parameters (see Subsection 2.1.2) are needed.

To ease the user's task and remove problem 1., Bae et al. (2010) warp the current camera image according to the camera rotation between the current camera frame and reference image. The user can then focus on correctly translating the camera.

To solve the other problems, user interactions is required. The user captures two frames of the scene with a wide baseline. Manual selection of some correspondences between the historic image and one of the two just taken eliminates the fourth problem, while the second one can be addressed by using the two images and the current camera frame for 3D reconstruction. The fifth problem is eliminated by estimating the unknown parameters after identifying some lines in the image which are parallel in reality.

The software runs on a laptop connected to a digital camera. On the computer screen, the user is shown the current camera image alongside two arrows indicating in which direction to move—one for movement in the sensor plane and one for movement along the optical axis.

The results of this method appear to be very successful, but two main drawbacks exist.

- The prototype is not very convenient, as it requires a (laptop) computer and a digital camera which is impractical for spontaneous rephotography.

- The application is not available to the public, neither in source nor binary form. It is therefore impossible to evaluate or adapt for more mobility.

1.3 GOALS OF THIS THESIS

This work's objective can thus be summarised as follows.

1. Implement in a prototypal fashion the process from (Bae et al., 2010) for a mobile operating system so it can be run on a smartphone or tablet and direct the user in approximate real-time.

2. Evaluate the approach and attempt to reproduce the results.

For a proof-of-concept application, several simplifying assumptions are made. Firstly, it is assumed that the "historic" photograph is captured with the same camera as the one running the application and that the camera is calibrated. Secondly, no strong visual differences between the reference and current scenes are assumed so that the reference image is accessible to the same feature detection algorithm without the user manually labelling correspondences. This entails that the procedure will not work on historic images, but the extension is relatively straightforward.

The application targets iOS 8 and current hardware, as image processing is computationally intense, and has been tested on an iPad Air 2.

2 CAMERA GEOMETRY

This chapter will introduce the geometry of image projection, largely following (Hartley and Zisserman, 2004, chapters 6,7), the geometry of two views (the epipolar geometry, Ma et al. (2003, ch. 5.1)) and how it can be used to recover relative camera position from two images of the same scene.

2.1 CAMERA MODELS

The camera model canonically used in camera geometry is the ideal pinhole camera model which postulates several idealised assumptions.

1. The aperture size is infinitely small.
2. There are no lens effects (*thin lens* assumption).
3. The angle of view is arbitrarily large.
4. All world points projected onto the image plane are in focus, owing to the small aperture.

Given a camera C whose centre of projection is the origin and a point $\mathbf{X}$ in camera-centric coordinates, the central projection of $\mathbf{X}$ onto C's image plane is depicted in a side-view in Figure 2.1. The image plane is virtually moved to the front of the camera, otherwise the image would be mirrored at the principal point as in real cameras. Let f be the focal length, which is the distance of the image plane to the optical centre. If $\mathbf{X} = (X, Y, Z)$, then $x = \left(f\frac{X}{Z}, f\frac{Y}{Z}, f\right)$ by use of the intercept theorem.

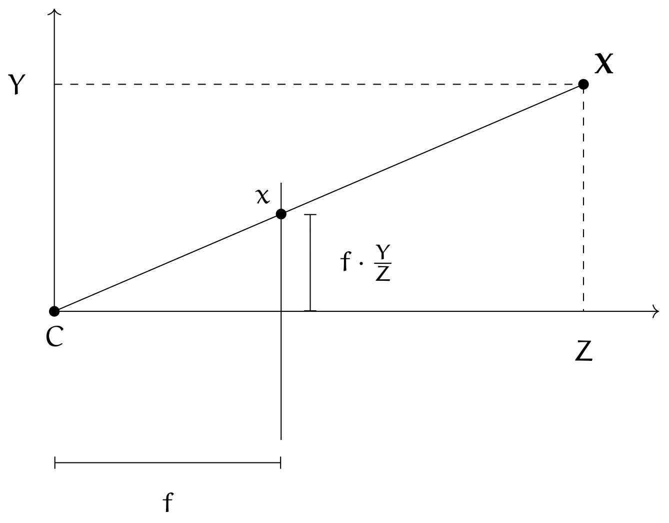


Figure 2.1: Central projection for a pinhole camera

When representing the points as homogeneous[1] quantities, the central projection can be expressed by a matrix multiplication. This can be written with homogeneous coordinates as

$$\begin{pmatrix} fX \\ fY \\ Z \end{pmatrix} = \underbrace{\begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{\text{Projection Matrix of C}} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{2.1}$$

or in short

$$\mathbf{x} \sim P\mathbf{X} \tag{2.2}$$

2.1.1 *Camera Extrinsics*

The above situation is a special case wherein the camera centre C defines the origin and the optical and image axes are the coordinate axes. Thus, the rotation and translation of the camera relative to this coordinate system is zero. More generally, there might be a world coordinate frame with different origin and different axes, so that a coordinate transform must be applied to $\mathbf{X}$ before the projection.

Let $R \in \mathbb{R}^{3\times3}$ be a rotation matrix giving the camera's rotation relative to the world frame and $T \in \mathbb{R}^{3\times1}$ its translation such that

$$\mathbf{X}_{\text{cam}} = R\mathbf{X}_{\text{world}} + T \tag{2.3}$$

Then the projection of a point $\mathbf{X}$ in world coordinates onto the image plane becomes

$$\mathbf{x} = P\mathbf{X} \tag{2.4}$$

$$\mathbf{x} = \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix} [R \mid T]\,\mathbf{X} \tag{2.5}$$

2.1.2 *Camera Intrinsics*

Most cameras are not pinhole cameras. To make them conform to the model, the camera intrinsics need to be known. Above, the resulting image points $\mathbf{x}$ were in *normalised* image coordinates. In particular, the principal point—the intersection of the image plane with the optical axis—was assumed to be $(0, 0)$. But generally, image coordinates are expressed in pixels relative to the upper left corner of the sensor. To convert between normalised and pixel coordinates, the camera's five in-

1 Homogeneous vectors are the elements of projective geometry. They can be obtained from Cartesian coordinates by appending a 1-element. All projective entities which differ only by a scalar factor are equivalent, one writes $\mathbf{x} \sim \mathbf{y}$ if $\mathbf{x} = \lambda\mathbf{y}, \lambda \neq 0$. This has the added effect that points at infinity can be represented by vectors whose last coordinate is zero.

trinsic parameters can be written in matrix form and premultiplied in equation 2.4 as

$$\mathbf{x} = \begin{pmatrix} s_x & s & c_x \\ 0 & s_y & c_y \\ 0 & & 1 \end{pmatrix} \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix} [R \mid T]\,\mathbf{X} \tag{2.6}$$

where s_x and s_y are the focal lengths in x- and y-directions expressed in pixel units per world unit (e.g. cm; s_x and s_y are not necessarily identical, if the sensor has non-square pixels), s the sensor skew (the pixels may not be rectangular; their edges may not be perpendicular) which is usually zero, and the coordinates of the principal point (c_x, c_y) with respect to the origin of the image plane which is usually placed at the upper left corner. The intrinsic camera parameters are assembled in

$$K = \begin{pmatrix} fs_x & s & c_x \\ 0 & fs_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \tag{2.7}$$

and are therefore essential to relate world points to image points which will be important for this application. The normalised coordinates $\hat{\mathbf{x}}$ for a pixel coordinate $\mathbf{x}$ can be computed as

$$\hat{\mathbf{x}} = K^{-1}\mathbf{x}, \tag{2.8}$$

which will remove the effects of the calibration parameters and thus make the image coordinates independent of the camera's internal characteristics.

In theory, these parameters could be obtained from the camera's vendor who knows the precise manufacturing specifications. In practice, only the focal lengths f_x, f_y are known, in most cases only one of them with the hopefully correct assumption of square pixels. Usually, the principal point is assumed to be at the sensor centre and the pixels are assumed to be rectangular. In practice however, there are variances introduced by different causes such as imprecise manufacturing or physical impacts which may decentre the lens such that the principal point is no longer at the centre.

A further complication is introduced by the camera lens which will often have a non-negligible distortion, most prominently radial distortion as depicted in Figure 2.2, but the thin lens assumption precludes distorted images. It can be modelled by the application of a distortion factor to the ideal undistorted image coordinates $(\tilde{x}, \tilde{y})$ and thus removed to satisfy the thin lens assumption. Distorted and ideal image coordinates are related as

$$\begin{pmatrix} x_d \\ y_d \end{pmatrix} = L(r) \begin{pmatrix} \tilde{x} \\ \tilde{y} \end{pmatrix} \tag{2.9}$$

where L is a nonlinear function of the distance r from the distortion centre—usually coincident with the principal point. The function can be approximated as an exponential with a Taylor expansion

$$L(r) = 1 + \sum_{i=1}^{k} \kappa_i r^i \tag{2.10}$$

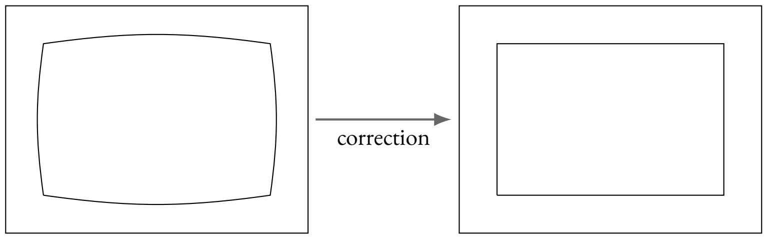


Figure 2.2: Radially distorted image on the left, the corrected image on the right.

for a given k (see Hartley and Zisserman, 2004, ch. 7.4). The intrinsic camera parameters which consist in the entries of K and distortion coefficients κ_i must be determined in order to accurately relate world coordinates to image coordinates. They can be found by calibrating the camera. Different methods exist (e.g Zhang, 2000) but will not be examined here.

2.2 EPIPOLAR GEOMETRY

Epipolar geometry is the geometry which relates the image points in two views of the same scene. Figure 2.3 shows the basic set-up.

We consider a scene viewed by two cameras with optical centres c_1 and c_2, where c_1 defines the origin, world points $\mathbf{X}_i \in \mathbb{R}^3$, where the subscript denotes the coordinate frame—the first camera, arbitrarily chosen to be the left one, or the second camera—and homogeneous image points $\mathbf{x}_i$. Those are the projections of $\mathbf{X}_i$ onto the image planes and thus correspond to the same world point. The cameras are related by a rigid body transform (R, T), where R is a 3×3 rotation matrix and T the translation between the camera centres. Throughout this work, the direction of coordinate frame transformation will be such that

$$\mathbf{X}_2 = R\mathbf{X}_1 + T \tag{2.11}$$

It is obvious that the following relation holds,

$$\lambda_i \mathbf{x}_i = \mathbf{X}_i, \; \lambda_i > 0 \tag{2.12}$$

that is, the world point lies on a ray through the optical centre and the image point.

Given the corresponding points $\mathbf{x}_i$ in two images, the ultimate goal is to retrieve the euclidean transform (R, T).

In case the image coordinates for both cameras are normalised (c.f. Subsection 2.1.2), they have equal units, so starting from equation 2.11, one can derive

$$\begin{aligned}
\mathbf{X}_2 &= R\mathbf{X}_1 + T & & \\
\lambda_2 \mathbf{x}_2 &= R\lambda_1 \mathbf{x}_1 + T & & \text{(by equation 2.12)} \\
\lambda_2 \widehat{T}\mathbf{x}_2 &= \widehat{T}R\lambda_1 \mathbf{x}_1 + \widehat{T}T & & \widehat{T} \in \mathbb{R}^{3\times 3} \text{ with } \widehat{T}\mathbf{x} = T \times \mathbf{x} \\
\lambda_2 \mathbf{x}_2^T \widehat{T}\mathbf{x}_2 &= \mathbf{x}_2^T \widehat{T}R\lambda_1 \mathbf{x}_1 + 0 & & T \times T = 0 \\
\lambda_2 \cdot 0 &= \mathbf{x}_2^T \widehat{T}R\lambda_1 \mathbf{x}_1 & & \widehat{T}\mathbf{x}_2 \text{ is perpendicular to } \mathbf{x}_2 \\
0 &= \mathbf{x}_2^T \underbrace{\widehat{T}R}_{E} \mathbf{x}_1 & &
\end{aligned} \tag{2.13}$$

The product $E = \widehat{T}R$ is the *essential matrix* and the constraint it imposes on corresponding image points the *essential constraint* (see Ma et al., 2003, ch. 5).

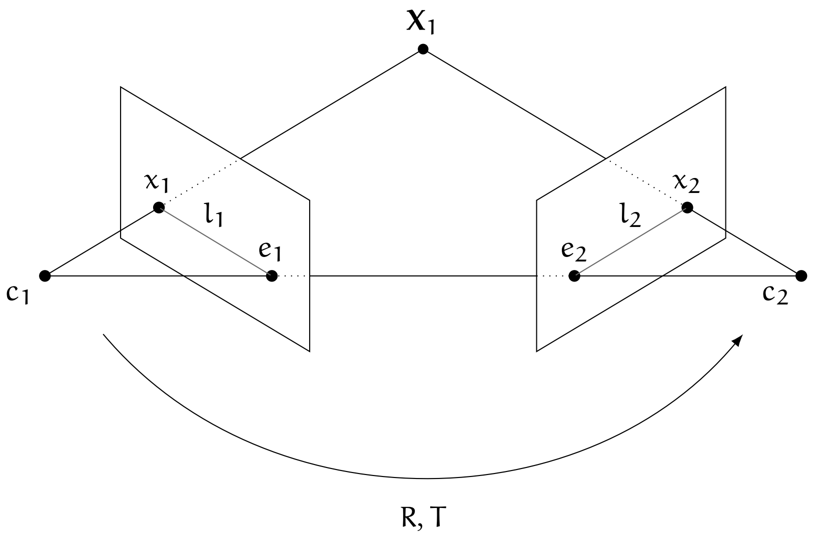


Figure 2.3: Basic epipolar geometry with camera centres c_1, c_2, image points $\mathbf{x}_1$, $\mathbf{x}_2$, a world point $\mathbf{X}_1$, epipoles e_1, e_2 and epipolar lines l_1, l_2

An intuition for the essential matrix can be obtained from Figure 2.3. Given an image point in one frame $\mathbf{x}_1$, in attempting to find the point $\mathbf{x}_2$ corresponding to the same world point $\mathbf{X}_1$, the epipolar geometry restricts the search space to one dimension—the epipolar line of $\mathbf{x}_1$ in the second camera's image plane. The camera centres and the world point define an epipolar plane. The backprojection of $\mathbf{x}_1$ is the ray through $\mathbf{x}_1$ and the optical centre c_1. All points on this ray are mapped to the same point on the image plane of c_1. Depending on how far away the world point $\mathbf{X}_1$ is from c_1, its image on the second camera's image plane will vary—but it will be on the intersection of the image plane and the epipolar plane, the epipolar line of $\mathbf{x}_1$. The line l_2 may be identified with its coimage (the orthogonal complement of its preimage) $l_2 = e_2 \times \mathbf{x}_2 \in \mathbb{R}^3$ so that

$$\forall \mathbf{x} : \mathbf{x} \in l_2 \Leftrightarrow \mathbf{x} \cdot l_2 = 0. \qquad (2.14)$$

The coimage of the epipolar line is the vector perpendicular to the epipolar plane, so every vector in this plane will have an inner product of 0 with this vector. Constrained to vectors in the image planes, this means that all vectors on the epipolar line will have an inner product of 0 with this vector l_2. Referring back to equation 2.13, it can be seen that multiplication with E will yield a term $\mathbf{x}_2^T E = l_2$ which fulfils

$$\mathbf{x}_1 \cdot l_2 = 0, \qquad (2.15)$$

which is precisely the relation stated in equation 2.14. The essential matrix thus maps an image point onto its epipolar line in the other image.

2.3 ESSENTIAL MATRIX ESTIMATION

Estimating the essential matrix between two cameras and decomposing it into relative rotation and translation is a necessity in the endeavour to communicate a necessary camera movement to the application's user. The most prominent algorithms in this regard are the 8-Point algorithm introduced in its original form by Longuet-Higgins (1987) and improved upon by Hartley (1997), and the 5-Point algorithm proposed by Nistér (2004). To illustrate the mathematical tractability of the problem, the former will be presented below.

2.3.1 *The 8-Point Algorithm*

The essential matrix has nine elements, but since all image coordinates are projective quantities, all essential matrices differing by a constant factor are equivalent so that one degree of freedom is removed and at most eight remain. If one were to formulate constraints on the elements as a system of linear equations, eight of those should be sufficient to uniquely determine an essential matrix. An improvement suggested in (Hartley, 1997) is the preprocessing of the input data (the image coordinates) by translation and scaling. This improves the robustness of the algorithm, but will be omitted here.

For each point correspondence $\{\mathbf{x}_1 = (x_1^i, y_1^i, 1), \mathbf{x}_2 = (x_2^i, y_2^i, 1)\}$, one linear equation

$$\mathbf{x}_2^T E \mathbf{x}_1 = 0 \tag{2.16}$$

is generated, which can be rewritten as

$$\begin{aligned} 0 = {} & x_2^i x_1^i e_{11} + x_2^i y_1^i e_{12} + x_2^i e_{13} \\ & + y_2^i x_1^i e_{21} + y_2^i y_1^i e_{22} + y_2^i e_{23} \\ & + \ x_1^i e_{31} \ + \ y_1^i e_{32} \ + \ e_{33} \end{aligned} \tag{2.17}$$

Let $\mathbf{e}$ denote the vector of E's entries in row-major order, then

$$0 = \left(x_2^i x_1^i, x_2^i y_1^i, x_2^i, y_2^i x_1^i, y_2^i y_1^i, y_2^i, x_1^i, y_1^i, 1\right) \cdot \mathbf{e} \tag{2.18}$$

If n correspondences are given, they each contribute one row to

$$A\mathbf{e} = \begin{bmatrix} x_2^1 x_1^1 & x_2^1 y_1^1 & x_2^1 & y_2^1 x_1^1 & y_2^1 y_1^1 & y_2^1 & x_1^1 & y_1^1 & 1 \\ \vdots & & & & & & & & \vdots \\ x_2^n x_1^n & x_2^n y_1^n & x_2^n & y_2^n x_1^n & y_2^n y_1^n & y_2^n & x_1^n & y_1^n & 1 \end{bmatrix} \mathbf{e} = 0 \tag{2.19}$$

For eight noise-free point correspondences in non-degenerate general position, there is a unique solution (up to scale) besides the trivial zero, but in practice, one uses more correspondences and the system is overdetermined, so a least-squares-solution minimising $||A\mathbf{e}|| = \sum_{ij} (A\mathbf{e})_{ij}^2$ is sought. The solution is unique up to scale, since all multiples of $\mathbf{e}$ will satisfy equation 2.19—this is also the reason why the scale of the translation cannot be determined. One therefore introduces the constraint $||\mathbf{e}|| = 1$ which also excludes a trivial zero solution.

This solution vector is the singular vector with the smallest singular value in the singular value decomposition of A or equivalently, the eigenvector of $A^T A$ with the smallest eigenvalue (see Hartley, 1997).

2.3.2 *Further Algorithms*

The 8-Point Algorithm is mathematically straightforward and linear, but in practice it suffers from noise (see Luong et al., 1993) and in real applications approaches the robustness of other methods only in its normalised form from (Hartley, 1997) (not related to *normalised image coordinates*).

It has been noted above that E can have at most eight degrees of freedom. In reality it has only five, as rotation and translation have three degrees of freedom each, and one is lost due to the indeterminable scale. In theory five constraints from five pairs of image points thus are sufficient for finding E. A solution was put forth by Nistér (2004) and improved in (Stewénius et al., 2006). The algorithm is nonlinear and thus much less easily understood and implemented, involving computing the roots of a ten degree polynomial, and requires only five points, but can also be applied to more. It can be considered a state-of-the-art solution to the relative pose estimation problem; its performance in overdetermined cases in the presence of noise compares favourably to other direct methods requiring six (Pizarro et al., 2003), seven or eight points.

Many other methods exist. Some—like the algorithms described above—find a globally optimal solution in closed form while others employ heuristic methods to iteratively approach a local optimum. A review is given in (Zhang, 1998).

Direct methods such as the five-point or eight-point algorithms are frequently used in schemes like RANSAC, which make the estimation more robust to outliers (correspondences which are imprecise or incorrect). For a number of iterations, a hypothesis for E is computed on a minimal number of correspondences and then evaluated on the whole data set. If the inliers in the data far outweigh the outliers, it is probable that a noise-free subsample is selected. The best hypothesis is kept. The simplified algorithm is shown in Algorithm 1 (c.f. Hartley and Zisserman, 2004, ch. 4.8) and requires an error measure for E.

```
   Data: $n$ point correspondences
   Result: a best-fitting essential matrix
1  Let $c_{\text{best}} := 0$;
2  for $i := 0, i < \mathtt{maxIter}$ do
3      Select randomly a minimal number of points to estimate $E_i$;
4      Compute error measure for $E_i$ on all $n$ points;
5      Let $C_i$ be the set of point pairs whose error does not exceed $\varepsilon$;
6      if $|C_i| > c_{best}$ then
7          $c_{\text{best}} := |C_i|$;
8          $E_{\text{best}} := E_i$;
9      end
10 end
11 return $E_{\text{best}}$
```

Algorithm 1: Simplified RANSAC scheme for essential matrix estimation

2.4 DECOMPOSING THE ESSENTIAL MATRIX

One step remains to recover the relative camera pose from corresponding points. As per the derivation in Section 2.2, the rotation and translation between the two cameras is encoded in E. Given an essential matrix, Hartley and Zisserman (2004, ch. 9.6) show that there are four mathematically valid decompositions of E into R and T, corresponding to four distinct geometrical scenarios (see Hartley and Zisserman, 2004, ch. 9.6). Only one of the solutions will place a point $\mathbf{X}_2 = R\mathbf{X}_1 + T$ in front of both cameras, the others cannot be realised in practice. Triangulating one point from its corresponding image points in the two views will therefore reveal the one correct solution, with the translation scale unknown.

3 CHALLENGES IN RELATIVE POSE ESTIMATION

The success of recovery of relative pose mainly depends on two factors: The accuracy and correctness of image point correspondences and them not being in a degenerate configuration, so that an epipolar geometry can be estimated from them. This chapter will examine the preconditions and the feasibility of pose recovery under realistic conditions. Several problematic cases will be identified and described. Furthermore, an assessment of different feature detection algorithms for the purpose of this work will be given.

3.1 DEGENERATE CONFIGURATIONS

Degenerate configurations are those in which the data on which the essential matrix is estimated allows for more than one mathematically valid solution. Two different cases can be observed.

3.1.1 *Structure Degeneracy*

Structure degeneracy is a configuration of points in the observed scene which do not provide enough information to wholly determine an essential matrix. The projection matrix $P \in \mathbb{R}^{3\times 3}$ with

$$\mathbf{x} = P\mathbf{X}$$

has twelve elements but is a projective quantity, so all non-zero multiples of P are equivalent, wherefore it has only eleven degrees of freedom. A degenerate case is for instance one in which all points observed by the two cameras lie on the same plane (or worse, a subspace of even lower dimension). The images of a planar surface in two cameras as well as the planar surface itself and its image are related by a homography (see Hartley and Zisserman, 2004, ch. 13), which is a mapping between planes and has eight degrees of freedom (being a 3×3 matrix and a projective element), meaning that three degrees of freedom are undetermined (Torr et al., 1999). A set of coplanar points alone thus does not provide enough information to uniquely determine an epipolar geometry.

However, not all algorithms are susceptible to this problem. While the 8-point algorithm alone cannot cope with this case, the five-point algorithm can and is generally more robust (Li and Hartley, 2006). While there are other approaches to work around such issues (e.g. algorithms developed by Chum et al. (2005) or Decker et al. (2008), respectively), in practice, a five-point algorithm in a RANSAC scheme works well and needs fewer iterations than an 8-point approach (Li and Hartley, 2006).

3.1.2 *Motion Degeneracy*

A second type of degeneracy occurs when the camera motion between two images has fewer degrees of freedom than the model to be estimated—the essential matrix. If the camera only translates or only rotates between images, the motion has at most three degrees of freedom. As Decker et al. (2008) point out, an essential matrix estimated under such conditions could be consistent with all correct point matches, but also with some false ones due to the mismatch in degrees of freedom between data and model. In schemes like RANSAC, such an estimate will have a large consensus set—all inliers *plus* outliers—and so may lead to the premature termination of the algorithm, despite the current hypothesis being inaccurate.

It is unlikely in rephotography that the observed scene will be completely planar, or that the user will move from the first frame in a motion which is pure rotation or translation. Therefore, these degeneracies can be labelled edge cases and are not specifically handled in the application.

However, the most degenerate case is the one with zero motion between the frames and of primary importance for this application. An initial idea to simply compare the original image with the current frame cannot be successful as relative pose estimation from corresponding points becomes unstable when the motion between the two cameras approaches zero. But this is the ultimate goal one wishes to achieve. When naïvley comparing the current camera image to the reference photograph, the estimate for relative rotation and translation would become increasingly unreliable as the camera approaches the original viewpoint.

3.2 FINDING CORRESPONDENCES

There is a variety of automatic feature detection algorithms which differ in repeatability, robustness to noise, speed, and invariance with respect to image characteristics such as scale, brightness, or rotation. Generally, a feature detector identifies potentially salient points in an image and computes a descriptor for these points in a way that the same point under different conditions will yield an ideally identical descriptor. When points of interest—usually called *keypoints*—are available in multiple images, their descriptors can be compared and the best match according to some metric can be selected for each keypoint. The matches found can then be used as corresponding points for relative pose estimation.

Classical state-of-the-art detectors include *Scale-invariant feature transform* (Lowe, 1999) and *Speeded-up robust features* (Bay et al., 2006) which both compute real-valued descriptors. A natural criterion for selecting the best matching keypoint for a given keypoint is the L_2-norm of the difference of their descriptors, which is a relatively expensive computation. While e.g. SURF improves performance over the computationally demanding SIFT detector, the speed of matching can still present a bottleneck in time-critical contexts. The proliferation of mobile devices with more economic hardware increased the demand for faster detection and matching of features and many solutions have been proposed. On the one hand, efforts have been made at faster feature detection algorithms in general (such as SURF or FAST (Rosten and Drummond, 2005)), but most promising are those which compute binary instead of real-valued descriptors. The matching of n features between images is an $\mathcal{O}(n^2)$ operation when done with brute force, thus often placing a limit on the computational efficiency of the whole process. On real hardware, floating point operations are generally less efficient when compared to integer or binary arithmetic. While for real-valued descriptors d_1, d_2 with size n, an L_2-norm

$$\sqrt{\sum_{i=1}^{n} (d_1^i - d_2^i)^2}$$

must be evaluated, binary strings can be compared with the Hamming distance

$$\sum_{i=1}^{n} d_1^i \otimes d_2^i$$

which is much faster. These descriptors include but are not limited to BRIEF (Calonder et al., 2010, Binary Robust Independent Elementary Features), ORB (Rublee et al., 2011, Oriented BRIEF) and FREAK (Ortiz, 2012, Fast Retina Keypoint). While some invariances are sacrificed in favour of performance (such as rotation invariance of BRIEF, corrected by ORB), they are claimed to match the repeatability of SIFT and others for many use cases while being much faster to compute and compare. Another recent development by Alcantarilla et al. (2013) are AKAZE features, building on and accelerating the previously proposed KAZE detector (Alcantarilla et al., 2012), also using binary strings to describe feature points. This detector has been chosen for this work because it offers significant speed improvements over SIFT or SURF while sacrificing no quality in the tested scenarios (see Chapter 5).[1]

SIFT is the feature detector used by Bae et al. (2010) which is why it and AKAZE will be intriduced and compared briefly here.

3.2.1 *SIFT & AKAZE*

The SIFT detector attempts to find keypoints by convolving an input image with Gaussian kernels of successively larger variance, thus building a stack of image *scales*. This process is repeated for several *octaves*, each starting from a downsampled version of one of the blurred images from the previous scale octave. The resulting pyramid is termed *scale space*. From these Gaussian images, difference-of-gaussians (DOG for short) are computed by subtracting neighbours in scale space. These difference images approximate an application of the Laplacian of Gaussian (a Laplacian filter with prior smoothing to remove noise) which computes the second spatial derivatives of the image and whose extremes are locations of rapid intensity change like edges or corners. Within this scale space, extrema are found which are such pixels in the DOG images whose absolute values are maximal or minimal in comparison with their 26 neighbours in scale space. This search is conducted in all DOG images for all octaves which contributes to the scale invariance. After locating the keypoint with subpixel accuracy in the image, keypoints are discarded if their absolute DOG value is below some threshold as those are unstable and unlikely to be found under different conditions. Furthermore, the Hessian of the image (the matrix of second-order partial derivatives of the image intensity function) is computed to filter out keypoints on edges as they are not accurately localised and thus likewise unstable. Descriptors are computed by assigning a primary orienta-

1 SIFT and SURF are both protected by US patents and cannot be used without a license in a commercial application in this country.

tion after analysing the intensity gradients in the region around the keypoint and binning them into histograms. A keypoint's coordinate system is rotated according to this primary orientation, thus achieving rotation invariance. The resulting descriptor has 128 elements which will occupy at least as many bytes.

In contrast to SIFT, where the scale space construction is linear since convolution is linear, the scale space in AKAZE is nonlinear. Gaussian blurring (a form of isotropic diffusion) at successively higher scale smooths not only noise, but also true object boundaries which decreases the accuracy of localising keypoints. A nonlinear scale space blurs the image in a way that respects the local image structure and thus preserves object boundaries while smoothing out noise. A visualisation of the difference is shown in Figure 3.1. The diffusion is large in homogeneous areas and small in areas of significant intensity change. A nonlinear scale space is built for AKAZE by means of fast explicit diffusion (Grewenig et al., 2010) and the diffusivity is adapted by considering the magnitude of the image gradient at a given point—meaning that it will be smaller for areas of strong change and larger for others. As in SIFT, the scale space consists of several octaves, each downsampled by two with respect to the previous octave. Within the octaves, the images are called *sublevels*. The Hessian matrix determinant is used to identify possible keypoints (in SIFT, this is used to filter out bad candidates), but the candidate must be extremal compared to other keypoint candidates in its neighbourhood instead of neighbouring pixels. The window searched depends on the scale at which the keypoint is located (the window is larger at higher scales with more blur).

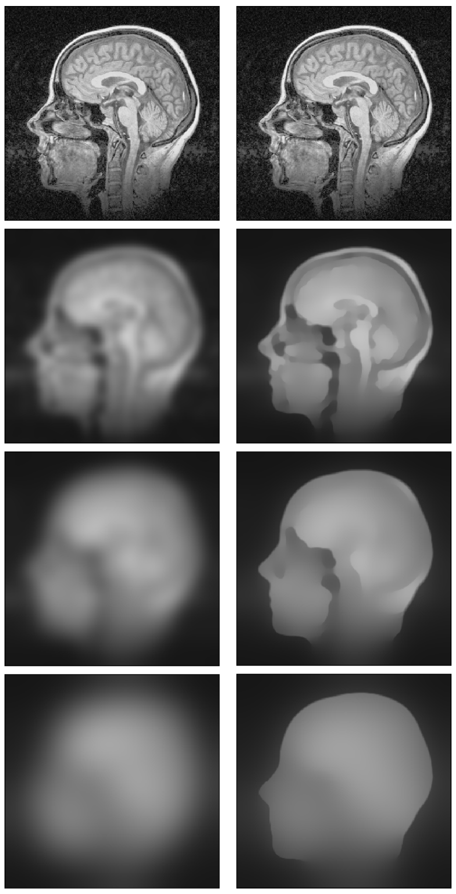


Figure 3.1: Image from (Weickert, 1998, p.120,121). Linear diffusion on the left, anisotropic nonlinear diffusion on the right.

The AKAZE descriptor is obtained by dividing the image region around a keypoint into a grid and then performing binary tests between all such grid cells i, j which test whether $f(i) > f(j)$ for some function f which encapsulates some information of the cell (Yang and Cheng, 2012). To achieve rotation invariance, this grid is rotated according to the dominant orientation computed by considering the image derivatives—already computed for the detection—in the neighbourhood (Alcantarilla et al., 2012). The results of these binary tests are assembled into a binary vector. Its size can be adjusted by making the grid coarser or finer. For f, Alcantarilla et al. (2012) consider both intensity differences and the two image derivatives. The three properties are referred to as *channels* and the descriptor may include one, two or all three of them, with three yielding best performance.

3.3 SCALE AMBIGUITY

Computing relative translation between two cameras from corresponding image points is possible only up to an unknown scale (see Subsection 2.3.1), meaning it is impossible to determine e.g. if an object viewed by the camera is small and close or large and further away. This poses the problem of how to determine if the user is close to the desired viewpoint and whether or not they have come closer or moved further away over the iterations. The issue of unknown scale alone would not be problematic, but an iterative procedure requires to maintain a consistent scale for all estimates of necessary motion. If at one point the estimate says "go right" and after doing so, again "go right", it is impossible to say whether the camera has moved into the right direction or the opposite one and thus whether the goal was approached.

3.4 PRACTICAL PROBLEMS

In the context of a rephotography application as in (Bae et al., 2010), five primary obstacles in viewpoint reconstruction of a historic photograph can be identified.

1. The necessary camera motion has six degrees of freedom—three for translation and three for rotation—which are challenging for the user to adjust simultaneously, as changing one parameter will often necessitate adjustments for the others to improve the registration. Furthermore, the number of degrees of freedom makes it difficult to communicate to the user how they must move the camera.

2. The estimate for relative translation between two views will be defined only up to an unknown scale (see above)

3. The original photo cannot be used directly for pose retrieval, because the estimate degenerates when the user approaches the original viewpoint (see above)

4. Automated computation of relative camera pose will rely on feature detection to find correspondences. However, historical images will often be vastly different from the current scene. Not only may the scene itself have changed considerably, but also the historical image—having been taken by a historical camera—may differ in contrast, sharpness and colours. Neither SIFT

nor AKAZE will reliably find correspondences between such an image and a current photo.

5. The calibration data—most importantly, focal length and principal point—of the historical camera are often unknown. The calibration data is needed for relative pose computation (see Section 2.2).

Bae et al. (2010) address all these issues in the following way. Initially, after loading a historical image, the user is instructed to take two photographs of the scene with a reasonably wide baseline (about 20°). One of them, termed *first frame* is supposed to be taken from some distance from the original viewpoint, the *second frame* should be the user's best eyeballed approximation of it. The wide baseline allows for a more reliable 3D-reconstruction of the scene used to tackle problems 2. and 3.

Problems 4. & 5. — Missing calibration and visual differences

SIFT features are computed and matched between the first and second frames. Given these correspondences, 3D coordinates of the points can be computed. A selection of these is reprojected into the second frame after which the user identifies six or more points in the historical photograph corresponding to these points in the second frame. This allows estimating extrinsic and intrinsic camera parameters of the historical camera by running an optimisation algorithm on an initial estimate for relative rotation and translation between first frame and reference image. Also sensor skew, focal length and principal point of the historical camera can be inferred this way (problem 5.). The reference photograph is not needed any more after this initial step, circumventing problem 4. The principal point's initial guess is found again with help of the user who identifies three sets of parallel lines in the historical image (see Hartley and Zisserman, 2004, chapter 8.8).

In this work, this problem is neglected and it is assumed that the original image can be matched with current photographs, so it should not actually be historic. This restriction is envisioned to be removed in the future.

Problem 3. — Motion degeneracy

A this point the pose of the reference camera relative to the first camera $T_{\mathtt{ref,first}}$ and $R_{\mathtt{ref,first}}$ is known. During the homing process, the current camera frame is compared to the first frame (not the reference frame, avoiding problem 3.), which avoids degeneracy due to the wide baseline. Thus one obtains $T_{\mathtt{current,first}}$ and $R_{\mathtt{current,first}}$. Given the locations of the reference camera and the current frame's camera, each relative to the first frame, one can compute the location of the reference relative to the current frame and thus guide the user in the right direction.[2]

2 Depending on the coordinate system used and whether R and T are given such that they transform points from frame 1 to frame 2 or vice versa, the equations will look different, as they do in (Bae et al., 2010)

$$X_{\text{ref}} = R_{\text{ref,first}} X_{\text{first}} + T_{\text{ref,first}} \tag{3.1}$$

$$X_{\text{current}} = R_{\text{current,first}} X_{\text{first}} + T_{\text{current,first}} \tag{3.2}$$

$$\begin{aligned}
X_{\text{first}} &= R^T_{\text{current,first}} \left(X_{\text{current}} - T_{\text{current,first}}\right) && \text{(3.2, R orthogonal)} \\
X_{\text{ref}} &= R_{\text{ref,first}} R^T_{\text{current,first}} \left(X_{\text{current}} - T_{\text{current,first}}\right) && (3.3) \\
&\quad + T_{\text{ref,first}} && \text{(3.1, substitute 3.2)} \\
X_{\text{ref}} &= R_{\text{ref,first}} R^T_{\text{current,first}} X_{\text{current}} \\
&\quad - R_{\text{ref,first}} R^T_{\text{current,first}} T_{\text{current,first}} + T_{\text{ref,first}} && (3.4)
\end{aligned}$$

and thus

$$T_{\text{ref,current}} = -R_{\text{ref,first}} R^T_{\text{current,first}} T_{\text{current,first}} + T_{\text{ref,first}} \tag{3.5}$$

and

$$R_{\text{ref,current}} = R_{\text{ref,first}} R^T_{\text{current,first}} \tag{3.6}$$

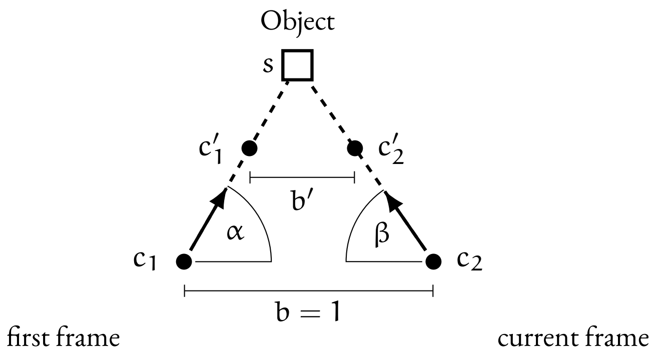
Problem 1. — Complicated motion

During homing, Bae et. al warp the current camera frame according to the necessary rotation before being shown to the user, allowing them to focus only on the translation (problem 1.). This is possible since for rephotography dealing with structures usually at some distance, the rotation will be small, otherwise the warped image would be unusable. This kind of support is also disregarded in this work, as achieving the correct rotation with the help of an overlayed edge image is easy enough, as long as one is directed to the correct spot. Therefore, only the translation is communicated.

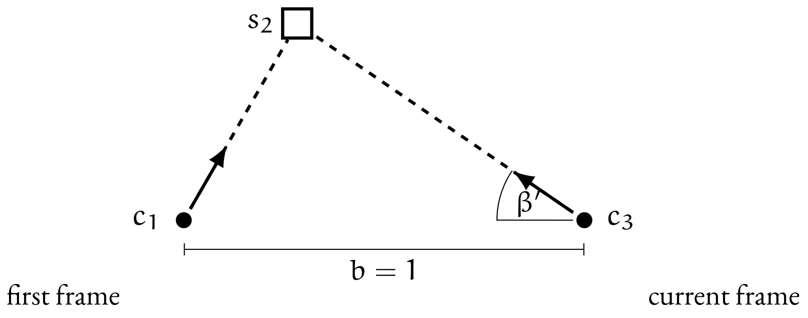
Problem 2. — Unknown scale

A remaining problem (2.) is that the scale of the necessary translation is unknown, so that only the direction can be determined. This poses the question of how to find out whether the user has come closer to the goal or not. It may be feasible to find the original viewpoint nonetheless, if it could be determined at least when the user reaches it, but this is not the case without further information. On top of that, it would make for a better user experience if the distance to the goal could be communicated, too.

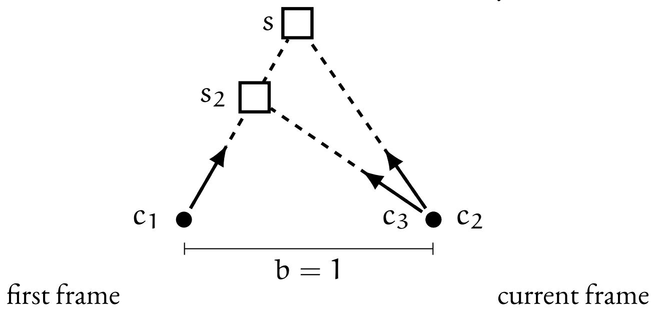
A key observation in this regard is that the actual scale of the translation is irrelevant, it is sufficient that there be a way to make the scale consistent across iterations. That is, it is unnecessary to know whether the goal is a specific distance away, if one can ensure that the translations computed one after the other can be somehow meaningfully compared. For this, Bae et al. (2010) observe that when triangulating 3D coordinates from corresponding points, their computed distance from the camera (the first frame) is inversely proportional to the distance between the cameras. An intuition can be obtained from Figure 3.2. To measure the scale of the world, the application uses the matches between the first and current frames and



(a) Per the intercept theorem $\frac{|sc'_1|}{|sc_1|} = \frac{b'}{b}$. With increasing baseline, $|sc_1|$ also increases to fulfil the equation.



(b) For another baseline, the estimate will still yield a translation with unit length.



(c) Equalising the scales in both estimates shows that the distance of the object to c_1 is smaller if c_3 was further away than c_2

Figure 3.2: The camera baseline length is inversely proportional to the distance of the viewed object to one of the cameras. Evaluating the distance of viewed objects to the first frame's camera yields a measure for how far the two cameras are apart.

computes 3D coordinates by triangulation.[3] The average distance of those points to the first frame's camera centre is computed and compared across iterations to make the scale consistent.

The scale computed in the current iteration compared to the one computed in the initial step for the first and second frames. Scaling the current translation vector by the ratio of the two scales makes the length consistent across iterations and decreasing with the distance to the goal. However, as this estimate relies on the intercept theorem, it is only valid as long as the user moves on a straight line between the initial estimate (position of the second frame) and the goal. Empirical analysis (see evaluation in Chapter 5) demonstrates that strong movement along the optical axis will decrease the usefulness of the scale estimation. However, in

3 The constraints $\mathbf{x} = P\mathbf{X}$ and $\mathbf{x}' = P'\mathbf{X}$ give rise to a linear equation system which can be solved for $\mathbf{X}$ by singular value decomposition as used in the 8-point-algorithm in Subsection 2.3.1. For details, see (Hartley and Zisserman, 2004, ch. 12.2).

reality this should be a minor problem, as the movement around a scene will be stronger than towards or away from it when rephotographing.

For a summary, Figure 3.3 shows the complete pipeline.

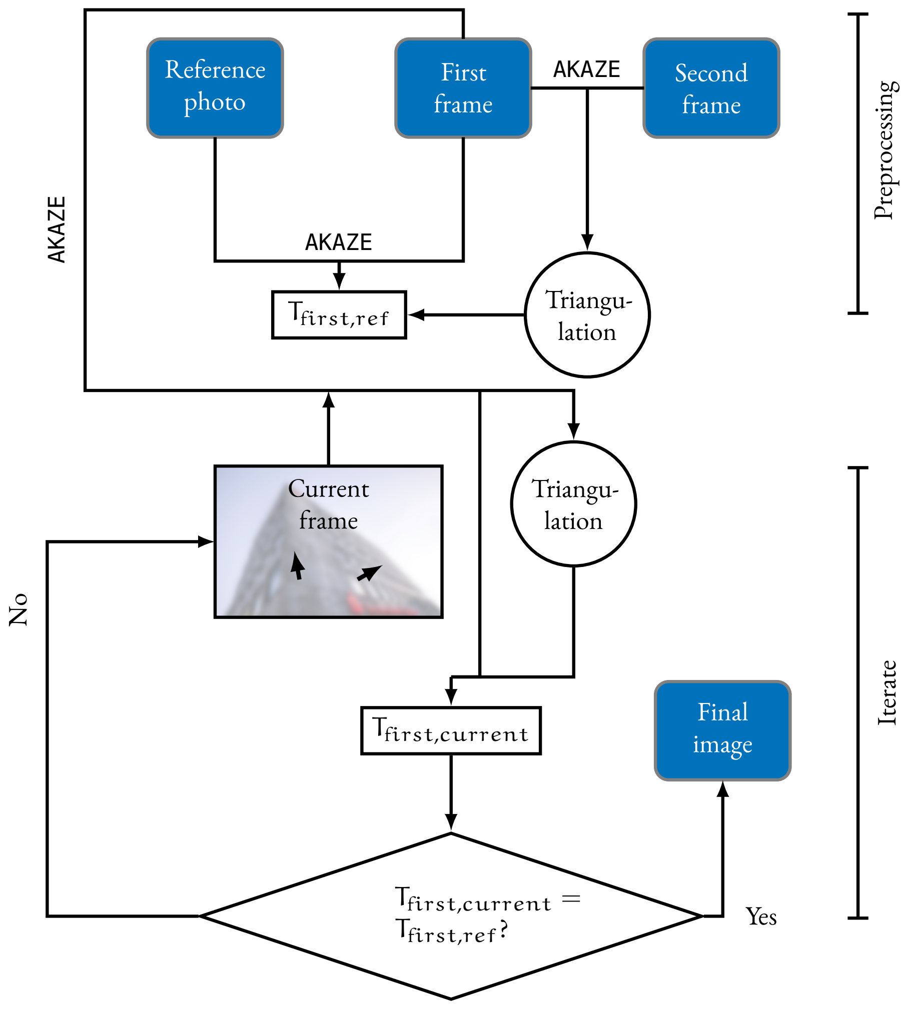


Figure 3.3: Computational rephotography procedure

4 REPHOTOGRAPHY APPLICATION

This chapter will document the development of the iOS application implementing the theory outlined in the previous chapter. The iOS platform will be briefly introduced before the user interface, features and implementation of the app will be described.

4.1 OVERVIEW

4.1.1 *iOS Operating System*

iOS is the operating system running on all of Apple's mobile devices. It is currently[1] at version 8.4.1 with its successor iOS 9 in beta stadium. Code is compiled by the clang compiler and thus all languages supported by it can be used. The primary development language is Objective-C, a strict, object-oriented superset of the C language[2]. Code of these languages can be freely mixed. An Objective-C++ dialect exists to support C++ as well.

The software development kit for the platform employs Objective-C for most high-level APIs and C for more low-level functionality. With respect to application design, the Model-View-Controller pattern (see Figure 4.1) is used throughout the Cocoa library. Cocoa incorporates the standard (`Foundation` classes) and graphical user interface (`AppKit` on OSX, `UIKit` on iOS) libraries as well as `Core Data` for persistence. For applications such as this one with little or no need for data manipulation, there are no dedicated model classes and the controller objects can fill in the role of those.

Following the MVC pattern, iOS requires a view controller for every view hierarchy to present it to the user and mediate interaction with them. View controllers, by virtue of being derived from `UIViewController`, have many methods which are automatically called by the runtime for certain events in the associated view's life cycle. For example

`viewDidLoad` Invoked after the controller's view hierarchy was loaded/inflated from an archive, but before it appears.

`viewDidAppear` Invoked right after the view has appeared on screen.

`prepareForSegue` Invoked before transitioning to another view controller.

In order to implement the relationship between view and model (c.f. Figure 4.1) and respond to user input, the delegate pattern (Gamma et al., 1995, ch. 1.6) is used. Views delegate the responsibility of deciding what should happen on user input to a delegate object, typically the view controller. For example, a view controller might conform to the `UIScrollViewDelegate` protocol[3] and is then required to implement callbacks such as

1 September 21, 2015 (Apple Inc., 2015)

2 There is no precise definition which C version it is a superset of, except that it is ANSI standardised; For each C version supported by the compiler, Objective-C will be compatible with it.

3 Protocols are an Objective-C equivalent of interfaces in other languages

- `scrollViewDidScroll`
- `scrollViewDidZoom`

and more for controlling the associated `UIScrollView`. The pattern is used for all complex views.

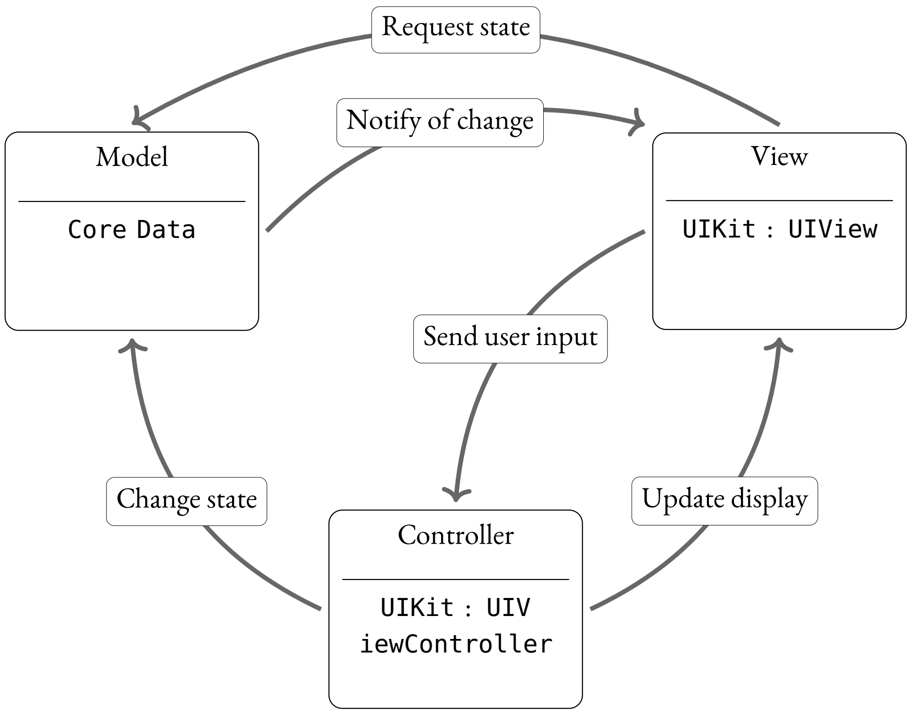


Figure 4.1: The Model-View-Controller pattern postulates three components. The model is the backing store for all data which is presented by the view component. Interaction between the user and the data is mediated by the controller. Decoupling a system like this allows the components to be interchangeable. For example, should the mode of presentation be changed form a graphical to a command-line interface, only view and possibly controller must be swapped, which is also possible at runtime.

Generally, an iOS application is a sequence of view controllers presented to the user in various ways—they can be independent or presented modally over other view controllers which typically control them by becoming their delegate. This form of presentation is used for instance for image pickers giving quick access to the user's photo library or the camera.

Apple's XCode integrated development environment allows to visually model the flow of the application by use of *Storyboard*s. These are XML files which define controllers and their relationships. This allows for a cleaner separation between user interface and business logic, while it is equally possible to specify the presentation order programmatically. Storyboards consist of a number of view controllers with associated views which are connected by *segues*. When an application developed with storyboards starts up, a `UINavigationController` is created which maintains a stack of view controllers on which segues push new ones or pop old ones from (*unwind segue*). The storyboard for the application developed in this work is shown in Figure 4.2.

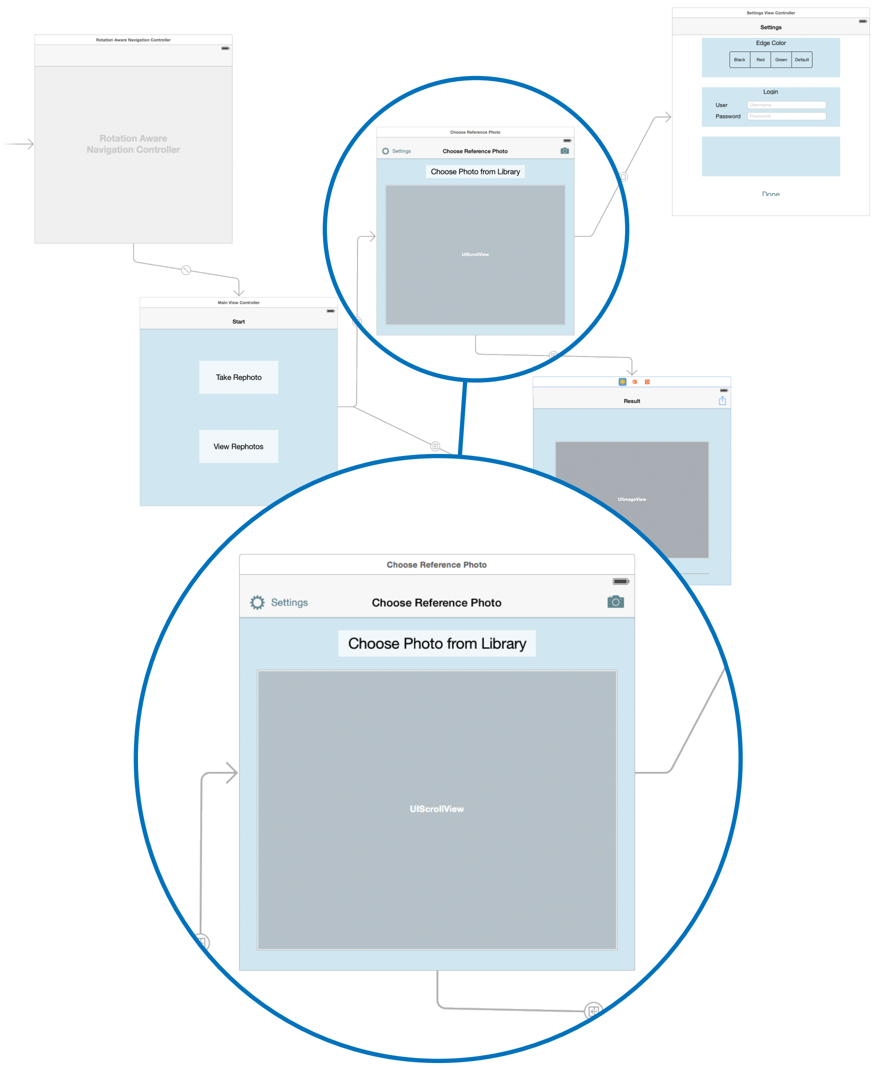


Figure 4.2: The Storyboard of this application. There is always a root view controller on whose stack new controllers are pushed or popped from, here on the top left. The different view controllers are connected by segues. The navigation controller on the top left initially has a Main View Controller pushed on its stack. This one is connected with other controllers with different actions attached to the two buttons and so forth. Inside the storyboard, the actual user interface can be created with drag-and-drop of widgets.

4.1.2 *The OpenCV Library*

For all image processing, the Open Source Computer Vision library (OpenCV) is used in version 3.0. Originally developed by Intel and then by Willow Garage, development is led today by itseez[4], but is strongly community-driven. The library contains many algorithms for image processing for purposes such as segmentation, geometric transformation, feature and object detection and 3D reconstruction. It can be natively used from C++, but Python and Java bindings as well as a deprecated C API exist. The modules used in this work are the `features2d` and `calib3d` modules for feature extraction and multiview geometry (pose recovery), respectively.[5]

4.1.3 *Mixing Objective-C & C++*

OpenCV encodes images in its `Mat` data type, which is the only C++ type also used in other (Obj-C) parts of the application. While it is possible to mix Objective-C and C++ code, using C++ types in header files forces every client including them to also be compiled as Objective-C++, which may be unwanted. For instance, XCode's refactoring tools cannot be used on these sources. It is thus reasonable to create wrapper classes for all C++ types encapsulating the access to the data so that the interface is pure Objective-C. Only the implementation of the wrapper class would need to be compiled as Objective-C++, no client would be affected. A large amount of boilerplate code would be required to translate Objective-C messages to the wrapper in C++ calls to the wrapped object. It is therefore convenient to encapsulate a C++ type but make it still accessible to client classes if needed.

One way of limiting the Objective-C++ to one class is a variant of the *Pointer-to-Implementation* pattern (PIMPL, also known as *Bridge* in (Gamma et al., 1995)). The wrapper class' interface (Listing 4.1) is specified in pure Objective-C and contains an opaque pointer to a C `struct` representing the wrapped data as a C type, which therefore does not lead to compatibility issues as all C code is valid Objective-C. For the original pattern this allows modifying the wrapper hierarchy independently of the implementation hierarchy (e.g. different implementations could be behind the pointers for different platforms). The definition of this pointer-to-implementation type is placed in a second header file (Listing 4.2) and is imported only by those clients which actually need to access the data, not only pass it on. This permits compiling only those source files as Objective-C++ that must deal with the actual OpenCV `Mat`. In principle it would be possible to create methods in the wrapper class for all uses of the C++ type throughout the program. However, this would waste the benefits which e.g. C++ operator overloading yields for readability as all operations would have to be node with method calls. Since all image processing is done in C++ anyway, it is more economic to simply provide access to the wrapped data to those classes that need it.

```
struct CVMatWrapperImpl;
@interface CVMatWrapper : NSObject
@property (nonatomic,readwrite) struct CVMatWrapperImpl* impl; //
    /< Pointer to struct wrapping the matrix
```

4 `http://itseez.com/OpenCV/`

5 Pose recovery functions for calibrated images were unavailable prior to version 3.0. Furthermore, the new version moved all patented feature detection algorithms to a separate repository.

```
@property (nonatomic,readonly) int rows; ///< Number of rows of
    the wrapped matrix
@property (nonatomic,readonly) int cols; ///< Number of columns
    of the wrapped matrix
-(NSArray*)eulerAngles;
-(NSString*)description;
@end
```

Listing 4.1: `CVMatWrapper.h`. The wrapper interface contains an opaque pointer to a C `struct`. including this header does not invalidate othewise valid Objective-C code.

```
struct CVMatWrapperImpl {
   cv::Mat cvMatrix; ///< The wrapped matrix
};
```

Listing 4.2: `CVMatWrapperImpl.h`. The implementation header defines the actually wrapped type. Including it will force a client to compile as Objective-C++ since a C++ type is used. Every class which deals with the data itself can include this header in addition to the wrapper one.

```
#import "CVMatWrapper.h"
#import "CVMatWrapperImpl.h"

@implementation CVMatWrapper
-(instancetype)init {
   self = [super init];
   if (self) self.impl = new CVMatWrapperImpl; // allocate the
       struct wrapping the matrix
   return self;
}
-(void)dealloc {
   if (self.impl)
   delete self.impl;
}
-(NSString*)description {
   //...
}
-(NSArray*)eulerAngles {
   // ...
}
-(int)rows {
   return (self.impl->cvMatrix).rows;
}
-(int)cols {
   return (self.impl->cvMatrix).cols;
}
@end
```

Listing 4.3: `CVMatWrapper.mm`. The wrapper class' implementation must be Objective-C++ and can encapsulate all access to the underlying data, if necessary.

4.2 APPLICATION LAYOUT

This section will walk through the different screens presented to the user and elaborate on the functions of the associated view controllers. An overview of the sequence of screens and controllers is shown in Figure 4.3.

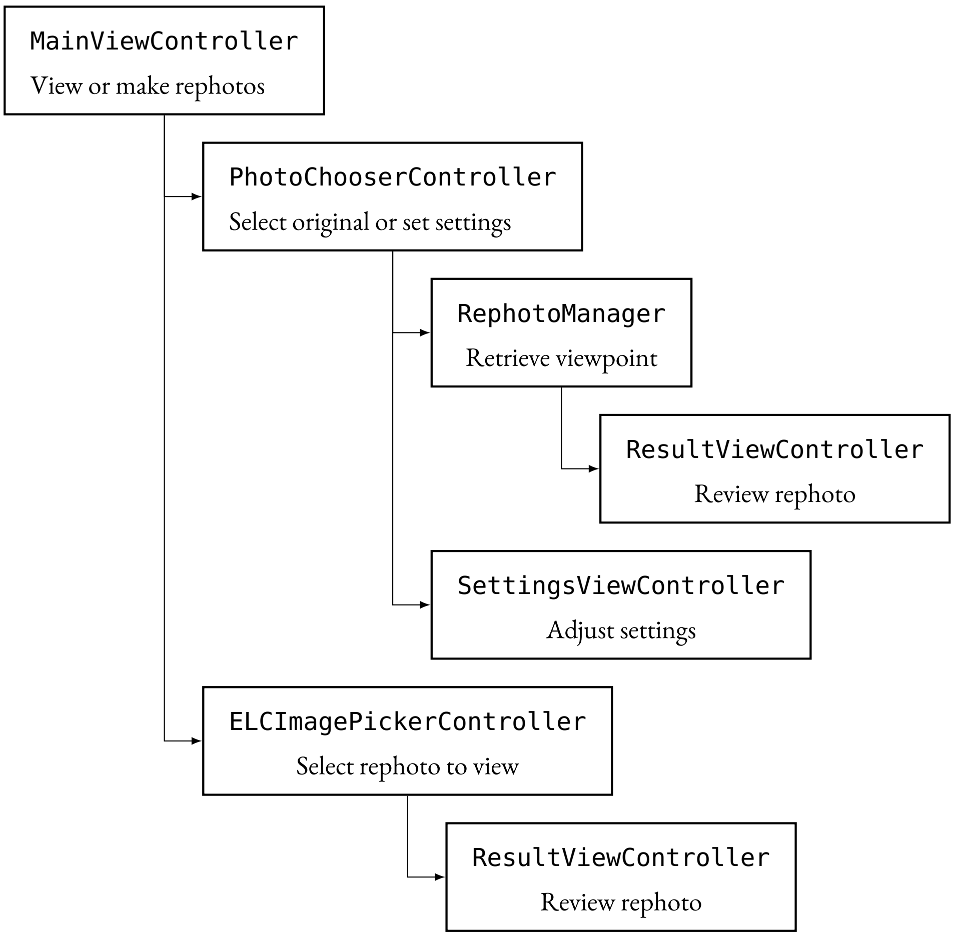


Figure 4.3: Sequence of view controllers

4.2.1 *MainViewController*

When the app starts up, the user is shown a `MainViewController` which offers a choice whether to make a new rephotograph or view existing ones. Clicking the `Take Rephoto` button will trigger a segue to a `PhotoChooserController`, while tapping `View Rephotos` will modally present an `ELCImagePickerController`.

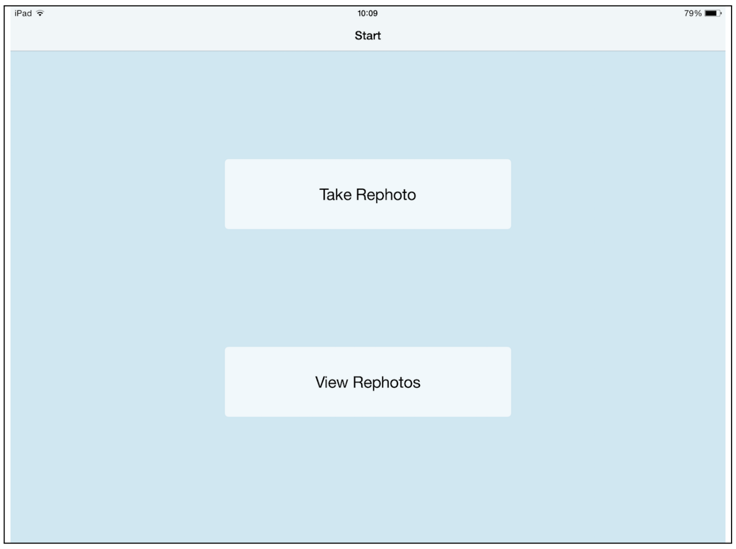


Figure 4.4: MainViewController view layout

4.2.2 *PhotoChooserController*

This view controller displays a scrollable and pannable view and a button `Choose Photo from Library` to load a reference photograph into it. The photo is picked from the user's photo album. Furthermore, the navigation bar contains a button to access the settings (⚙, presents a `SettingsViewController`) as well as a button to start the rephotograph (📷). The latter will modally present a `UIImagePickerController` which exposes a simple API to access the device's camera. A `RephotoManager` is attached to it that controls the rephotography. The `RephotoManager` has a delegate property which points to the `PhotoChooserController`. The controller can then be informed when the rephotograph is done or cancelled.

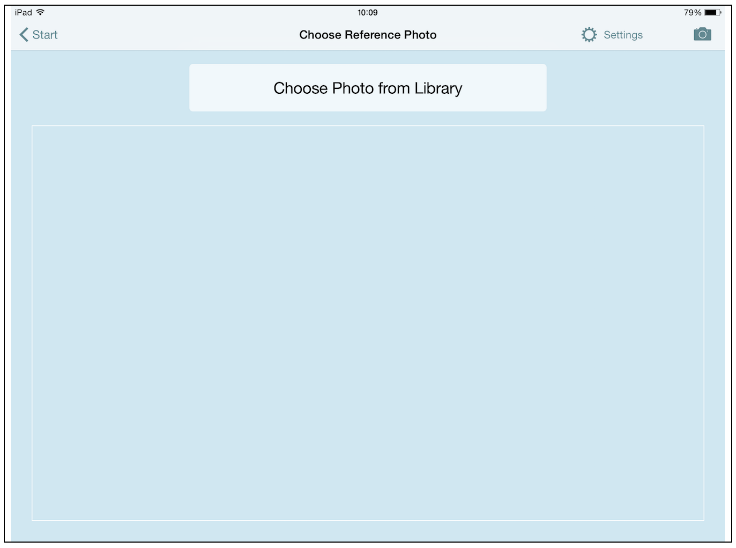


Figure 4.5: PhotoChooserController view layout

When an image is picked, the controller uses the Canny edge detection algorithm (Canny, 1986) to generate an overlay which will then help the user in finding the perfect alignment.

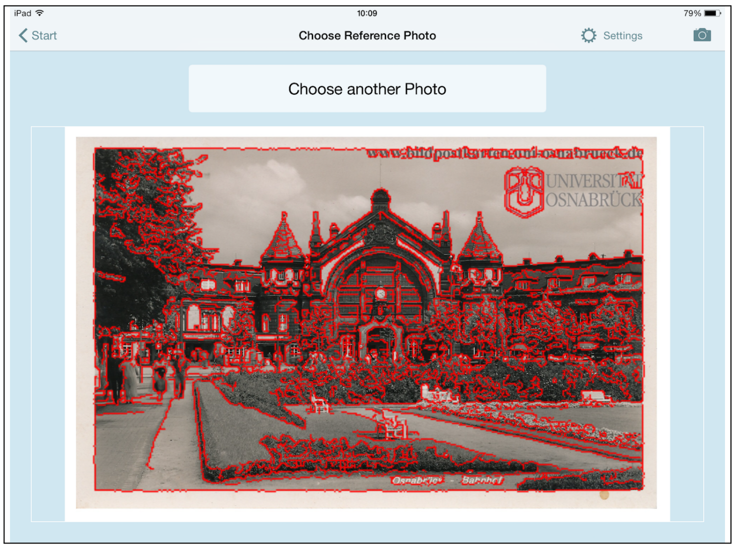


Figure 4.6: PhotoChooserController with loaded image

4.2.3 *SettingsViewController*

The `SettingsViewController` possesses three panels which allow for modification of the colour of the edge overlay and insert authentication data for future use with an envisioned online platform. The third panel is unused in the current version and could be appropriated for other kinds of preferences such as the default type of visualisation for the necessary camera motion (see Chapter 6).

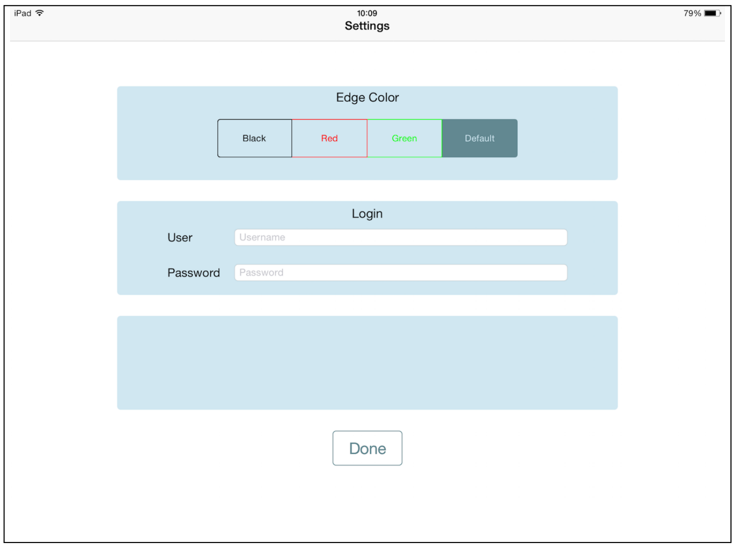


Figure 4.7: SettingsViewController view layout

4.2.4 *RephotoManager*

The two most important classes are the `RephotoManager` and the `PoseDifferenceEstimationOperation`. A `RephotoManager` is attached to a `UIImagePickerController` in camera mode and controls the execution of the rephotograph. An overlay view is added to the camera stream and contains a shutter button and information labels telling the user what to do next. Tapping the info button (ⓘ) expands the label to reveal more information. The manager is realised as a state machine, depicted as a flow chart in Figure 4.8.

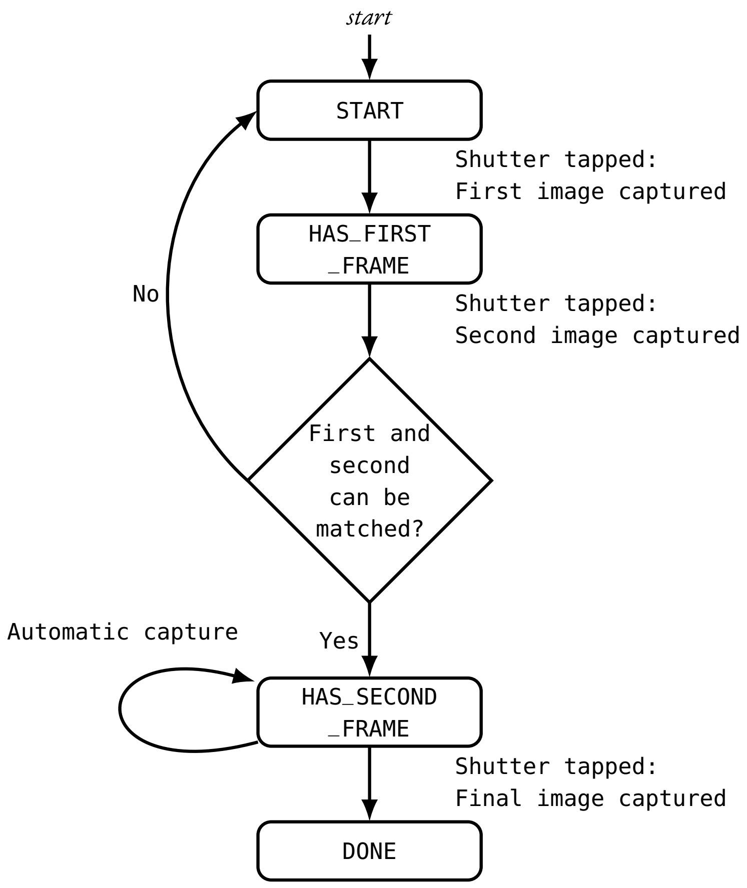


Figure 4.8: States of the rephoto manager object

START

Initially, the user interface presents shutter and cancel buttons in a toolbar at the bottom and an info indicator on the top left, telling the user the take the first frame. When the user taps the shutter button, the first frame is captured and saved.

HAS_FIRST_FRAME

When transitioning to this state, the manager changes the info label accordingly and displays a check mark for the first frame in the centre of the toolbar. Furthermore, at this point the precomputed edge overlay is presented so that taking the second frame which should be reasonably close to the original viewpoint is possible.

HAS_SECOND_FRAME

Once the second frame is captured, the second check mark is shown, the info label changes again and the preprocessing step is attempted. A `PoseDifferenceEstimationOperation` is started with the first and reference frames to obtain $T_{\texttt{ref,first}}$ and $R_{\texttt{ref,first}}$ to be used in equation 3.5. A second operation is run for the reference scale computation by use of first and second frames. Relative translation, rotation and goal scale are saved.

Should one of the computations fail, the user interface resets to the initial state except for an error label at the bottom centre informing the user that the matching failed, underpinned by a red box. Otherwise a timer is initiated that will take pictures at a certain interval which should accord with the hardware running the program. For the tested device, it is set to one second, although somewhat higher frame rates are possible. For each frame captured like this, a `PoseDifferenceEstimationOperation` (see below) is started. The captured frame and the first frame are processed and their relative pose computed. With the information computed during preprocessing, the necessary translation is inferred with equation 3.5.

Meanwhile, a panel at the bottom centre is revealed which visualises this necessary motion of the camera. For this, several means are possible and discussed in Section 6.1. Bae et al. (2010) found that an arrow visualisation for the translation in the XY-plane and Z-direction is most effective so this approach has been chosen here. Two camera pictograms are shown, one back view and one top view. There is one arrow positioned on each so that the translation in the XY-plane and Z-direction are independently visualised.

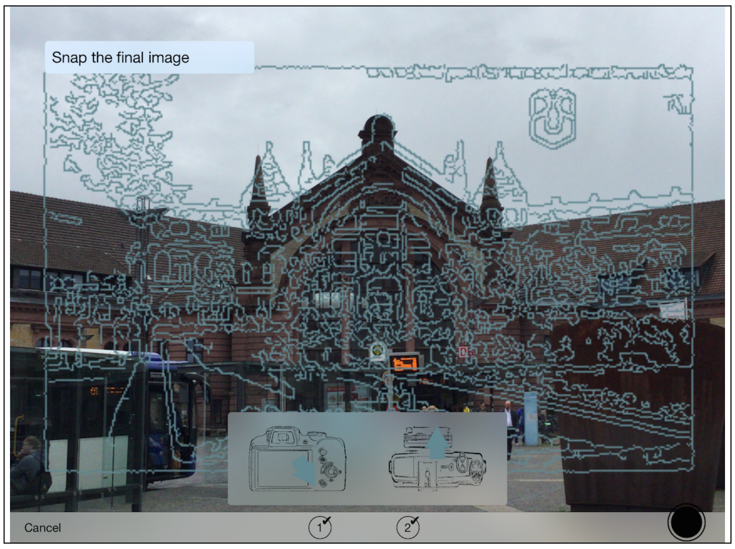


Figure 4.9: RephotoManager after second frame was captured

DONE

Once the final image is taken, the controller informs its delegate that the rephotograph has finished and passes the final capture. The delegate—in this case the `PhotoChooserController`—can then put together an `ImageData` object to present the rephoto to the user.

4.2.5 *ResultViewController*

This controller must be passed a completed rephotograph in an `ImageData` object and displays the new image over the original. With a slider, the new image is horizontally clipped to some percentage of its width and can thus be dynamically unrolled over the original. Depending on a flag in the `ImageData` object, the rephoto passed to the controller will be saved. For this, the app makes use of the `sqlite` library. The `DBManager` class manages communication with a lightweight SQL database. Since all photos are part of the user's photo library—there is no way to save assets locally for an application—they are saved as a tuple of two `NSURL`s.

Both the built-in `UIImagePickerController` and the `ELCImagePickerController` return the URL of a selected image so it can be used to map a rephoto on its original or vice versa when it should be displayed.

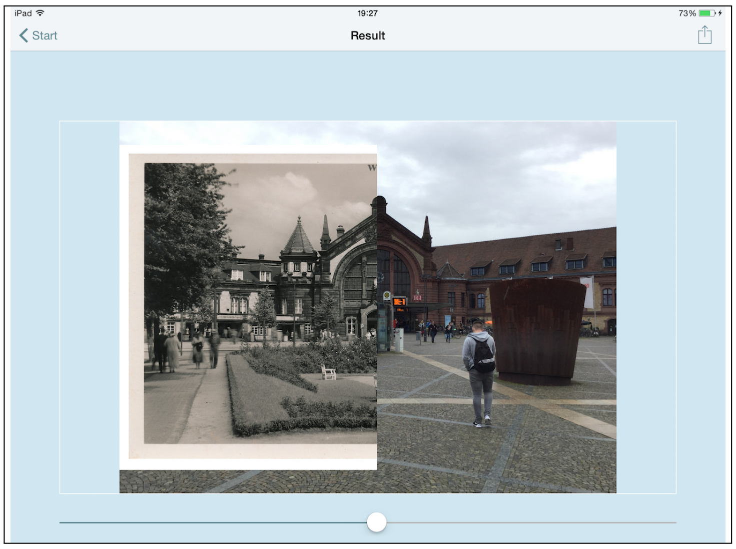
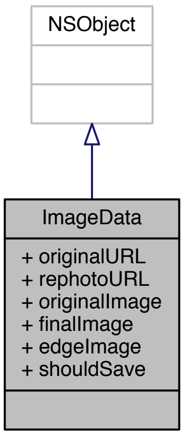


Figure 4.10: ResultViewController view layout

On the top right, a share button (⍐) lets the user upload a rephoto to an experimental test server for display, but the functionality is only rudimentary and requires the server to be running with a known IP address.



4.2.6 *ELCImagePickerController*

For the task of reviewing rephotographs, the `ELCImagePickerController` is used. Contrarily to the built-in `UIImagePickerController`, this reimplementation by Nutting et al. (2013) allows to inspect a specific asset, like a photo album instead of all assets there are. In the album display, the user is shown the new photographs of each rephoto and upon selection is presented a `ResultViewController` with the rephotograph.

4.3 IMPLEMENTATION

This section will survey the more important pieces of the software and elaborate on the particular challenges addressed by them. The implementation of the relative pose estimation is described in detail.

4.3.1 *User Interface*

For the user interface several custom views classes are implemented.

1. `ArrowView` A view which is backed by an `ArrowLayer` and allows displaying it as `CALayer`s cannot be shown without a wrapping `UIView`.

2. `ArrowLayer` A `CALayer` subclass which can display a parametrisable arrow shape whose parameters can be seamlessly animated. `CALayers` are the backbone of all `UIView` objects.

3. `ImageScrollView` A `ScrollView` subclass which allows displaying a zoomable and pannable image while also preserving the visible portion when the user interface orientation changes.

4. `CircleView` A simple view purely for drawing a circle parametrised with colour and line width.

5. `MaskableUIImageView` An `UIImageView` which allows the image to be clipped to some percentage of its width. This is used to display a before-after comparison of the original and repeat photographs.

Furthermore, two view hierarchies and the storyboard belong to this module. If a custom view must exhibit some particular behaviour which necessitates custom methods, creating classes is appropriate. For interface elements which contain a larger hierarchy of nested views without exposing any particular behaviour besides what the contained views can do, specifying them in a visual manner is less involved. Such view hierarchies can be built visually as `XIB` files in an XML format. The `CameraOverlay` overlayed on the current camera picture is created like this, as well as the launch screen shown when starting the app.

4.3.2 *Image Processing*

The bulk of the work is done inside the utility class' `ImageUtils` static methods. To compute the pose difference between two frames, clients can call the public method `+[computePoseDifferenceBetweenCamera:secondCamera:calibrationFile:scale:]`. The two images must be passed, as well as the path to a calibration file in OpenCV's `FileStorage` format. A scale parameter can be used to speed up the computation by downsampling the images. The scale is set to 0.33, compromising between speed and matching robustness. This number takes into account that compared to the computations performed on a computer (see Chapter 5) some detail is lost during conversion between iOS's `UIImage` format and OpenCV's `Mat` type.

Internally, the method converts the images, uses OpenCV image processing functions to compute the pose difference, and the average distance of the matched features to the first camera. The client is returned an `NSDictionary` containing the rotation, translation, average point distance, and—should the computation fail—an error code in which case all other values are invalid.

The method first runs `AKAZE` on both images, the parameters used are shown in Table 4.1. Computation is aborted if one of the images yields fewer than 100 keypoints. Matches are found by brute force with a ratio test as suggested by Lowe (2004). Let $d(x, y)$ denote the distance (here the $L2$-norm) of descriptors x and y. For all descriptors x in the first image, the two best matching descriptors y_1 and y_2 in the other image are found. If

$$\frac{d(x, y_1)}{d(x, y_2)} < \rho$$

Parameter	Value
`nOctaves`	4
`threshold`	0.001
`nOctaveLayers/sublevels`	4
`descriptor_size`	486 bits[8]
`descriptor_channels`	3

Table 4.1: Parameters used for AKAZE (number of octaves, detector response threshold, levels per octave, size of the descriptor and the number of channels, see Subsection 3.2.1)

Parameter	Value
`nOctaveLayers/Sublevels`	3
`contrastThreshold`	0.04
`edgeThreshold`	10
`sigma`	1.6

Table 4.2: Parameters used for SIFT (number of octaves, number of sublevels per octave, detector response threshold, edge threshold for filtering edges, standard deviation of the Gaussian for the initial image)

then the pair (x, y_1) is chosen as a match. Lowe suggests $\rho = 0.8$, the app uses a stricter ratio of $\rho = 0.7$. This test will filter unstable matches between e.g. repeating patterns in an image (windows, wall structure etc.) as for such a point the corresponding point is very ambiguous and its best matches will be similar in distance.

The function then removes the image distortion (see Subsection 2.1.2) on the sparse set of keypoints instead of the whole image for efficiency.[6] On the correspondences, RANSAC and the five-point algorithm are used to find the best-fitting essential matrix. The confidence threshold is set to 0.999, a point is considered an outlier if its distance to its epipolar line exceeds 3 pixels.

Once E is fixed, the optimal triangulation method (see Hartley and Zisserman (2004, ch. 12.5.2)) is used to correct the corresponding points. Since one assumes the computed essential matrix is correct, the information can be used to refine the position of corresponding points. This should make pose recovery with E and the points more accurate.

The essential matrix is then decomposed into R and T. A sanity check for a valid rotation matrix is performed by checking that $|\det R| \approx 1$.[7]

6 Since undistortion requires resampling of the image, it is better to find features beforehand as some information may be lost by interpolation.

7 A rotation matrix is orthogonal and orthogonal matrices' determinants are ± 1

8 `AKAZEFeatures.cpp line 720`, commit `09b9b0f`

Category Name	Function
`NSMutableURLRequest (RephotoUpload)`	Adds method to post a rephoto to an experimental server via HTTP
`ALAssetsLibrary (CustomPhotoAlbum)`	Adds method to add an image to a specific album, not possible per default. Used to save rephotos to the `Rephotos` album
`NSUserDefaults (UI Color)`	By default, only some types can be persistently saved in the user defaults. Adds method to save colours. Used to remember the user's edge colour.
`CALayer (UIColor)`	Properties for view objects can be set in Interface Builder, but only if they are of object type. To set a view's backing layer's border colour outside of the code, a colour property of object type is defined, mapped to the C type `CGColor` used by `CALayer`.
`UIBezierPath (Arrow)`	Adds a method to generate an arrow shape with a bézier curve. Used for displaying user guidance.
`UIImage (Zoom)`	Adds a method to crop an image to a rectangle centred on the image centre, effectively zooming in. Used to allow the user to zoom the camera image.
`UIScrollView (Center)`	Adds methods to centre the content and zoom to a particular size. Used for display of the original image.
`UIColor (CustomColors)`	Contrarily to Android, iOS possesses no Resources framework, so this Category declares some commonly used colours for the user interface
`UIView (Effects)`	Adds some animation methods to all views to avoid repeating boilerplate code.

Table 4.3: Summary of categories

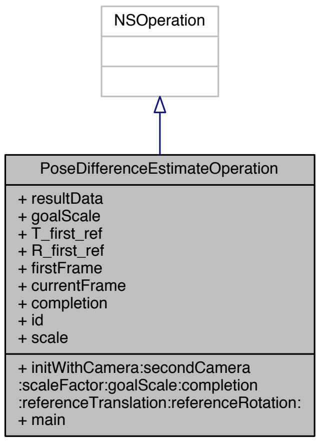
4.3.3 *Categories*

Code reuse in Objective-C can be accomplished by inheritance as in other object-oriented languages. But when only one piece of behaviour is to be added, not overridden[9], the lightweight Category concept is often employed instead. A Category on a class adds methods or properties to that class. The header file declaring the category can be included by all clients that wish to make use of the modified or added behaviour, and left out by all others. This way, no unnecessarily large class hierarchy is created.

In this application, categories are used for different purposes. They are summarised in Table 4.3.

9 The behaviour is undefined when a Category method has the same signature as one of the class (Apple Inc., 2014a)

4.3.4 *Computation Of Necessary Translation*


NSOperation
PoseDifferenceEstimateOperation
+ resultData
+ goalScale
+ T_first_ref
+ R_first_ref
+ firstFrame
+ currentFrame
+ completion
+ id
+ scale
+ initWithCamera:secondCamera
:scaleFactor:goalScale:completion
:referenceTranslation:referenceRotation:
+ main

The computation of the necessary translation is implemented by the `PoseDifferenceEstimationOperation` class and was conceived to allow parallelisation.

iOS supports different APIs for concurrency. Besides manual thread creation, Grand Central Dispatch allows to easily dispatch blocks (function objects) to different kinds of dispatch queues. The runtime automatically removes blocks from the queues for execution. Dispatch queues can be serial or concurrent and both synchronous and asynchronous dispatch is possible as well as synchronisation, but the API is not object-oriented and some useful features are unavailable. It is not possible to specify how many blocks should be executed in parallel or to cancel tasks after submission. Since the application should make optimal use of the hardware due to high computational demand while not queueing up many tasks or tasks which are obsolete, such functionality is desirable.

Operation queues—internally implemented with GCD[10]—sacrifice some efficiency, but expose a more high-level object-oriented interface in which `NSOperation` objects are submitted to `NSOperationQueue`s and their `maxConcurrentOperationCount` property limits the maximum number of concurrent operations. The application sets this limit to 3 for an iPad (assuming 3 cores like an iPad Air 2) and 2 for iPhones, as the current devices possess two cores. Since there is no concurrent modification of data and tasks are started in slow intervals, explicit synchronisation is not needed.

A `PoseDifferenceEstimationOperation` is created with two images (one of which will always be the first frame), a scale parameter for downsampling, and computed values for the relative pose between first and reference frames and the goal scale. Furthermore, a completion block is passed which can accept three `float` parameters—one for the direction in the XY-plane, one for the ratio between reference and current world scale, and one for the magnitude of translation along the optical axis. The intended purpose of this block is to drive some kind of visualisation. The operation then computes the relative pose and the necessary translation as in equation 3.5.

If $T_{\text{current,ref}} = (x, y, z)$[11], then the vector direction in the XY-plane is given by

$$\alpha = \text{atan2}\,(y, x).$$

Let d_0 be the average distance of scene points to the first frame's camera, computed with the second frame, and d_i the same with the current frame. The ratio of scales $r = \frac{d_i}{d_0}$ is used to scale the translation vector. d_i increases with decreasing distance to the first frame (see Figure 3.2), which in turn (usually) means it should decrease

10 (Apple Inc., 2014b)

11 One should note that this calculation depends on the coordinate system. In OpenCV, the origin is at the top left corner of an image, the positive x-axis is the image width, the positive y-axis the image height and points downward. In a more canonical representation, the y-value thus may be flipped

when approaching the target, since when done right, motion away from the first frame is motion towards the original viewpoint.

The completion block is called with z, α and r.

4.4 HARDWARE

The application has been developed on an iPad Air 2. This model features an A8X processor with three cores and 1.5GHz per core.[12] The number of concurrently running `PoseDifferenceEstimationOperation`s is thus set to three which enables frequent updates of the visualisation. Images with 1077×807 (full resolution with a 0.33 scale factor) take roughly 0.7 seconds to process, so that updates can be delivered more than once per second. The concurrent operation count is set to two for iPhones owing to their smaller processing power and number of CPU cores.

12 The number is not official and based on empirical tests (Primate Labs Inc., 2014)

5 EVALUATION

The approach has been evaluated on two realistic datasets which can be found in Appendix B. Computation was performed off-line on a computer with the same C++ code running on the mobile device. The most important questions are whether the direction of the necessary translation is correctly identified and its scale decreasing with distance to the target. For both sets of images, the ground truth translation between each image and the first frame has been measured with centimetre accuracy, while the ground-truth rotation has been estimated from manually labelled correspondence as it is difficult to measure without the proper instruments. For the case of noise-free correspondences in a non-degenerate configuration, relative pose estimation algorithms are mathematically correct. Therefore estimating the true rotation like this has been deemed sufficiently accurate to evaluate the procedure. For each image pair, 19–27 correspondences have been labelled, of which the majority is used for pose recovery. For pose recovery, RANSAC is used in conjunction with the five-point solver, a point is considered an inlier for a given essential matrix if its distance to its epipolar line is no more than three pixels, the confidence threshold is 0.999. These parameters lead to the majority of points being inliers of the pose recovery, the few outliers can be explained by imprecise labelling.

In both data sets, the translation was mostly in the horizontal direction and along the optical axis; the vertical translation is thus neglected. Similarly, rotation was applied mainly around the vertical axis.

In order to idealise the condition, the reference photograph has been used to fill the role of the second frame for world scale computation. In reality, since the reference location is unknown, the reference world scale is obtained from a position somewhat off, thereby decreasing the accuracy of scale estimation.

The scale at which the relative pose is computed is referred to as s in all plots. Besides the full resolution, the images are scaled down by a factor of 2 (both dimensions are halved, resulting in quarter size) and 4 (sixteenth size). The resolutions evaluated are thus 3264×2448, 1632×1224, and 816×612.

The following graphics illustrate three things.

1. The difference between the computed necessary rotation and the actually necessary one
2. The difference in direction of the computed necessary translation and the actually necessary one
3. The correlation between the true ratio of distances obtained by measuring camera movement and the average distance ratio computed with first and second (or in this case reference) frames based on automatic feature matching, at three different scales

Table 5.1: Ground truth for the train data. Image 0 is the reference frame, translations and rotations are given as in equation 2.11 relative to the reference frame.

Image number	Relative translation $[x, y, z]$	Relative Rotation $[\theta_x, \theta_y, \theta_z]$	ratio
0	$[0, 0, 0]$	$[0, 0, 0]$	1
1	$[.9053, 0, .4246]$	$[-3.3779, -9.3779, 1.05121]$	3.8936
2	$[.9986, 0, .0512]$	$[-1.3274, -5.7134, -.1884]$	1.6461
3	$[.9993, 0, .0361]$	$[-1.7156, -2.4761, .3469]$	1.0965
4	$[.9950, 0, .0995]$	$[.054606, -4.4867, .2452]$	1.0343

5.1 TRAIN STATION DATA SET

In this series, the camera was moved horizontally from the reference to the right while also coming closer to the building. A schematic bird's eye view of the captures is shown in Figure 5.1. Table 5.1 summarises the ground truth for the five images.

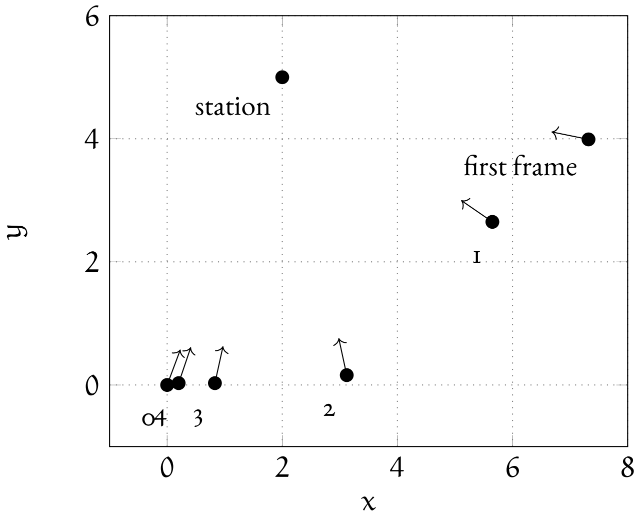
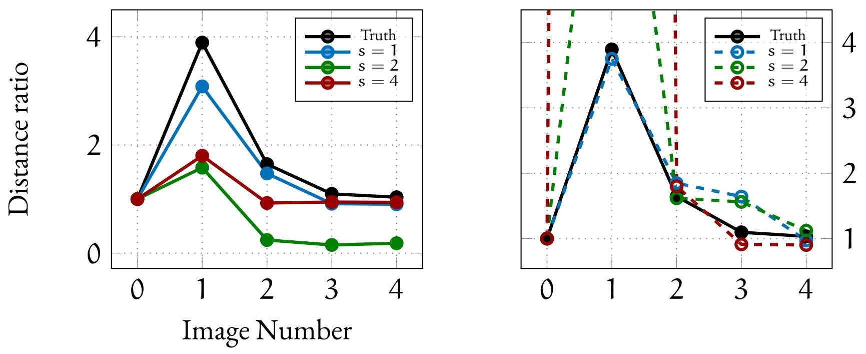


Figure 5.1: Schematic representation of the Train Station data set. Lengths and angles are not precise.

5.1.1 *Scale Estimation*

Figure 5.2 shows how the average distance of points to the first frame's camera varies with the second image used for triangulation. The plot illustrates that—especially at full resolution—the ratio based on feature matching closely mirrors the real value. The difference increases with decreasing image scale, but the slope of the graphs is quite similar. This shows that indeed with increasing distance to the first frame, the ratio decreases. This allows a deduction as to how close the camera is to the target, at least with respect to previous iterations, which is the primary objective. The decrease in ratio closely correlates with the decrease in distance which is apparent on inspection of Figure 5.1. For instance, the viewpoints 3 and 4 are much closer together than e.g. 2 and 3, and the difference in ratios is much smaller between 3 and 4 as well.



(a) AKAZE features

(b) SIFT features

Figure 5.2: Train Station data set: Evolution of the distance ratio between images

The correlation is higher for AKAZE features than for SIFT ones, where a strong spike for image 1 can be observed. For SIFT, the decrease of ratio between images 2 and 3 is also hardly visible at $s = 2$. The unusual spike for image 1 poses the problem that the visualisation would tell the user that they need to move disproportionally far compared to the other images. Since the error in this case is confined to image, this may not be a big problem, but will affect user experience. For SIFT features, one can also observe that the scale of the images appears to be less relevant, possibly an indication for the better scale invariance of the descriptor.

Generally, it can be concluded that on this data set, AKAZE features are an appropriate means of estimating the scale of relative camera translation.

5.1.2 *Rotation Estimation*

Figure 5.3 and Figure 5.4 illustrate the difference between the actually necessary camera rotation and the computed one for AKAZE and SIFT features, respectively. Rotations about the optical and X axes are small and thus not very interesting and the deviation is small.

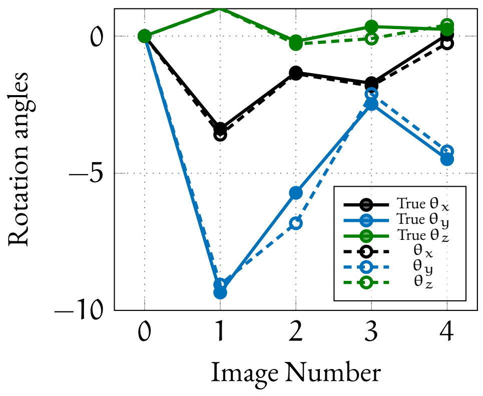
Focusing on the Y-rotation, it is obvious that the estimation quality decreases especially for $s = 4$, but the difference does not exceed 5 degrees and thus the estimate is usable. In particular, for reasonably quick updates mostly the direction of necessary rotation is important, not the absolute magnitude.

The performance of SIFT is even better for scales $s = 1$ and $s = 2$, but slightly worse on the smallest scale (see Figure 5.4c).

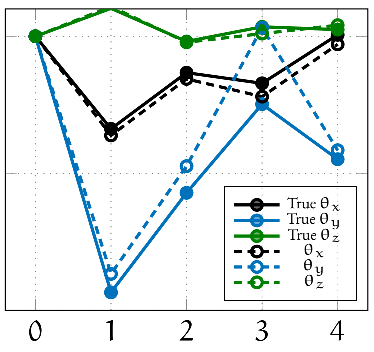
5.1.3 *Translation Estimation*

Finally and most importantly, the directions of the necessary translation must be evaluated. Figure 5.5 plots the angular difference in degree between the actual necessary translation and the computed one. The reference frame 0 is omitted since its translation relative to itself is $(0, 0, 0)$.

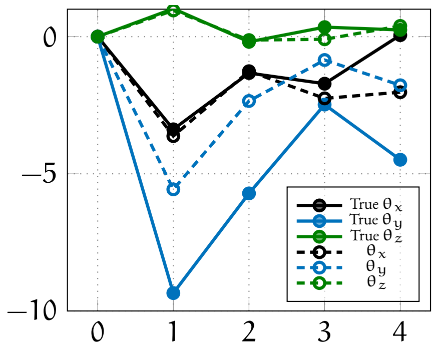
It is obvious that the estimates are completely useless, the difference exceeds 80 degrees in all cases. With these estimates, the user will be sent into an entirely wrong direction. For an explanation of this failure refer to Section 5.3.



(a) $s = 1$

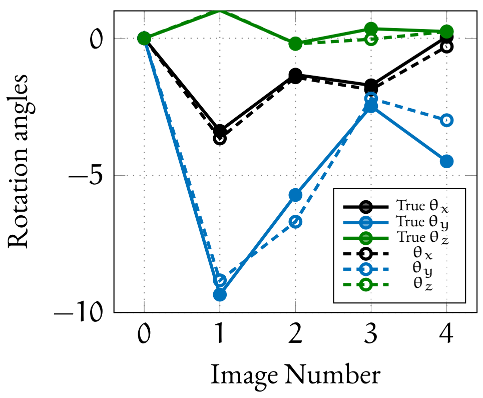


(b) $s = 2$

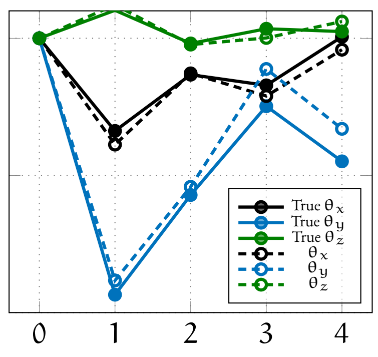


(c) $s = 4$

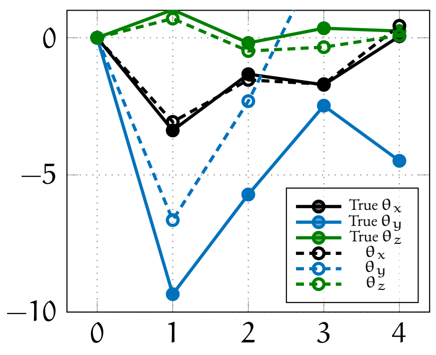
Figure 5.3: Train Station data set: Angles of rotation relative to reference with AKAZE features on full, quarter and sixteenth resolution



(a) s = 1

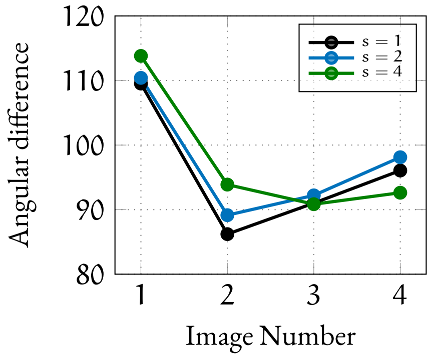


(b) s = 2

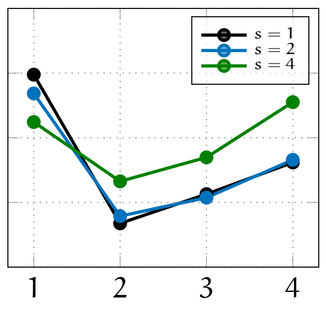


(c) s = 4

Figure 5.4: Train Station data set: Angles of rotation relative to reference with SIFT features



(a) AKAZE features



(b) SIFT features

Figure 5.5: Train Station data set: Angular difference between actually necessary translation and algorithmic estimate

5.2 MANOR DATA SET

Seven images (including the reference photo) have been taken with movement to the right and backwards as well as forwards. The motif was always centred in the frame, thus there is prominent rotation around the y-axis. The schematic positions for the seven manor captures (including reference photograph) are shown in Figure 5.6 (note that here the image number decreases with distance from the first frame).

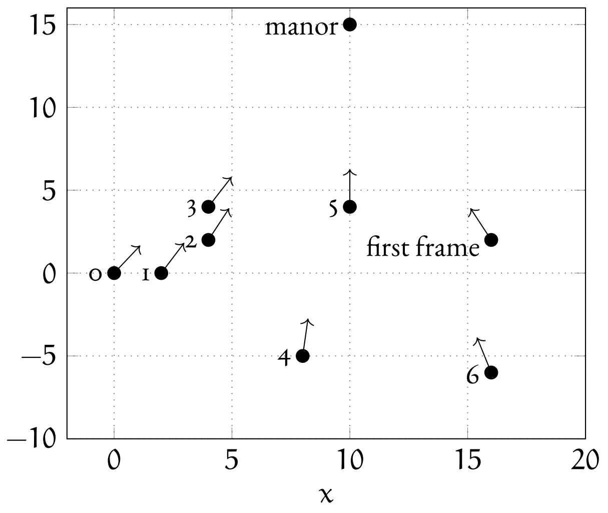


Figure 5.6: Schematic representation of the Manor data set. Lengths and angles are not precise.

In contrast to the train station set, there is more significant movement along the optical axis. The ground truth data is summarised in Table 5.2.

Image number	Relative translation $[x, y, z]$	Relative Rotation $[\theta_x, \theta_y, \theta_z]$	ratio
0	$[0, 0, 0]$	$[0, 0, 0]$	1
1	$[1, 0, 0]$	$[-1.7857, -5.4827, 2.1073]$	1.1401
2	$[0.8944, 0., 0.4472]$	$[-2.1428, -6.5773, 1.6584]$	1.3437
3	$[0.7071, 0., 0.7071]$	$[0.7263, -5.0686, 2.6176]$	1.3254
4	$[0.8479, 0., -0.5299]$	$[-1.4146, -10.7998, 2.2250]$	1.5168
5	$[0.9284, 0., 0.3713]$	$[-0.1887, -16.6670, 1.2211]$	2.5495
6	$[0.9363, 0., -0.3511]$	$[-0.8725, -18.0933, 1.5385]$	2.0155

Table 5.2: Ground truth for the manor data. Image 0 is the reference frame, translations and rotations are given as in equation 2.11 relative to the reference frame.

5.2.1 *Scale Estimation*

The evolution of the translation scale is shown in Figure 5.7. It is apparent that the movement purely along the optical axis between images 2 and 3 is a problem. As the real distance to the target marginally increases, so should the ratio, but it decreases instead. Frames 5 and 6 illustrate a problem with the scale estimation

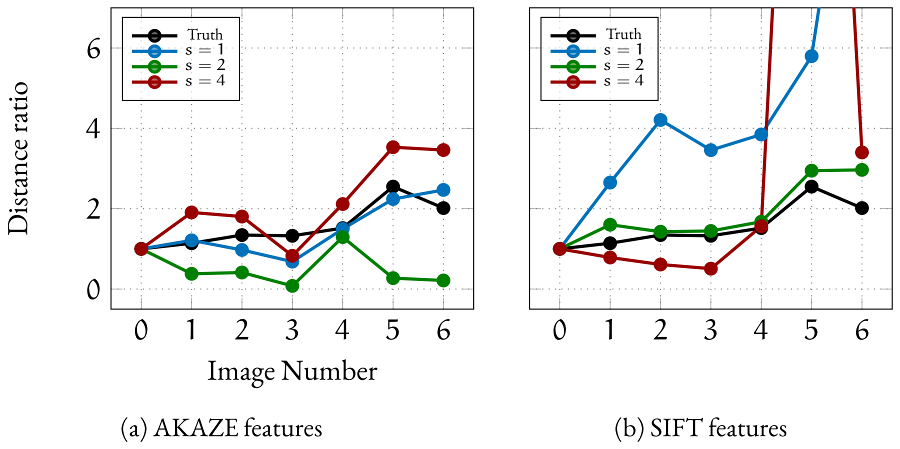


Figure 5.7: Manor data set: Evolution of the distance ratio between images

procedure itself. For it to work precisely, only movement along the line between first and reference frames is assumed, as a decreased distance to the first frame is interpreted as an increased distance to the reference frame, which is not necessarily the case as shown here (see Equation 3.4). Even the "ground truth" computed from actual camera distances is thus of limited use. In reality, the assumption is justified that the user will move between first, second and reference locations in a more or less straight line, with more movement to the sides than front or back.

For the AKAZE descriptor, only the full resolution comes reasonably close in magnitude and somewhat in slope. With SIFT, the slope is more accurately reproduced with $s = 2$, but strangely less accurately on full resolution. It is possible that the reduction in noise brought about by downsampling can improve the estimate, but the other experiments do not show it. For the smallest scale, the estimate degenerates strongly.

In general is can be stated that the estimates are less close than those for the train data set, but also that large movement along the optical axis shows the limits of this simple approach at scale estimation. Realistically, the user will not move as erratically so this kind of scenario is extreme.

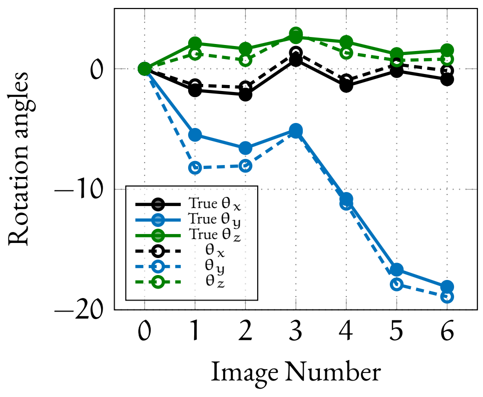
5.2.2 *Rotation Estimation*

Figure 5.8 and Figure 5.9 illustrate how accurately the necessary rotation is computed. On this data, AKAZE outperforms SIFT with default parameters (see Table 4.1). On both full and half scale, there is negligible deviation from the truth, but on quarter scale, there are more than 5 degrees of difference and a complete failure for frame 3 (the direction is wrong, not only the magnitude).

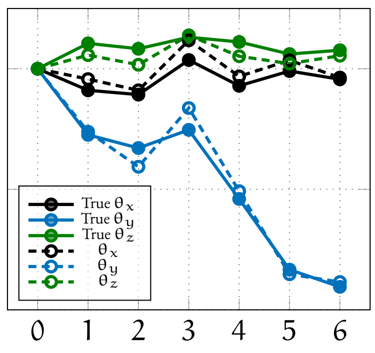
With default parameters (Table 4.2), SIFT compares much worse, particularly on full resolution where it grossly overestimates the necessary rotation. The results are better on the scaled-down images, possibly because of the reduction of noise, but still only partly useful on the smallest resolution.

5.2.3 *Translation Estimation*

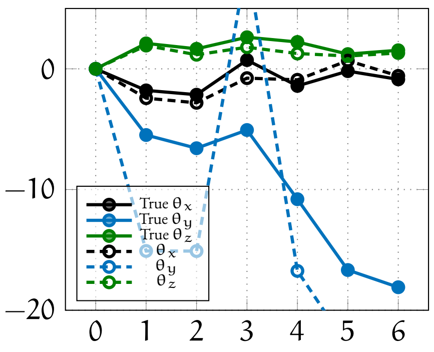
Lastly, the direction of necessary translation is evaluated in Figure 5.10. It is moot to discuss any improvement in comparison with the train data set, as the results are



(a) s = 1

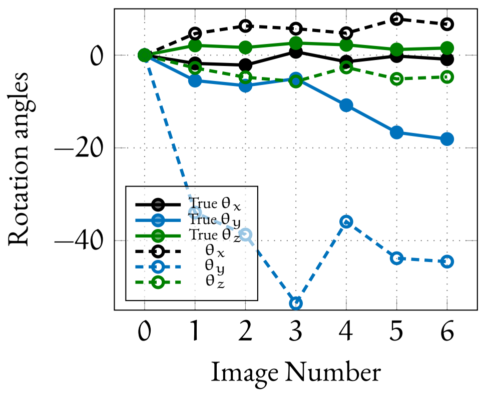


(b) s = 2

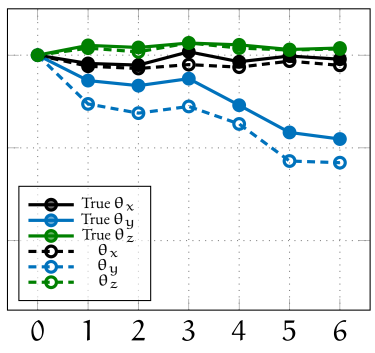


(c) s = 4

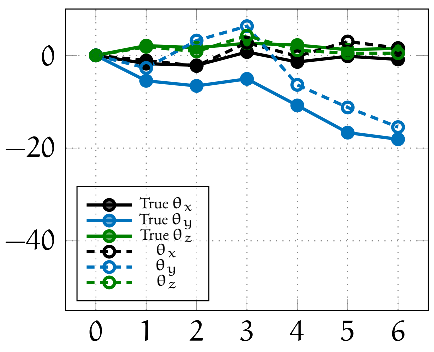
Figure 5.8: Manor data set: Angles of rotation relative to reference with AKAZE features



(a) s = 1



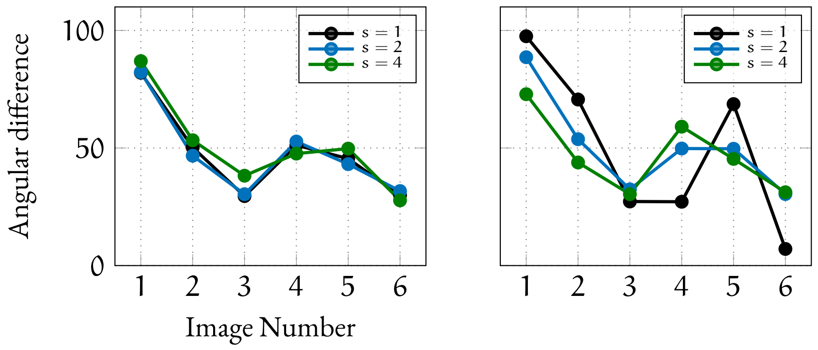
(b) s = 2



(c) s = 4

Figure 5.9: Manor data set: Angles of rotation relative to reference with SIFT features

also completely false, SIFT displaying a larger variance than AKAZE, but neither are useful.



(a) AKAZE features (b) SIFT features

Figure 5.10: Manor data set: Angular difference between actually necessary translation and estimate

5.3 SUMMARY

Of the three pieces of information needed for user guidance, only the most important one—the direction of translation—cannot be recovered to any satisfying degree with this method. Both scale and necessary camera rotation estimation work, at least if the movement over iterations is mostly horizontal and not along the optical axis. A principal problem with the estimation of necessary translation could be observed. If the movement is mostly in one direction and the translation between reference and current frame is computed as in equation 3.5

$$-R_{\text{ref,first}}R^{T}_{\text{current,first}}T_{\text{current,first}} + T_{\text{ref,first}}$$

then both summands will have mostly the same orientation which will consequently be zeroed out by the sign inversion of the first one. If the resulting vector is then normalised to unit length to be scaled with the appropriate factor, the other two dimensions will have nonzero values. These are determined by small differences in the vectors' orientation and thus by noise and hence the sum will point into a completely wrong direction. It is evident that this solution cannot work for movement in only one principal direction as will most often be the case in rephotography.

Two possible approaches to circumvent this issue come to mind.

1. One could scale each summand before the addition with the inverse of its respective world scale (that is the scale computed with current and first frames, and the scale computed with reference and first frames). As the world scale (computed as the average distance of points to the camera) is inversely proportional to the camera distance and thus the length of the translation vector, the summands may be scaled to better represent reality.

2. The comparison of object distances across iterations is an indirect measure for the camera distance and was shown to be fallible when the movement deviates strongly from the line between first and reference frame locations. Another idea is to relate all iterations via the computed 3D coordinates of

the scene objects. For this, 3D coordinates can be triangulated from the first and second frames to create a scene representation in the first frame's coordinate system. Then, assuming the object coordinate system is the first frame's coordinate system, the *Perspective-n-Point* (PnP) problem can be solved with the object points and their corresponding image points in the reference frame (i.e. their projections into the first frame). The problem consists in finding a camera pose from 3D-2D-point correspondences and can be solved for instance with Levenberg-Marquardt optimisation of the projection matrix $P = [R \mid T]$.[1] Thus one obtains $T_{\text{ref,first}}$ with some scaling—not necessarily unit length. The procedure can be carried out in each iteration with the precomputed object points and their correspondences in the current frame, obtaining $T_{\text{current,first}}$. This will require caching the first frame's keypoints to keep them constant, just as the object points are constant. Since the 3D scene coordinates stay the same, a consistent scale of the translation vectors should be the result.

Both approaches will be evaluated in future work.

It could also be demonstrated that AKAZE features yield the more accurate results, except on the smallest scale, where SIFT compares somewhat favourably. The smallest scale however also leads to general deterioration in quality, suggesting that a scale factor between 2 and 4 may be required to combine accuracy with speed of processing.

Improvements for all estimates could possibly be achieved by fine-tuning the parameters of both descriptors to adapt them to scenes with buildings, which has not been tried here.

1 OpenCV supplies the `solvePnP` function for this purpose, implementing various algorithms including the LM-optimisation or more efficient solvers like (Lepetit et al., 2009)

6 APPLICATION EVALUATION, OUTLOOK & FUTURE WORK

6.1 EXTENSIONS

Historic Images

Some features from Bae et. al's work have been simplified away, so that currently, no really historic photograph can be used. The app should therefore be extended with another view controller presented after first and second frames have been taken, in order for the user to label correspondences. These correspondences can be used to fit a homography between the two views and warp one of them accordingly.

Real-time User Guidance

A higher rate of processing could be achieved by also using fast feature tracking between robust pose estimations where the AKAZE detector is run. Bae et al. use Lucas-Kanade-Tomasi tracking for this (Lucas and Kanade, 1981; Tomasi and Kanade, 1991), an OpenCV implementation is already available. During automatic capture, it can be observed that the found features often change abruptly between successive frames. This and the fact that the number of found features changes strongly results in the scale estimation jumping rapidly. A feature tracking may increase the stability as the already-found points are found again, instead of radically different ones.

Rephoto Postprocessing

It is likely that despite all assistance, the user will not manage to perfectly line up the original photograph with the current camera picture. In order to better register the original and final images, the latter can be warped according to the last pose difference estimate. Bae et. al use an infinite homography during the homing process, but the same method can be used after the final capture. The infinite homography is the one between two views induced by the plane at infinity, i.e. the set of all projective points $(x, y, z, 0)$ and maps vanishing points in one view onto vanishing points in the other. Details can be found in (Hartley and Zisserman, 2004, ch. 13.4). Given the calibration matrices of both views and their relative rotation, the homography can be computed and one image can be warped with it by remapping the pixels.

Other Visualisation Options

Bae et al. (2010) explored different ways of guiding the user. A side-by-side view or a linear blend with the original photo were less helpful than the arrow visualisation, but one could imagine offering different kinds, such as a three-dimensional arrow which shows both parts of the necessary translation. A more sophisticated idea is to project an approximation of the correct location into the camera picture, but this would require a more detailed reconstruction of the scene.

Online Platform

It is envisioned to create an online community where rephotographs can be uploaded by users. There are web applications dedicated to rephotography already, for example *Third View*, a project directed by Mark Klett dealing mostly with repeated captures of geographical landmarks, not man-made structures (Klett et al., 2004). However, it is a documentation project and not community-driven. In contrast, the aforementioned Timera app also comes with a community where users can upload their rephotos which are then presented in the same fashion as in the app itself, or old photos for others to use as a reference.

A platform letting users upload their pictures is also the subject of another thesis by Weber (2015). The web service developed in this work sports the following features.

- Users can upload, rate and comment on rephotographs.
- Rephotos can be viewed in detail in an overlay and a slider masks the images just as in the iOS app presented here.
- Rephotos can be tagged with metadata, including time, location or title.
- Rephotos can be filtered by timespan between the two images, the geographic location and the category.
- A map displays the locations for which users have shared their rephotos.
- The two images can be registered in the web form via identification of corresponding points. With OpenCV, a homography is estimated to better register the images.

The web service supplies an application programming interface of which the mobile app should make use in the future. Rudimentary provisions have been made, but the authentication must be implemented and the HTTP request adapted to interface with the server.

6.2 USABILITY & PERFORMANCE

The focus of this work was the theoretical exploration and evaluation of Bae et. al's approach to computational rephotography, not the design of a release-ready application. As such, some functionality is missing and the app is not extensively tested. Notable shortcomings are the following.

1. In the gallery inside the app, the final captures of each rephoto are shown as thumbnails. Should multiple rephotos be based on the same original, they will be indistinguishable. The thumbnails should be made more meaningful.
2. Deletion of rephotos works via the user's gallery, where the rephoto album created by the app is visible. Deletion will leave invalid entries in the

database which should be cleaned up by the app. More reasonably perhaps, deletion should be offered inside the app.

3. The upload of rephotos is experimentally hard-coded to work with a test server and is a blocking request.

4. To enhance user experience, a choice between different overlays should be offered, where at this time only an edge overlay is implemented. The edge overlay could also be made more helpful by post-processing it into a more distinctive subset of lines to remove noisy clutter when fine-grained patterns are present (e.g. vegetation).

5. All frames are matched with the first frame, so it would suffice to compute the keypoints once and reuse them. Currently, the first frame's keypoints are recomputed for every relative pose estimation.

6. The application requires iOS 8, but actually has few dependencies on it. The user interface employs `UIVisualEfffectView`s which were previously unavailable. It should be fairly straightforward to create versions of the UI without these elements.

7. Currently, the calibration data cannot be loaded from a user-supplied file. Ideally, since not all devices will have the exact same camera characteristics, each user would have to individually calibrate their camera.

8. The rephotography makes use of the simple `UIImagePickerController` which is presented modally. This makes the flow of the app less seamless. For instance, when the `PhotoChooserController` is presented and the user loads a reference image, the image picker will pop up, but upon image selection, one is not immediately shown the camera, but is briefly shown the photo chooser again before the rephoto starts with yet another `UIImagePickerController` (this time in camera mode instead of gallery). Furthermore, the possibility to modify the camera stream is limited to affine transformations, which precludes projective transformation. This would be necessary to implement the image warping used by Bae et al. (2010). Also, images are captured by a timer. It would be more intuitive to attempt to process frames as they come and simply drop them if no CPU time is available which would also scale more easily to other devices. Both could be accomplished by rebasing the app on the much more customisable `AVFoundation` framework which gives more low-level access to the device's camera at the cost of significantly more development effort. For this, OpenCV wrappers exist already which would simplify processing the frames.

9. The application has only been tested on a single device. While it should work on iPhones as well, this could not be tested and some tuning may be necessary.

10. The application creates a photo album for the rephotos with the name *Rephoto-Album*, without checking if it already exists.

11. Every finished rephotograph is saved. There should be an option to discard an attempt in the result view.

7 CONCLUSION

This thesis presented an attempt at retrieving the location of a photograph with the help of image processing. After introducing the basic geometry underlying world-to-image mapping and the geometry relating two views of a scene, a computational approach developed by Bae et al. (2010) has been presented. Structure-from-motion is used to compute the motion between two captures of the same scene and infer the original viewpoint from the reference photo and two images of the current scene. The theoretical and practical problems have been highlighted, including such scenes which do not allow determining the relative pose of cameras viewing them and how to acquire matching points in two images. Algorithms to solve the pose recovery problem have been introduced.

Evaluation on two real scenes revealed that only two elements—the necessary camera rotation and the closeness to the goal—show promising estimates with this approach. The direction of necessary translation could not be recovered with this method. Since this aspect is not further documented in Bae et. al's work, it warrants further investigation.

An implementation of most of the ideas as a mobile application has been prototyped and described, being the first of its kind despite not being as functional as intended. The application requires the user to load an image, instructs them two capture two images of the scene and then visualises an estimate for the necessary motion with two arrows, one for the sensor plane and one for the optical axis. Further assistance is provided by overlaying the edges of the original image over the current camera picture, making the application useful despite the failure of the arrow visualisation. When the final image is captured, the app lets the user review the rephoto. A slider is used to mask the old over the new image and the change of the scene can be dynamically visualised. Rephotos are saved to the gallery to be viewed at a later time.

It must be concluded that the original objective could not be achieved in its entirety and the application is currently not more useful than existing approaches. With an edge overlay, it can still help a user in capturing rephotographs, although they still need to determine the necessary motion on their own. There is hope however, that a solution to the translation problem can be found and implemented in the future, increasing the capabilities of the app.

A

APPENDIX A — BUGS

Some bugs in the SDK have been discovered in the making of the software.

1. When using `UIImagePickerController` in camera mode with disabled camera controls—necessary to provide custom controls—zooming and then rotating the device will create a black bar at one side of the image. Rotating back will remove it but hide the otherwise visible zoom level indicator. It is necessary to implement a manual zooming and apply an appropriate transform to the camera stream. The bug ID is 20992021. The issue is claimed to be fixed in iOS 9.[1]

2. When all images from the app-created rephoto album are deleted, the album will not be shown in the gallery anymore, but `ALAssetsLibrary` does not permit adding images to it or recreate an album of the same name. Since developers are encouraged to move to the more recent `Photos` framework, this problem is not likely to get fixed.

3. It is not feasible to support background and foreground transitions while using a timed capture on `UIImagePickerController` as has been done here. If the app becomes inactive while an image is being captured, it appears as if sometimes the data is released and therefore unavailable when transitioning back to the foreground. This results in an exception being thrown, which the developer can do nothing about, except avoid `UIImagePickerController` for timed capture. The bug ID is 19953748. The issue is claimed to be fixed in iOS 9.[2]

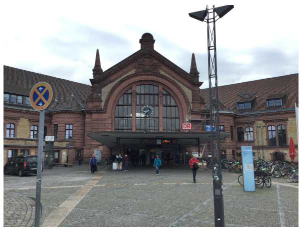
1 Correspondence with Apple
2 Correspondence with Apple

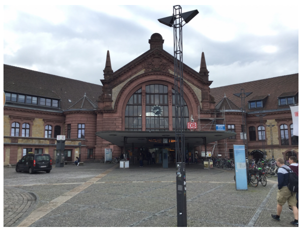
B

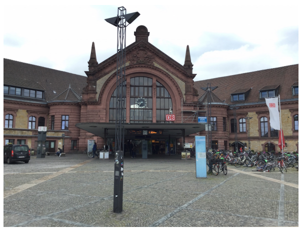
APPENDIX B — DATA SETS

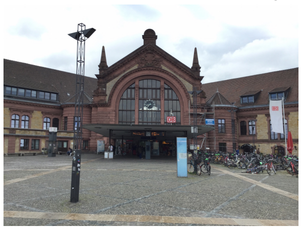
The images used for the evaluation in Chapter 5 can be found here. The captions relate the images to their position in the bird's eye view from that chapter.

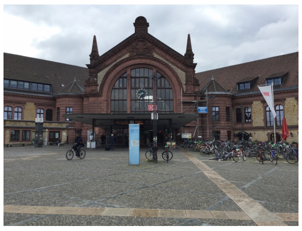
B.1 TRAIN STATION DATA SET


(a) 0: The "reference" image


(b) 4


(c) 3


(d) 2


(e) 1

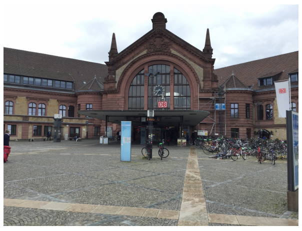

(f) First frame

Figure B.1: Images in the train station data set

B.2 MANOR DATA SET

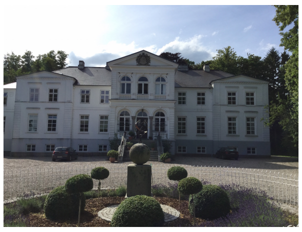
(a) 0: The “reference” image

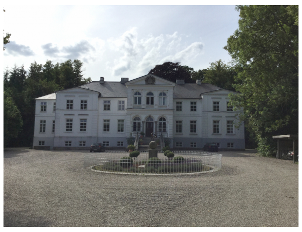
(b) 1

(c) 2

(d) 3

(e) 4


(f) 5


(g) 6

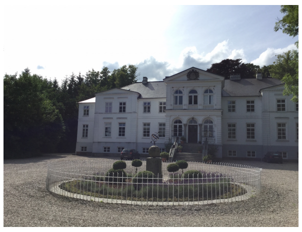

(h) First frame

Figure B.2: Images in the manor data set

BIBLIOGRAPHY

Adams, A., Talvala, E.-V., Park, S. H., Jacobs, D. E., Ajdin, B., Gelfand, N., Dolson, J., Vaquero, D., Baek, J., Tico, M., Lensch, H. P. A., Matusik, W., Pulli, K., Horowitz, M., and Levoy, M. (2010). The Frankencamera: An Experimental Platform For Computational Photography. *ACM Transaction on Graphics*, 29(4):29:1–29:12.

Alcantarilla, P. F., Davison, A. J., and Bartoli, A. (2012). KAZE Features. In *European Conference on Computer Vision*.

Alcantarilla, P. F., Nuevo, J., and Bartoli, A. (2013). Fast Explicit Diffusion For Accelerated Features In Nonlinear Scale Spaces. In *British Machine Vision Conference*.

Apple Inc. (2014a). Customizing Existing Classes. `https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/CustomizingExistingClasses/CustomizingExistingClasses.html`. accessed 2015-09-02.

Apple Inc. (2014b). NSOperationQueue. `https://developer.apple.com/library/ios/documentation/Cocoa/Reference/NSOperationQueue_class/index.html`. accessed 2015-09-06.

Apple Inc. (2015). iOS 8.4.1. `https://support.apple.com/kb/DL1818`. accessed 2015-08-31.

Bae, S., Durand, F., and Agarwala, A. (2010). Computational Re-Photography. *ACM Transactions on Graphics*, 29(3).

Bay, H., Tuytelaars, T., and Van Gool, L. (2006). SURF: Speeded Up Robust Features. In *European Conference on Computer Vision*, pages 404–417. Springer.

Calonder, M., Lepetit, V., Strecha, C., and Fua, P. (2010). BRIEF: Binary Robust Independent Elementary Features. In *Proceedings of the 11th European Conference on Computer Vision: Part IV*, European Conference on Computer Vision, pages 778–792, Berlin, Heidelberg. Springer-Verlag.

Canny, J. (1986). A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698.

Chum, O., Werner, T., and Matas, J. (2005). Two-View Geometry Estimation Unaffected By A Dominant Plane. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 772–779. IEEE.

Decker, P., Paulus, D., and Feldmann, T. (2008). Dealing With Degeneracy In Essential Matrix Estimation. In *15th IEEE International Conference on Image Processing*, pages 1964–1967. IEEE.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Grewenig, S., Weickert, J., and Bruhn, A. (2010). From Box Filtering to Fast Explicit Diffusion. In *Proceedings of the 32Nd DAGM Conference on Pattern Recognition*, pages 533–542, Berlin, Heidelberg. Springer-Verlag.

Hartley, R. I. (1997). In Defense of the Eight-Point Algorithm. *IEEE Transactions on Pattern Analysis Machine Intelligence*, 19(6):580–593.

Hartley, R. I. and Zisserman, A. (2004). *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2nd edition.

Klett, M., Wolfe, B., Bajakian, K., Ueshina, T., Fox, W. L., and Marshall, M. (2004). Third View. `http://www.thirdview.org/3v/home/index.html`. accessed 2015-09-11.

Lepetit, V., Moreno-Noguer, F., and Fua, P. (2009). EPnP: An Accurate $\mathcal{O}(n)$ Solution to the PnP Problem. *International Journal of Computer Vision*, 81(2):155–166.

Levoy, M. (2014). Camera 2.0: New Computing Platforms For Computational Photography. `http://graphics.stanford.edu/projects/camera-2.0/`. accessible with a US IP address or via `https://webcache.googleusercontent.com/search?q=cache:http://graphics.stanford.edu/projects/camera-2.0/`; accessed 2015-09-12.

Li, H. and Hartley, R. (2006). Five-Point Motion Estimation Made Easy. In *Proceedings of the 18th International Conference on Pattern Recognition - Volume 01*, ICPR '06, pages 630–633, Washington, DC, USA. IEEE Computer Society.

Longuet-Higgins, H. C. (1987). Readings in Computer Vision: Issues, Problems, Principles, and Paradigms. chapter A Computer Algorithm for Reconstructing a Scene from Two Projections, pages 61–62. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Lowe, D. G. (1999). Object Recognition from Local Scale-Invariant Features. In *Proceedings of the International Conference on Computer Vision*, volume 2 of *ICCV '99*, pages 1150–, Washington, DC, USA. IEEE Computer Society.

Lowe, D. G. (2004). Distinctive Image Features From Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110.

Lucas, B. D. and Kanade, T. (1981). An Iterative Image Registration Technique With An Application To Stereo Vision. In *IJCAI*, volume 81, pages 674–679.

Luong, Q.-T., Deriche, R., Faugeras, O., and Papadopoulo, T. (1993). On Determining The Fundamental Matrix: Analysis Of Different Methods And Experimental Results. Research Report RR-1894.

Ma, Y., Soatto, S., Kosecka, J., and Sastry, S. S. (2003). *An Invitation to 3-D Vision: From Images to Geometric Models*. Springer Verlag.

Nistér, D. (2004). An Efficient Solution to the Five-Point Relative Pose Problem. *IEEE Transactions on Pattern Analysis Machine Intelligence*, 26(6):756–777.

Nutting, J., Bennett, E., Ruffenach, C., Schepman, C., Yang, E., and Keeley, G. (2013). ELCImagePickerController. `https://github.com/B-Sides/ELCImagePickerController`.

Ortiz, R. (2012). FREAK: Fast Retina Keypoint. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '12, pages 510–517, Washington, DC, USA. IEEE Computer Society.

Pizarro, O., Eustice, R., and Singh, H. (2003). Relative Pose Estimation For Instrumented, Calibrated Imaging Platforms. In *Proceedings of Digital Image Computing: Techniques and Applications*, pages 601–612, Sydney, Australia.

Primate Labs Inc. (2014). iPad5,4. `http://browser.primatelabs.com/geekbench3/1061742`. accessed 2015-09-05.

Rosten, E. and Drummond, T. (2005). Fusing Points and Lines for High Performance Tracking. In *Proceedings of the Tenth IEEE International Conference on Computer Vision*, ICCV '05, pages 1508–1515, Washington, DC, USA. IEEE Computer Society.

Rublee, E., Rabaud, V., Konolige, K., and Bradski, G. (2011). ORB: An Efficient Alternative to SIFT or SURF. In *Proceedings of the 2011 International Conference on Computer Vision*, ICCV '11, pages 2564–2571, Washington, DC, USA. IEEE Computer Society.

Stewénius, H., Engels, C., and Nistér, D. (2006). Recent Developments on Direct Relative Orientation. *ISPRS Journal of Photogrammetry and Remote Sensing*, 60:284–294.

Tomasi, C. and Kanade, T. (1991). Detection and Tracking of Point Features. Technical report, International Journal of Computer Vision.

Torr, P. H., Fitzgibbon, A. W., and Zisserman, A. (1999). The Problem Of Degeneracy In Structure And Motion Recovery From Uncalibrated Image Sequences. *International Journal of Computer Vision*, 32(1):27–44.

Weber, S. (2015). Aufbau einer interaktiven Internetplattform als Django-Projekt zur Erstellung und Ver/"offentlichung von Vorher-Nachher-Bildpaaren. Bachelor thesis, Osnabrück Universtiy.

Weickert, J. (1998). *Anisotropic Diffusion In Image Processing*, volume 1. Teubner Stuttgart.

Yang, X. and Cheng, K.-T. (2012). LDB: An Ultra-fast Feature For Scalable Augmented Reality On Mobile Devices. In *2012 IEEE International Symposium on Mixed and Augmented Reality*, pages 49–57.

Zhang, Z. (1998). Determining the Epipolar Geometry and Its Uncertainty: A Review. *International Journal of Computer Vision*, 27(2):161–195.

Zhang, Z. (2000). A Flexible New Technique for Camera Calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334.

DECLARATION OF AUTHORSHIP

I hereby certify that the work presented here is—to the best of my knowledge and belief—original and the result of my own investigations, except as acknowledged, and has not been submitted, either in part or whole, for a degree at this or any other university.

Rasmus Diederichsen Osnabrück, September 21, 2015