Efficient Video Rectification and Stabilisation for Cell-Phones

Erik Ringaby and Per-Erik Forssén

Noname manuscript No.
(will be inserted by the editor)

Efficient, High-Quality Video Stabilisation for Cell-Phones

Erik Ringaby · Per-Erik Forssén

Received: date / Accepted: date

Abstract This article presents a method for rectifying and stabilising video from cell-phones with *rolling shutter* (RS) cameras. Due to size constraints, cell-phone cameras have constant, or near constant focal length, making them an ideal application for calibrated projective geometry. In contrast to previous RS rectification attempts that model distortions in the image plane, we model the 3D rotation of the camera. We parameterise the camera rotation as a continuous curve, with knots distributed across a short frame interval. Curve parameters are found using non-linear least squares over inter-frame correspondences from a KLT tracker. By smoothing a sequence of reference rotations from the estimated curve, we can at a small extra cost, obtain a high-quality image stabilisation. Using synthetic RS sequences with associated ground-truth, we demonstrate that our rectification improves over two other methods. We also compare our video stabilisation with the methods in iMovie and Deshaker.

Keywords Cell-phone, Rolling shutter · CMOS · Video stabilisation

Fig. 1 Example of rolling shutter imagery. Top left: Frame from an iPhone 3GS camera sequence acquired during fast motion. Top right: Rectification using our rotation method. Bottom left: Rectification using the global affine method. Bottom right: Rectification using the global shift method. Corresponding videos are available on the web (Ringaby, 2010).

1 Introduction

Almost all new cell-phones have cameras with CMOS image sensors. CMOS sensors have several advantages over the conventional CCD sensors: notably they are cheaper to manufacture, and typically offer on-chip processing (Gamal and Eltoukhy, 2005), for e.g. automated white balance and auto-focus measurements.

E. Ringaby and P.-E. Forssén
Department of Electrical Engineering, Linköping University
SE-581 83 Linköping, Sweden
Tel.: +46(0)13 281302 (Ringaby)
E-mail: ringaby@isy.liu.se

However, most CMOS sensors, by design make use of what is known as a *rolling shutter* (RS). In an RS camera, detector rows are read and reset sequentially. As the detectors collect light right until the time of readout, this means that each row is exposed during a slightly different time window. The more conventional CCD sensors on the other hand use a *global shutter* (GS), where all pixels are reset simultaneously, and collect light during the same time interval. The downside with a rolling shutter is that since pixels are acquired at different points in time, motion of either camera or target will cause geometrical distortions in the acquired images. Figure 1 shows an example of geometric distor-

tions caused by using a rolling shutter, and the result of our rectification step, as well as two other methods.

1.1 Related Work

A camera motion between two points in time can be described with a three element translation vector, and a 3 *degrees-of-freedom* (DOF) rotation. For hand-held footage, the rotation component is typically the dominant cause of image plane motion. Whyte et al. (2010) gave a calculation example of this for the related problem of motion blur during long exposures. (A notable exception where translation is the dominant component is footage from a moving platform, such as a car.) Many new camcorders thus have *mechanical image stabilisation* (MIS) systems that move the lenses (some instead move the sensor) to compensate for small pan and tilt rotational motions (image plane rotations, and large motions, are not handled). The MIS parameters are typically optimised to the frequency range caused by a person holding a camera, and thus work well for such situations. However, since lenses have a certain mass, and thus inertia, MIS has problems keeping up with faster motions, such as caused by vibrations from a car engine. Furthermore, cell-phones, and lower end camcorders lack MIS and recorded videos from these will exhibit RS artifacts.

For cases when MIS is absent, or non-effective, one can instead do post-capture image rectification. There exist a number of different approaches for dealing with special cases of this problem (Chang et al., 2005; Liang et al., 2008; Nicklin et al., 2007; Cho and Kong, 2007; Cho et al., 2007; Chun et al., 2008). Some algorithms assume that the image deformation is caused by a globally constant translational motion across the image (Chang et al., 2005; Nicklin et al., 2007; Chun et al., 2008). After rectification this would correspond to a constant optical flow across the entire image, which is rare in practise. Liang et al. (2008) improve on this by giving each row a different motion, that is found by interpolating between constant global inter-frame motions using a Bézier curve. Another improvement is due to Cho and Kong (2007) and Cho et al. (2007). Here geometric distortion is modelled as a global affine deformation that is parametrised by the scan-line index. Recently Baker et al. (2010) improved on Cho and Kong (2007) and Liang et al. (2008) by using not one model per frame, but instead blended linearly between up to 30 affine or translational models across the image rows. This means that their model can cope with motions that change direction several times across a frame. This was made possible by using a high-quality dense optical flow field. They also used a L1 optimisation based on linear programming. However, they still model distortions in the image plane.

All current RS rectification approaches perform warping of individual frames to rectify RS imagery. Note that a perfect compensation under camera translation would require the use of multiple images, as the parallax induced by a moving camera will cause occlusions. Single frame approximations do however have the advantage that ghosting artifacts caused by multi-frame reconstruction is avoided, and is thus often preferred in video stabilisation (Liu et al., 2009).

Other related work on RS images include structure and motion estimation. Geyer et al. (2005) study the projective geometry of RS cameras, and also describe a calibration technique for estimation of the readout parameters. The derived equations are then used for structure and motion estimation in synthetic RS imagery. Ait-Aider et al. (2007) demonstrate that motion estimation is possible from single rolling shutter frames if world point-to-point distances are known, or from curves that are known to be straight lines in the world. They also demonstrate that structure and motion estimation can be done from a single stereo pair if one of the used cameras has a rolling shutter (Ait-Aider and Berry, 2009).

Video stabilisation has a long history in the literature, an early example is Green et al. (1983). The most simple approaches apply a global correctional image plane translation (Green et al., 1983; Cho and Kong, 2007). A slightly more sophisticated approach is a global affine model. A special case of this is the zoom and translation model used by Cho et al. (2007).

A more sophisticated approach is to use rotational models. Such approaches only compensate for the 3D rotational motion component, and neglect the translations, in a similar manner to mechanical stabilisation rigs (Yao et al., 1995). Rotational models estimate a compensatory rotational homography, either using instantaneous angular velocity (Yao et al., 1995) (differential form), or using inter-frame rotations, e.g. represented as unit quaternions (Morimoto and Chellappa, 1997). Since only rotations are corrected for, there are no parallax-induced occlusions to consider, and thus single-frame reconstructions are possible.

The most advanced (and computationally demanding) video stabilisation algorithms make use of *structure-from-motion* (SfM). An early example is the quasi-affine SfM explored by Buehler et al. (2001). These methods attempt to also correct for parallax changes in the stabilised views. This works well on static scenes, but introduces ghosting when the scene is dynamic, as blending from multiple views is required (Liu et al., 2009).

A variant of SfM based stabilisation is the content preserving warps introduced by Liu et al. (2009). Here single frames are used in the reconstruction, and geometric correctness is traded for perceptual plausibility.

Recently Liu et al. (2011a) presented a new stabilisation approach based on subspace constraints on 2D feature trajectories. This has the advantage that it does not rely on SfM, which is computationally heavy and sensitive to rolling shutter artifacts. The algorithm does not explicitly model a rolling shutter, instead it is treated as noise. The new algoritm can deal with rolling shutter wobble from camera shake, but not shear introduced by a panning motion.

1.2 Contributions

All the previous approaches to rectification of RS video (Chang et al., 2005; Liang et al., 2008; Nicklin et al., 2007; Cho and Kong, 2007; Cho et al., 2007; Chun et al., 2008; Baker et al., 2010) model distortions as taking place in the image plane. We instead model the 3D camera motion using calibrated projective geometry.

In this article, we focus on a distortion model based on 3D camera rotation, since we have previously shown that it outperforms a combined rotation and translation model (Forssén and Ringaby, 2010). In this article, we extend the rotational model to use multiple knots across a frame. This enables the algorithm to detect non-constant motions during a frame capture. We also introduce a scheme for positioning of the knots that makes the optimisation stable.

We demonstrate how to apply smoothing to the obtained rotation sequence to obtain a high quality video stabilisation, at a low computational cost. This results in a stabilisation similar to rotational models previously used on global shutter cameras. However, as we perform the filtering offline, the amount of smoothing can be decided by the user, post capture. This can e.g. be done using a slider in a video editing application running on a cell-phone. We compare our video stabilisation with the methods in iMovie and Deshaker.

We also introduce a rectification technique using forward interpolation. Many new smartphones have hardware for graphics acceleration, e.g. using the *OpenGL ES 2.0* application programming interface. Such hardware can be exploited using our forward interpolation technique, to allow rectification and stabilisation during video playback.

We recently introduced the first (and currently only) dataset for evaluation of algorithms that rectify rolling shutter video (Ringaby, 2010). We now extend it with another 2 sequences, and add a more sophisticated comparison between rectifications and ground truth. Using the dataset, we compare our own implementations of the global affine model (Liang et al., 2008), and the global shift model (Chun et al., 2008) to the new method that we propose. Our dataset, evaluation code and supplementary videos are available for download at (Ringaby, 2010).

1.3 Overview

This article is organised as follows: Section 2, describes how to calibrate a rolling-shutter camera. Section 3 introduces the model and cost function for camera egomotion estimation. Section 4 discusses interpolation schemes for rectification of rolling-shutter imagery. Section 5 describes how to use the estimated camera trajectory for video stabilisation. Section 6 describes the algorithm complexity and cell-phone implementation feasibility. Section 7 describes our evaluation dataset. In section 8 we use our dataset to compare different rectification strategies, and to compare our 3D rotation model to our own implementations of Liang et al. (2008) and Chun et al. (2008). We also compare our stabilisation to iMovie and Deshaker. In section 9 we describe the supplemental material and discuss the performance of the algorithms. The article concludes with outlooks and concluding remarks in section 10.

2 Rolling Shutter Camera Calibration

In this article, we take the *intrinsic camera matrix*, the *camera frame-rate* and the *inter-frame delay* to be given. This reduces the number of parameters that need to be estimated on-line, but also requires us to calibrate each camera before the algorithms can be used.

On camera equipped cell-phones, such calibration makes good sense, as the parameters stay fixed (or almost fixed, in the case of variable focus cameras) throughout the lifetime of a unit. We have even found that transferring calibrations between cell-phones of the same model works well.

2.1 Geometric Calibration

A 3D point, $\mathbf{X}$, and its projection in the image, $\mathbf{x}$, given in homogeneous coordinates, are related according to

$$\mathbf{x} = \mathbf{K}\mathbf{X} \;, \text{ and } \quad \mathbf{X} = \lambda \mathbf{K}^{-1}\mathbf{x} \;, \tag{1}$$

where $\mathbf{K}$ is a 5DOF upper triangular 3×3 *intrinsic camera matrix*, and λ is an unknown scaling (Hartley and Zisserman, 2000).

We have estimated $\mathbf{K}$ for a number of cell-phones using the calibration plane method Zhang (2000) as implemented in OpenCV.

Note that many high-end cell-phones have variable focus. As this is implemented by moving the single lens of the camera back and forth, the camera focal length will vary slightly. On e.g. the iPhone 3GS the field-of-view varies with about 2°. However, we have found that such small changes in the $\mathbf{K}$ matrix makes no significant difference in the result.

2.2 Readout Time Calibration

The RS chip frame period $1/f$ (where f is the *frame rate*) is divided into a *readout time* t_r, and an *inter-frame delay*, t_d as

$$1/f = t_r + t_d \,. \tag{2}$$

The readout time can be calibrated by imaging a flashing light source with known frequency (Geyer et al., 2005), see appendix A for details. For a given frame rate f, the inter-frame delay can then be computed using (2). For our purposes it is preferable to use rows as fundamental unit, and express the inter-frame delay as a number of *blank rows*:

$$N_b = N_r t_d/(1/f) = N_r(1 - t_r f) \,. \tag{3}$$

Here N_r is the number of image rows.

3 Camera Motion Estimation

Our model of camera motion is a rotation about the camera centre during frame capture, in a smooth, but time varying way. Even though this model is violated across an entire video clip, we have found it to work quite well if used on short frame intervals of 2-4 frames (Forssén and Ringaby, 2010). We represent the model as a sequence of rotation matrices, $\mathbf{R}(t) \in \mathrm{SO}(3)$.

Two homogeneous image points $\mathbf{x}$, and $\mathbf{y}$, that correspond in consecutive frames, are now expressed as:

$$\mathbf{x} = \mathbf{K}\mathbf{R}(N_x)\mathbf{X} \;, \text{ and } \quad \mathbf{y} = \mathbf{K}\mathbf{R}(N_y)\mathbf{X} \tag{4}$$

where N_x and N_y correspond to the time parameters for point $\mathbf{x}$ and $\mathbf{y}$ respectively. This gives us the relation:

$$\mathbf{x} = \mathbf{K}\mathbf{R}(N_x)\mathbf{R}^T(N_y)\mathbf{K}^{-1}\mathbf{y} \,. \tag{5}$$

The time parameter is a linear function of the current image row (i.e. x_2/x_3 and y_2/y_3). Thus, by choosing the unit of time as image rows, and time zero as the top row of the first frame, we get $N_x = x_2/x_3$ for points in the first image. In the second image we get $N_y = y_2/y_3 + N_r + N_b$, where N_r is the number of image rows in a frame, and N_b is defined in (3).

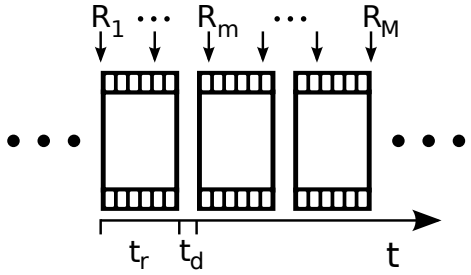

Fig. 2 Rotations, $\mathbf{R}_2$, $\mathbf{R}_3, \ldots, \mathbf{R}_M$ found by non-linear optimisation. Intermediate rotations are defined as interpolations of these, and $\mathbf{R}_1 = \mathbf{I}$. Readout time t_r, and inter-frame delay t_d are also shown.

Each correspondence between the two views, (5) gives us two equations (after elimination of the unknown scale) where the unknowns are the rotations. Unless we constrain the rotations further, we now have six unknowns (a rotation can be parametrised with three parameters) for each correspondence. We thus parametrise the rotations with an interpolating linear spline with a number of knots placed over the current frame window, see figure 2 for an example with three frames and $M = 6$ knots. Intermediate rotations are found using spherical linear interpolation (Shoemake, 1985).

As we need a reference world frame, we might as well fixate that to the start of frame 1, i.e. set $\mathbf{R}_1 = \mathbf{I}$. This gives us $3(M-1)$ unknowns in total for a group of M knots.

3.1 Motion Interpolation

Due to the periodic structure of SO(3) the interpolation is more complicated than regular linear interpolation.

We have chosen to represent rotations as three element vectors, $\mathbf{n}$, where the magnitude, ϕ, corresponds to the rotation angle, and the direction, $\hat{\mathbf{n}}$, is the axis of rotation, i.e. $\mathbf{n} = \phi\hat{\mathbf{n}}$. This is a minimal parametrisation of rotations, and it also ensures smooth variations, in contrast to e.g. Euler angles. It is thus suitable for parameter optimisation. The vector $\mathbf{n}$ can be converted to a rotation matrix using the matrix exponent, which for a rotation simplifies to Rodrigues formula:

$$\mathbf{R} = \mathrm{expm}(\mathbf{n}) = \mathbf{I} + [\hat{\mathbf{n}}]_x \sin\phi + [\hat{\mathbf{n}}]_x^2(1 - \cos\phi) \tag{6}$$

$$\text{where } [\hat{\mathbf{n}}]_x = \frac{1}{\phi}\begin{pmatrix} 0 & -n_3 & n_2 \\ n_3 & 0 & -n_1 \\ -n_2 & n_1 & 0 \end{pmatrix} \,. \tag{7}$$

Conversion back to vector form is accomplished through the matrix logarithm in the general case, but for a rotation matrix, there is a closed form solution. We note

that two of the terms in (6) are symmetric, and thus terms of the form $r_{ij} - r_{ji}$ will come from the anti-symmetric part alone. Conversely the trace is only affected by the symmetric parts. This allows us to extract the axis and angle as:

$$\mathbf{n} = \text{logm}(\mathbf{R}) = \phi\hat{\mathbf{n}}\,, \text{ where } \begin{cases} \tilde{\mathbf{n}} = \begin{pmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{pmatrix} \\ \phi = \tan^{-1}(||\tilde{\mathbf{n}}||, \text{tr}\mathbf{R} - 1) \\ \hat{\mathbf{n}} = \tilde{\mathbf{n}}/||\tilde{\mathbf{n}}||\,. \end{cases} \tag{8}$$

It is also possible to extract the rotation angle from the trace of $\mathbf{R}$ alone (Park and Ravani, 1997). We recommend (8), as it avoids numerical problems for small angles. Using (6) and (8), we can perform SLERP (Spherical Linear intERPolation) (Shoemake, 1985) between two rotations $\mathbf{n}_1$ and $\mathbf{n}_2$, using an interpolation parameter $\tau \in [0, 1]$ as follows:

$$\mathbf{n}_{\text{diff}} = \text{logm}\left(\text{expm}(-\mathbf{n}_1)\text{expm}(\mathbf{n}_2)\right) \tag{9}$$

$$\mathbf{R}_{\text{interp}} = \text{expm}(\mathbf{n}_1)\text{expm}(\tau\mathbf{n}_{\text{diff}}) \tag{10}$$

3.2 Optimisation

We now wish to solve for the unknown motion parameters, using iterative minimisation. For this we need a cost function:

$$J = \epsilon(\mathbf{n}_1, \ldots, \mathbf{n}_N). \tag{11}$$

To this end, we choose to minimise the (symmetric) image-plane residuals of the set of corresponding points $\mathbf{x}_k \leftrightarrow \mathbf{y}_k$:

$$J = \sum_{k=1}^{K} d(\mathbf{x}_k, \mathbf{H}\mathbf{y}_k)^2 + d(\mathbf{y}_k, \mathbf{H}^{-1}\mathbf{x}_k)^2 \tag{12}$$

$$\text{where } \mathbf{H} = \mathbf{K}\mathbf{R}(\mathbf{x}_k)\mathbf{R}^T(\mathbf{y}_k)\mathbf{K}^{-1} \tag{13}$$

Here the distance function $d(\mathbf{x}, \mathbf{y})$ for homogeneous vectors, is given by:

$$d(\mathbf{x}, \mathbf{y})^2 = (x_1/x_3 - y_1/y_3)^2 + (x_2/x_3 - y_2/y_3)^2\,. \tag{14}$$

For each frame interval we have M knots for the linear interpolating spline. The first knot is at the top row of the fist frame, and the last knot at the bottom row of the last frame in the interval. The position of a knot, the *knot time* (N_m), is expressed in the unit rows, where rows are counted from the start of the first frame in the interval. E.g. a knot at the top row of the second frame would have the knot time $N_m = N_r + N_b$ and correspond to the rotation $\mathbf{n}_m$.

We denote the evaluation of the spline at a row value N_{curr} by:

$$\mathbf{R} = \texttt{SPLINE}(\{\mathbf{n}_m, N_m\}_1^M\,, N_{\text{curr}})\,. \tag{15}$$

The value is obtained using SLERP as:

$$\mathbf{R} = \texttt{SLERP}(\mathbf{n}_m, \mathbf{n}_{m+1}, \tau)\,, \text{ for} \tag{16}$$

$$\tau = \frac{N_{\text{curr}} - N_m}{N_{m+1} - N_m}\,, \quad \text{where} \quad N_m \le N_{\text{curr}} \le N_{m+1}. \tag{17}$$

Here SLERP is defined in (9,10), N_{curr} is the current row relative to the top row in first frame and N_m, N_{m+1} are the two neighbouring knot times.

For speed, and to ensure that the translation component of the camera motion is negligible, we have chosen to optimise over short intervals of $N = 2$, 3 or 4 frames.

We have chosen to place the knots on different height for the different frames within the frame interval, see figure 3 for two examples. If the knots in two consecutive frames have the same height, the optimisation might drift sideways without increasing the residuals.

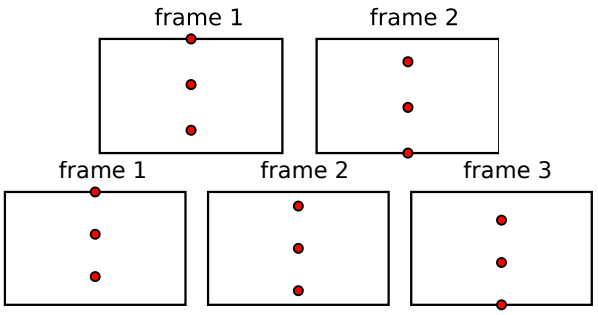

Fig. 3 Top: Frame interval of 2 frames with 6 knots. Bottom: Frame interval of 3 frames with 9 knots.

There is a simple way to initialise a new interval from the previous one. Once the optimiser has found a solution for a group of frames, we change the origin to the second camera in the sequence (see figure 2). This shift of origin is described by the rotation:

$$\mathbf{R}_{\text{shift}} = \texttt{SPLINE}(\{\mathbf{n}_m, N_m\}_1^M\,, N_r + N_b)\,, \tag{18}$$

Then we shift the interval one step, by re-sampling the spline knots $\{\mathbf{n}_m\}_1^M$, with an offset of $N_r + N_b$

$$\mathbf{R}_m = \texttt{SPLINE}(\{\mathbf{n}_k, N_k\}_1^M\,, N_m + N_r + N_b)\,, \tag{19}$$

and finally correct them for the change of origin, using

$$\mathbf{n}'_m = \text{logm}(\mathbf{R}_{\text{shift}}^T\mathbf{R}_m)\,, \quad \text{for} \quad m = 1, \ldots, L\,, \tag{20}$$

where N_L is the last time inside the frame interval. As initialisation of the rotations in the newly shifted-in frame, we copy the parameters for the last valid knot, $\mathbf{n}'_L$.

3.3 Point Correspondences

The point correspondences needed to estimate the rotations are obtained through point tracking. We use Harris points (Harris and Stephens, 1988) detected in the current frame and the chosen points are tracked using the KLT tracker (Lucas and Kanade, 1981; Shi and Tomasi, 1994). The KLT tracker uses a spatial intensity gradient search which minimises the Euclidean distance between the corresponding patches in the consecutive frames. We use the scale pyramid implementation of the algorithm in OpenCV. Using pyramids makes it easier to detect large movements.

To increase the accuracy of the point tracker, a crosschecking procedure is used (Baker et al., 2007). When the points have been tracked from the first image to the other, the tracking is reversed and only the points that return to the original position (within a threshold) are kept. The computation cost is doubled but outliers are removed effectively.

3.3.1 Correspondence Accuracy Issues

If the light conditions are poor, as is common indoors, and the camera is moving quickly, the video will contain motion blur. This is a problem for the KLT tracker, which will have difficulties finding correspondences and many tracks will be removed by the crosschecking step.

If the light conditions are good enough for the camera to have a short exposure, and a frame has severe rolling shutter artifacts, the KLT tracker may also have problems finding any correspondences due to local shape distortions. We have however not seen this in any videos, not even during extreme motions. The tracking can on the other hand result in a small misalignment or a lower number of correspondences, when two consecutive frames have very different motions (and thus different local appearances).

It may be possible to reduce the influence of rolling-shutter distortion on the tracking accuracy, by detecting and tracking points again, in the rectified images. This is however difficult because the inverse mapping back to the original image is not always one-to-one and it may thus not be possible to express these new correspondences in the original image grid. How to improve the measurements in this way is an interesting future research possibility.

4 Image Rectification

Once we have found our sequence of rotation matrices, we can use them to rectify the images in the sequence. Each image row has experienced a rotation according to (15). This rotation is expressed relative to the start of the frame, and applying it to the image points would thus rectify all rows to the first row. We can instead align them all to a reference row $\mathbf{R}_{\text{ref}}$ (we typically use the middle row), using a compensatory rotation:

$$\mathbf{R}'(\mathbf{x}) = \mathbf{R}_{\text{ref}}\mathbf{R}^T(\mathbf{x})\,. \tag{21}$$

This gives us a forward mapping for the image pixels:

$$\mathbf{x}' = \mathbf{K}\mathbf{R}_{\text{ref}}\mathbf{R}^T(\mathbf{x})\mathbf{K}^{-1}\mathbf{x} \tag{22}$$

This tells us how each point should be displaced in order to rectify the scene. Using this relation we can transform all the pixels to their new, rectified locations.

We have chosen to perform the rectifying interpolation by utilising the parallel structure of the GPU. A grid of vertices can be bound to the distorted rolling shutter image and rectified with (22) using the *OpenGL Shading Language* (GLSL) vertex shader. On a modern GPU, a dense grid with one vertex per pixel is not a problem, but on those with less computing power, e.g. mobile phones with OpenGL ES 2.0, the grid can be heavily down-sampled without loss of quality. We have used a down-sampling factor of 8 along the columns, and 2 along the rows. This gave a similar computational cost as inverse interpolation on Nvidia G80, but without noticeable loss of accuracy compared to dense forward interpolation.

By defining a grid larger than the input image we can also use the GPU texture out-of-bound capability to get a simple extrapolation of the result.

It is also tempting to use regular, or *inverse interpolation*, i.e. invert (22) to obtain:

$$\mathbf{x} = \mathbf{K}\mathbf{R}(\mathbf{x})\mathbf{R}_{\text{ref}}^T\mathbf{K}^{-1}\mathbf{x}'\,. \tag{23}$$

By looping over all values of $\mathbf{x}'$, and using (23) to find the pixel locations in the distorted image, we can cubically interpolate these. If we have large motions this approach will however have problems since different pixels within a row should be transformed with different homographies, see figure 4. Every pixel within a row in the distorted image does however share the same homography as we described in the forward interpolation case. For a comparison between the results of forward and inverse interpolation see figure 5.

5 Video Stabilisation

Our rectification algorithm also allows for video stabilisation, by smoothing the estimated camera trajectory. The rotation for the reference row $\mathbf{R}_{\text{ref}}$ in section 4 is typically set to the current frame's middle row. If we instead do a temporal smoothing of these rotations for a sequence of frames we get a more stabilised video.

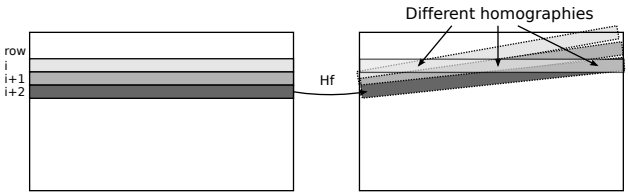

Fig. 4 Left: RS distorted image. Right: Output image.

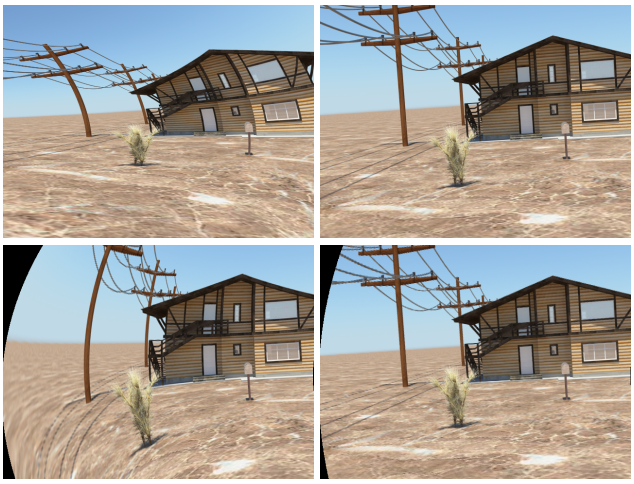

Fig. 5 Example of forward and inverse interpolation with large motion. Top left: Rolling shutter frame. Top right: Ground truth. Bottom left: Rectification using inverse interpolation. Bottom right: Rectification using forward interpolation.

5.1 Rotation Smoothing

Averaging on the rotation manifold is defined as:

$$\mathbf{R}^* = \underset{\mathbf{R}\in \mathrm{SO}(3)}{\arg\min} \sum_k d_{\mathrm{geo}}(\mathbf{R}, \mathbf{R}_k)^2 \tag{24}$$

where

$$d_{\mathrm{geo}}(\mathbf{R}, \mathbf{R}_k)^2 = ||\mathrm{logm}(\mathbf{R}_1^T \mathbf{R}_2)||_{\mathrm{fro}}^2 \tag{25}$$

is the *geodesic distance* on the rotation manifold (i.e. the relative rotation angle), $||\cdot||_{\mathrm{fro}}$ is the Fröbenius matrix norm, and logm is defined in (8). Finding $\mathbf{R}$ using (24) is iterative, and thus slow, but Gramkow (2001) showed that the barycentre of either the quaternions or rotation matrices are good approximations.

We have tried both methods, but only discuss the rotation matrix method here. The rotation matrix average is given by:

$$\tilde{\mathbf{R}}_k = \sum_{l=-n}^{n} w_l \mathbf{R}_{k+l} \tag{26}$$

where the temporal window is $2n+1$ and w are weights for the input rotations $\mathbf{R}_k$. We have chosen w_l as a Gaussian filter kernel. To avoid excessive correction at the start and end of the rotation sequence, we extend the sequence by replicating the first (and last) rotation a sufficient number of times.

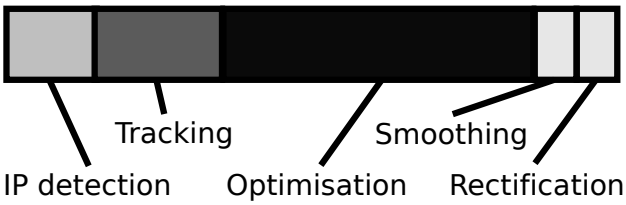

Fig. 6 Overview of algorithm complexity. The width of each box roughly corresponds to the fraction of time the algorithm spends there during one pass of optimisation, and one pass of smoothing and playback.

The output of (26) is not guaranteed to be a rotation matrix, but this can be enforced by constraining it to be orthogonal (Gramkow, 2001):

$$\hat{\mathbf{R}}_k = \mathbf{U}\mathbf{S}\mathbf{V}^T, \text{ where} \tag{27}$$
$$\mathbf{U}\mathbf{D}\mathbf{V}^T = \mathrm{svd}(\tilde{\mathbf{R}}_k), \text{ and } \mathbf{S} = \mathrm{diag}(1, 1, |\mathbf{U}||\mathbf{V}|).$$

5.2 Common Frame of Reference

Since each of our reference rotations $\mathbf{R}_{\mathrm{ref}}$ (see section 4) has its own local coordinate system, we have to transform them to a common coordinate system before we can apply rotation smoothing to them.

We do this using the $\mathbf{R}_{\mathrm{shift},k}$ matrices in (18) that shift the origin from one frame interval to the next. Using these, we can recursively compute shifts to the absolute coordinate frame as:

$$\mathbf{R}_{\mathrm{abs},k} = \mathbf{R}_{\mathrm{abs},k-1}\mathbf{R}_{\mathrm{shift},k}, \text{ and } \mathbf{R}_{\mathrm{abs},1} = \mathbf{I}. \tag{28}$$

Here $\mathbf{R}_{\mathrm{abs}}$ is the shift to the absolute coordinate system, and $\mathbf{R}_{\mathrm{shift}}$ is the local shift computed in (18).

We can now obtain reference rotations in the absolute coordinate frame:

$$\mathbf{R}'_{\mathrm{ref},k} = \mathbf{R}_{\mathrm{abs},k}\mathbf{R}_{\mathrm{ref},k}. \tag{29}$$

After smoothing these, using (26) and (27), we change back to the local coordinate system by multiplying them with $\mathbf{R}^T_{\mathrm{abs},k}$.

6 Algorithm Complexity

The rectification and stabilisation pipeline we have presented can be decomposed into five steps, as shown in figure 6. We have analysed the computational complexity of these steps on our reference platform, which is a HP Z400 workstation, running at 2.66GHz (using one

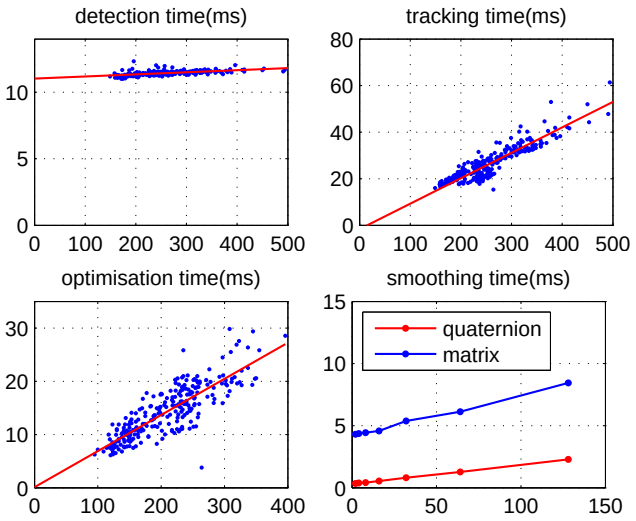

Fig. 7 Execution times for algorithm steps. Top left: Detection as function of number of points. Top right: Tracking as function of number of points. Bottom left: Optimisation as function of number of accepted trajectories. Dots are individual measurements, solid lines are least-squares line fits. Bottom right: Rotation smoothing time as function of σ in the Gaussian kernel.

core). The graphics card used in the rectification step is an Nvidia GeForce 8800 GTS.

As can be seen in figure 6, most computations are done in three preprocessing steps. These are executed once for each video clip: (1) Interest-point detection, (2) KLT tracking and crosschecking, and (3) Spline optimisation. We have logged execution times of these steps during processing of a 259 frame video clip, captured with an iPhone 3GS (The stabilisation example video in our CVPR paper (Forssén and Ringaby, 2010)). This video results in a highly variable number of trajectories (as it has large portions of textureless sky), and thus gives us good scatter plots. We present these timings in figure 7.

The remaining two steps are (4) Rotation smoothing, and (5) Playback of stabilised video. Although these are much less expensive, these steps must be very efficient, as they will be run during interaction with a user when implemented in a cell-phone application.

In the following subsections, we will briefly analyse the complexity of each of the five steps.

6.1 Interest-Point Detection

The time consumption for the Harris interest point detection is fairly insensitive to the number of points detected. Instead it is roughly linear in the number of pixels. If needed, it can with a small effort be implemented in OpenGL. Average absolute timing: 11.4 ms/frame.

6.2 KLT Tracking

The KLT tracker (see section 3.3) has a time complexity of $O(K \times I)$, where K is the number of tracked points, and I is the number of iterations per point. The average absolute timing for this step is: 26.9 ms/frame including crosschecking rejection. The tracking time as a function of number of interest points is plotted in figure 7.

6.3 Spline Optimisation

The spline optimisation (see section 3.2) has a time complexity of $O(N \times I \times M \times F)$, where N is the number of points remaining after crosschecking rejection, I is the number of iterations, M is the number of knots used in the frame interval, and F is the number of frames in the interval. Average absolute timing: 14.4 ms/frame, when using $M = 3$ and the `levmar` library[1]. The optimisation time as a function of number of accepted trajectories is plotted in figure 7.

6.4 Rotation Smoothing

Rotation smoothing (see section 5.1) is run each time the user changes the amount of stabilisation desired. It is linearly dependent on the length of the video clip, and the kernel size. Figure 7, bottom right, shows a plot of execution times over a 259 frame video clip (averaged over 10 runs).

6.5 Playback of Stabilised Video

The rectification and visualisation step is run once for each frame during video playback. This part is heavily dependent on the graphics hardware and its capability to handle calculations in parallel. The execution time can approximately be kept constant, even though the resolution of the video is changed, by using the same number of vertices for the vertex shader.

6.6 Cell-phone Implementation Feasibility

Our aim is to allow this algorithm to run on mobile platforms. It is designed to run in a streaming fashion by making use of the sliding frame window. This has the advantage that the number of parameters to be estimated is kept low together with a low requirement of memory, which is limited on mobile platforms.

[1] http://www.ics.forth.gr/∼lourakis/levmar/

The most time-consuming part of our algorithm is usually the non-linear optimisation step when optimising for many knots. Our sliding window approach does however enable us to easily initialise the new interval from the previous one. Since cell-phone cameras have constant focal length we do not need to optimise for this. The KLT tracking step also takes a considerable amount of the total time. The time for both tracking and optimisation can however be regulated by changing the Harris detector to select fewer points.

The most time critical part is the stabilisation and the visualisation, because when the camera motion has been estimated, the user wants a fast response when changing the stabilisation settings. The stabilisation part is already fast, and since the visualisation is done on a GPU it is possible to play it back in real-time by downsampling the vertex grid to match the current hardware capability.

7 Synthetic Dataset

In order to do a controlled evaluation of algorithms for RS compensation we have generated eight test sequences, available at (Ringaby, 2010), using the Autodesk Maya software package. Each sequence consists of 12 RS distorted frames of size 640×480, corresponding ground-truth *global shutter* (GS) frames, and *visibility masks* that indicate pixels in the ground-truth frames that can be reconstructed from the corresponding rolling-shutter frame. In order to suit all algorithms, the ground-truth frames and visibility masks come in three variants: for rectification to the time instant when the first, middle and last row of the RS frame were imaged.

Each synthetic RS frame is created from a GS sequence with one frame for each RS image row. One row in each GS image is used, starting at the top row and sequentially down to the bottom row. In order to simulate an inter-frame delay, we also generate a number of GS frames that are not used to build any of the RS frames. The camera is however still moving during these frames.

We have generated four kinds of synthetic sequences, using different camera motions in a static scene, see figure 8. The four sequence types are generated as follows:

#1 In the first sequence type, the camera rotates around its centre in a spiral fashion, see figure 8 top left. Three different versions of this sequence exist to test the importance of modelling the inter-frame delay. The different inter-frame delays are $N_b = 0$, 20 and 40 *blank rows* (i.e. the number of unused GS frames).

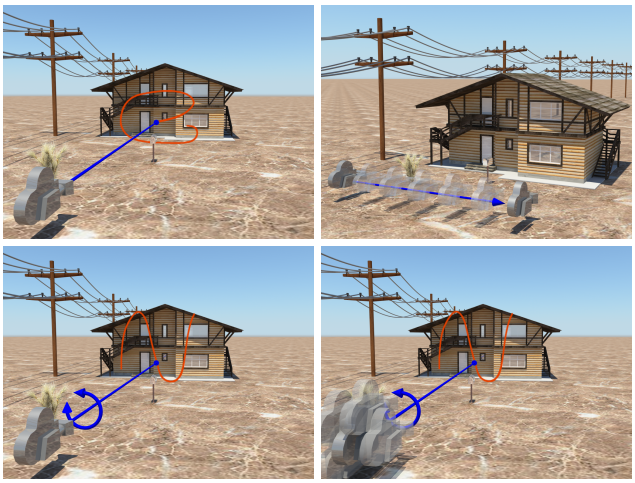

Fig. 8 The four categories of synthetic sequences. Left to right, top to bottom: #1 rotation only, #2 translation only, # full 3DOF rotation. and #4 3DOF rotation and translation.

#2 In the second sequence type, the camera makes a pure translation to the right and has an inter-frame delay of 40 blank rows, see figure 8 top right.

#3 In the third sequence type the camera makes an up/down rotating movement, with a superimposed rotation from left to right, see figure 8 bottom left. There is also a back-and-forth rotation with the viewing direction as axis.

#4 The fourth sequence type is the same as the third except that a translation parallel to the image plane has been added, see figure 8 bottom right. There are three different versions of this type, all with different amounts of translation.

For each frame in the ground-truth sequences, we have created *visibility masks* that indicate pixels that can be reconstructed from the corresponding RS frame, see figure 9. These masks were rendered by inserting one light source for each image row, into an otherwise dark scene. The light sources had a rectangular shape that illuminates exactly the part of the scene that was imaged by the RS camera when located at that particular place. To acquire the mask, a global shutter render is triggered at the desired location (e.g. corresponding to first, middle or last row in the RS-frame).

8 Experiments

8.1 Choice of Reference Row

It is desirable that as many pixels as possible can be reconstructed to a global shutter frame, and for this purpose, rectifying to the first row (as done by Chun et al. (2008) and Liang et al. (2008)), middle row, or

Fig. 9 Left to right: Rendered RS frame from sequence of type #2, with $N_b = 40$ (note that everything is slightly slanted to the right), corresponding global-shutter ground-truth, and visibility mask with white for ground-truth pixels that were seen in the RS frame.

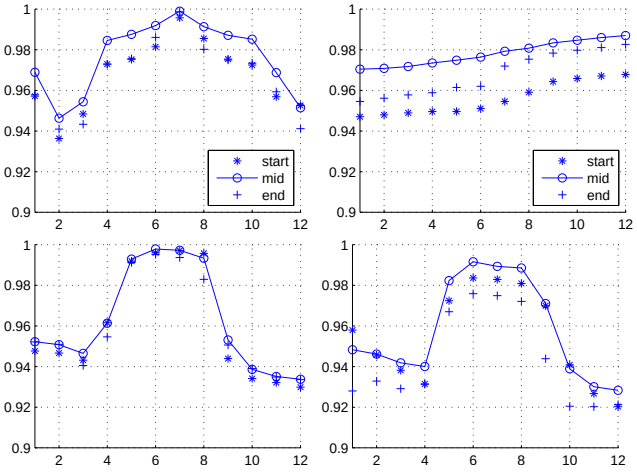

Fig. 10 Comparison of visibility masks for sequences #1, #2, #3, and #4 with $B = 40$. Plots show the fraction of non-zero pixels of the total image size.

last image row make a slight difference. The dataset we have generated allows us to compare these three choices. In figure 10, we have plotted the fraction of visible pixels for all frames in one sequence from each of our four camera motion categories. As can be seen in this plot, reconstruction to the middle row gives a higher fraction of visible pixels for almost all types of motion. This result is also intuitively reasonable, as the middle row is closer in time to the other rows, and thus more likely to have a camera orientation close to the average.

8.2 Rectification Accuracy

We have compared our methods to the *global affine model* (GA) (Liang et al., 2008), and the *global shift model* (GS) (Chun et al., 2008) on our synthetic sequences, see section 7.

8.2.1 Contrast Invariant Error Measure

When we introduced the RS evaluation dataset (Forssén and Ringaby, 2010) we made use of a thresholded Euclidean colour distance to compare ground-truth frames with the rectification output. This error measure has the disadvantage that it is more sensitive in high-contrast regions, than in regions with low contrast. It is also overly sensitive to sub-pixel rectification errors. For these reasons, we now instead use a variance-normalised, error measure:

$$\epsilon(\mathbf{I}_{\text{rec}}) = \sum_{k=1}^{3} \frac{(\mu_k - I_{\text{rec},k})^2}{\sigma_k^2 + \varepsilon\mu_k^2} \,. \tag{30}$$

Here μ_k and σ_k are the means and standard deviations of each colour band in a small neighbourhood of the ground truth image pixel (we use a 3×3 region), and ε is a small value that controls the amount of regularisation. We use $\varepsilon = 2.5e-3$, which is the smallest value that suppresses high ϵ values in homogeneous regions such as the sky.

The error measure (30) is invariant to a simultaneous scaling of the reconstructed image and the ground-truth image, and thus automatically compensates for contrast changes.

8.2.2 Statistical Interpretation of the Error

If we assume that the colour channels are Gaussian and independent, we get $\epsilon \in \mathcal{X}_3^2$, and this allows us to define a threshold in terms of probabilities. E.g. a threshold t that accepts 75% of the probability mass is found from:

$$0.75 = \int_0^t p_{\mathcal{X}_3^2}(\epsilon)d\epsilon = \int_0^t \sqrt{\frac{\epsilon}{2\pi}} e^{-\epsilon/2} d\epsilon \,. \tag{31}$$

This results in $t = 4.11$, which is the threshold that we use.

8.2.3 Sub-Pixel Shifts

An additional benefit of using (30) is that the sensitivity to errors due to sub-pixel shifts of the pixels is greatly reduced. We used bicubic resampling to test sub-pixel shifts in the range $\Delta x, \Delta y \in [0.5, 0.5]$ (step size 0.1) on sequence #2. For (30), we never saw more than 0.05% pixels rejected in a shifted image. The corresponding figure for the direct Euclidean distance is 2.5% (using a threshold of 0.3 as in (Forssén and Ringaby, 2010)).

8.2.4 Accuracy Measure

As accuracy measure we use the fraction of pixels where the error in (30) is below t. The fraction is only computed within the visibility mask.

For clarity of presentation, we only present a subset of the results on our synthetic dataset. As a baseline, all plots contain the errors for uncorrected frames represented as continuous vertical lines for the means and dashed lines for the standard deviations.

8.2.5 Results

As our reconstruction solves for several cameras in each frame interval, we get multiple solutions for each frame (except in outermost frames). If the model accords with the real motion, the different solutions for the current frame are similar, and we have chosen to present all results for the first reconstruction.

The size of the temporal window used in the optimisation has been studied. The longer the window, the less the rotation-only assumption will hold, and a smaller number of points can be tracked through the frames. In figure 11 the result for two of the sequences can be seen, where the number of frames varied between 2 and 4. In each row, the mean result of all available frames is represented by a diamond centre. Left and right diamond edges indicate the standard deviation.

The result for different numbers of knots can also be seen in figure 11. For fewer knots the optimisation is faster, but the constant motion between sparsely spaced knots is less likely to hold. For many knots, the optimisation time will not only increase (and become more unstable), the probability to have enough good points within the corresponding image region also declines.

On pure rotation scenes it is also interesting to compare the output rotations with the corresponding ground truth. In figure 12 an example from sequence type #3 with four knots spaced over four frames is shown, together with the ground truth rotations. The rotations are represented with the three parameters described in section 3.1. A comparison between different numbers of knots and the ground thruth is shown in figure 13, using the geodesic distance defined in (25).

From figure 11 we see that 6 knots over 2 frames, 9 knots over 3 frames and 12 knots over 4 frames give good results. We have also seen good result on real data with these choices and have chosen to do the remaining experiments with the 2 and 3 frame configurations.

For the pure rotation sequences (type #1 and #3) we get almost perfect results, see figures 14 and 15. The GS and GA methods however, do no improvement to the unrectified frames. Figure 16 shows the results from three sequences of type #4 with different amounts of translation. Our methods work best with little translation, but they are still better than the unrectified baseline, as well as the GS and GA methods. In the bottom figure the amount of translation is big and would not be possible to achieve while holding a device in the hand.

An even more extreme case is sequence type # 2, where the camera motion is pure translational. Results for this can be seen in figure 17. This kind of motion only gives rolling shutter effects if the camera translation is fast, e.g. video captured from a car. To better

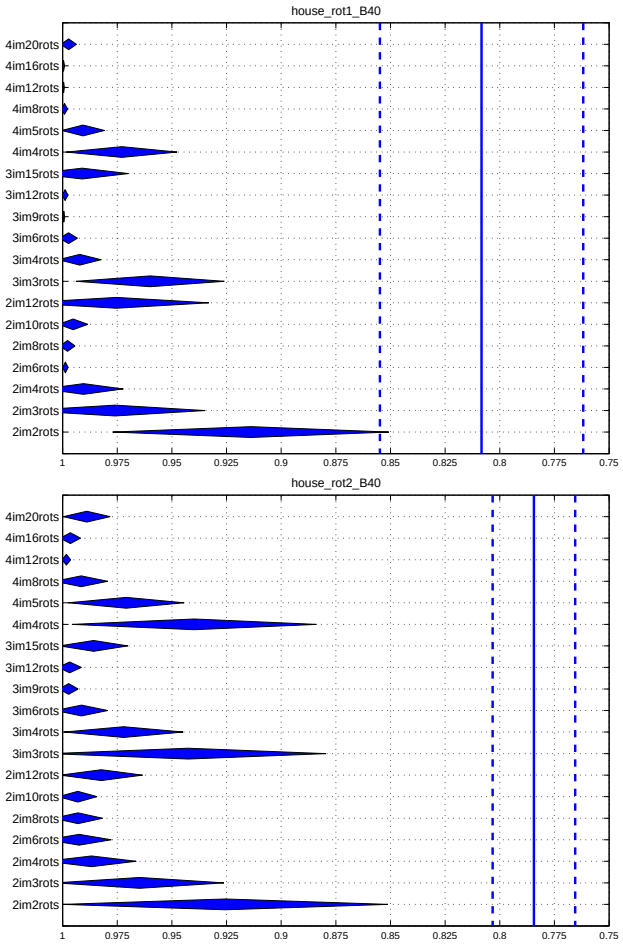

Fig. 11 Top: sequence type #1. Bottom: sequence type #3. Two examples of rectification results with different number of knots and window sizes. Diamond centres are mean accuracy, left and right indicate the standard deviation. The continuous line is the unrectified mean and the dashed lines show the standard deviation.

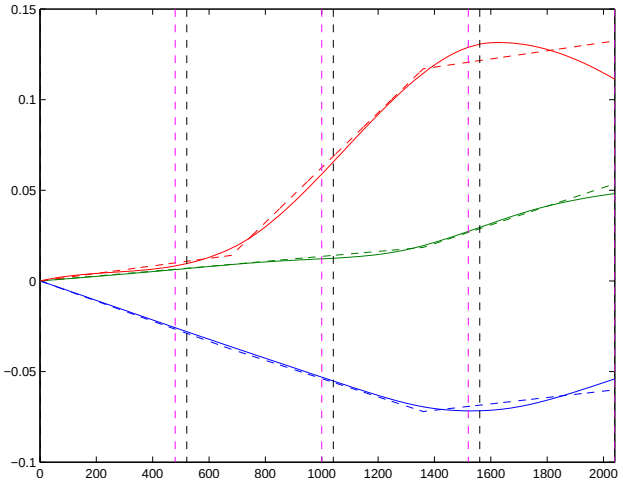

Fig. 12 Sequence type #3 with four knots spaced over four frames. Continuous line: ground truth rotations. Dashed lines: result rotations. Vertical black lines define beginning of frames and vertical magenta defines end of frames.

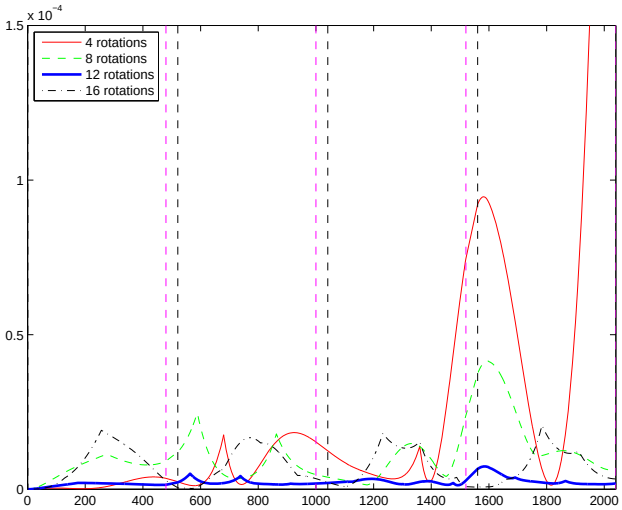

Fig. 13 Sequence type #3. Comparison between different numbers of knots and the ground truth rotations. Vertical black lines define beginning of frames and vertical magenta defines end of frames.

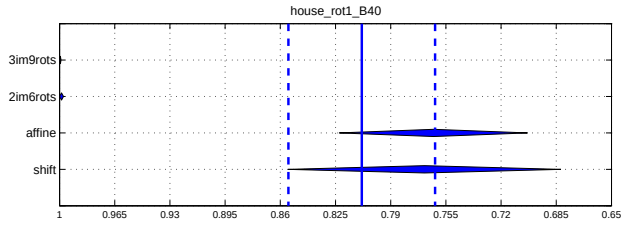

Fig. 14 Rectification results for sequence type #1, $N_b = 40$.

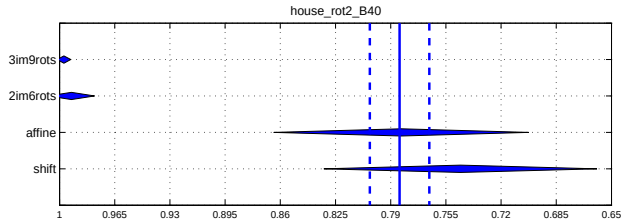

Fig. 15 Rectification results for sequence type #3.

cope with this case, a switching of models can be integrated. The GA method performs better than the GS method on average. This is because the GS method finds the dominant plane in the scene (the house) and rectifies the frame based on this. The ground does not follow this model and thus the rectification there differs substantially from the ground-truth.

Worth noting is that if the rolling shutter effects arise from a translation, objects on different depth from the camera will get different amounts of geometrical distortions. For a complete solution one needs to compensate for this locally in the image. Due to occlusions, several frames may also have to be used in the reconstruction.

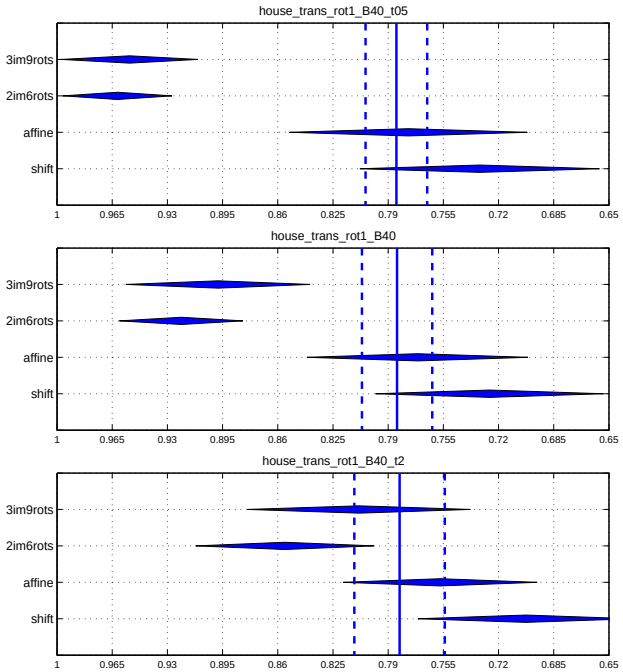

Fig. 16 Rectification results for sequences of type #4. Different amounts of translation, starting with least translation at top and most translation at the bottom.

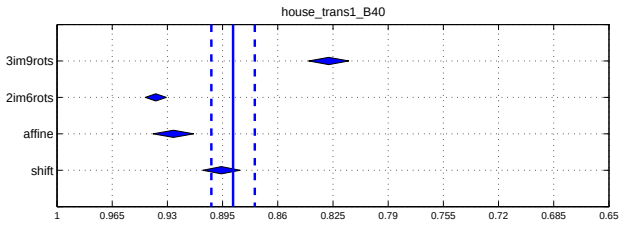

Fig. 17 Rectification results for sequence type #2.

8.3 Stabilisation of Rolling-Shutter Video

A fair evaluation of video stabilisation is very difficult to make, as the goal is to simultaneously reduce image-plane motions, and to maintain a geometrically correct scene. We have performed a simple evaluation that only computes the amount of image plane motion. In order to see the qualitative difference in preservation of scene geometry, we strongly encourage the reader to also have a look at the supplemental video material.

We evaluate image plane motion by comparing all neighbouring frames in a sequence using the error measure in (30). Each frame thus gets a stabilisation score which is the fraction of pixels that were accepted by (30) over the total number of pixels in the frame. An example of this accuracy computation is given in figure 18.

The comparisons are run on a video clip from an iPhone 3GS, where the person holding the camera was walking forward, see (Online Resource 01, 2011). The

Fig. 18 Stabilisation accuracy on iPhone 3GS sequence. Left to right: Frame #12, frame #13, accepted pixels (in white).

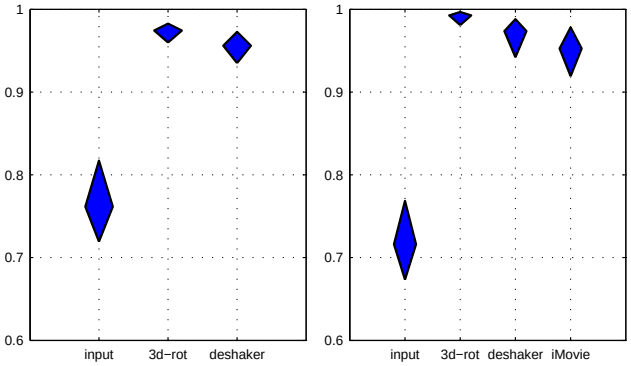

Fig. 19 Stabilisation accuracy on iPhone 3GS sequence. Left: stabilisation on original frames. Right: with extrapolation, and zoom to 135%. Diamond centres are median accuracy, tops and bottoms indicate the 25%, and 75% percentiles.

walking motion creates noticeable rolling shutter wobble at the end of each footstep.

We compare our results with the following stabilisation algorithms:

#1 Deshaker v 2.5 (Thalin, 2010). This is a popular and free rolling shutter aware video stabiliser. We have set the rolling shutter amount in Deshaker to 92.52% (See appendix A), and consistently chosen the highest quality settings.

#2 iMovie'09 v8.0.6. (Apple Inc., 2010). This is a video stabiliser that is bundled with MacOS X. We have set the video to 4:3, 30fps, and stabilisation with the default zoom amount 135%.

The rotation model is optimised over 2 frame windows, with $M = 6$ knots. The stabilised version using a Gaussian filter with $\sigma = 64$ can be seen in (Online Resource 03, 2011) and with extrapolation and 135% zoom in (Online Resource 04, 2011).

In figure 19, left we see a comparison with stabilisation in the input grid. This plot shows results from resampling single frames only, borders with unseen pixels remain in these clips (see also figure 18 for an illustration). As no extrapolation is used here, this experiment evaluates stabilisation in isolation. Borders are of similar size in both algorithms, so no algorithm should benefit from border effects.

The second experiment, see figure 19, right, shows results with border extrapolation turned on, and all sequences are zoomed to 135%. This setting produces video with few noticeable border effects, but at the price of discarding a significant amount of the input video.

The evaluation plots in figure 19, show that our algorithm is better than Deshaker 2.5 at stabilising image plane motion. iMovie'09, which is not rolling-shutter aware, falls significantly behind. However, we emphasise that this comparison tests for image-plane motion only. It is also our impression that the stabilised video has more of a 3D feel to it after processing with the 3D rotation method, than with Deshaker, see (Online Resource 04, 2011; Online Resource 06, 2011; Online Resource 07, 2011). The reason for this is probably that Deshaker uses an image plane distortion model.

9 Video examples

In the supplemental material we show results from three videos captured with three different cell-phones, and our result on Liu et al. (2011b) supplemental video set 1, example 1.

In section 8.3 we used a sequence captured with the iPhone 3GS where Online Resource 01 (2011) is the original video, Online Resource 02 (2011) is a 135% zoomed version, Online Resource 03 (2011) rectification and stabilisation with our method, Online Resource 04 (2011) with additional zoom and extrapolation, Online Resource 05 (2011) Deshaker results, Online Resource 06 (2011) Deshaker results with zoom and extrapolation and Online Resource 07 (2011) results from iMovie.

The supplemental material also contains videos captured from an HTC Desire at 25.87 Hz, see figure 20 left, and a SonyEricsson Xperia X10 at 28.54 Hz, right.

In the HTC Desire sequence the camera is shaken sideways while recording a person, see Online Resource 08 (2011) for the original video. Online Resource 09 (2011) is the result with our rectification and stabilisation method, while Online Resource 10 (2011) is stabilisation without rolling-shutter compensation. Online Resource 11 (2011) and Online Resource 12 (2011) are the results from Deshaker and iMovie respectively.

In the SonyEricsson Xperia X10 sequence the person holding the camera is walking while recording sideways, see Online Resource 13 (2011) for the original video. Online Resource 14 (2011) contains the result from our method, Online Resource 15 (2011) the result from Deshaker and Online Resource 16 (2011) the result from iMovie.

From these sequences it is our impression that our 3D rotation model is better at preserving the geometry of the scene. We can also see that Deshaker is significantly better than iMovie which is not rolling shutter

aware. Especially on Online Resource 15 (2011) we observe good results for Deshaker. Artifacts are mainly visible near the flag poles.

Liu et al. (2011a) demonstrated their stabilisation algorithm on rolling shutter video. The result can be seen on the supplemental material website, video set 1, example 1 (Liu et al., 2011b). The algorithm is not rolling shutter aware and distortions are instead treated as noise. The authors report that their algorithm does not handle artifacts like shear introduced by a panning motion but reduces the wobble from camera shake while stabilising the video (Liu et al., 2011a).

Our algorithm needs a calibrated camera, but the parameters are unfortunately not available for this sequence. To obtain a useful **K** matrix, the image centre was used as the projection of the optical centre and different values for the focal length and readout time were tested. As can be seen in Online Resource 17 (2011), the 3D structure of the scene is kept whereas the output of Liu et al. (2011a) still have a a noticeable amount of wobble. The walking up and down motion can still be seen in our result, and this is an artifact of not knowing the correct camera parameters.

Fig. 20 Left: First frame from the HTC Desire input sequence. Right: First frame from the SonyEricsson Xperia X10 input sequence.

10 Concluding Remarks

In this article, we have demonstrated rolling-shutter rectification by modelling the camera motion, and shown this to be superior to techniques that model movements in the image plane only. We even saw that image-plane techniques occasionally perform worse than the uncorrected baseline. This is especially true for motions that they do not model, e.g. rotations for the Global shift model (Chun et al., 2008).

Our stabilisation method is also better than iMovie and Deshaker when we compare the image plane motion. The main advantage of our stabilisation is that the visual appearance of the result videos and the geometry of the scene is much better. Our model also corrects for more types of camera motion than does mechanical image stabilisation (MIS).

In future work we plan to improve our approach by replacing the linear interpolation with a higher order spline defined on the rotation manifold, see e.g. Park and Ravani (1997). Another obvious improvement is to optimise parameters over full sequences. However, we wish to stress that our aim is currently to allow the algorithm to run on mobile platforms, which excludes optimisation over longer frame intervals than the 2-4 that we currently use.

For the stabilisation we have used a constant Gaussian filter kernel. For a dynamic video with different kinds of motions, e.g. camera shake and panning, the result would benefit from adaptive filtering.

In general, the quality of the reconstruction should benefit from more measurements. In MIS systems, camera rotations are measured by MEMS gyro sensors (Bernstein, 2003). Such sensors are now starting to appear in cell-phones, e.g. the recently introduced iPhone4. It would be interesting to see how such measurements could be combined with measurements from KLT-tracking when rectifying and stabilising video.

A Readout Time Calibration

The setup for calibration of readout time is shown in figure 21, top left. An LED is powered from a function generator set to produce a square pulse. The frequency f_o of the oscillation is measured by an oscilloscope. Due to the sequential readout on the RS chip, the acquired images will have horizontal stripes corresponding to on and off periods of the LED, see figure 21, top right. If we measure the period T of the vertical image oscillation in pixels, the readout time can be obtained as:

$$t_r = N_r/(Tf_o)\,, \tag{32}$$

where N_r is the number of image rows, and f_o is the oscillator frequency, as estimated by the oscilloscope.

As can be seen in figure 21 top right, the images obtained in our setup suffer from inhomogeneous illumination, which complicates estimation of T. In their paper, Geyer et al. (2005) suggest removing the lens, in order to get a homogeneous illumination of the sensor. This is difficult to do on cell-phones, and thus we instead recommend to collect a sequence of images of the flashing LED, and then subtract the average image from each of these. This removes most of the shading seen in the input camera image, see figure 21, middle left, for an example. We then proceed by averaging all columns in each image, and removing their DC. An image formed by stacking such rows from a video is shown in figure 21, middle right. We compute Fourier coefficients for this image along the y-axis, and average their magnitudes to obtain a 1D Fourier spectrum, $F(u)$, shown in figure 21, bottom left.

$$F(u) = \frac{1}{N_f}\sum_{x=1}^{N_f}\left|\frac{1}{N_r}\sum_{y=1}^{N_r} f(y,x)e^{-i2\pi uy/N_r}\right|\,. \tag{33}$$

Here N_f is the number of frames in the sequence. We have consistently used $N_f = 40$. We refine the strongest peak of (33),

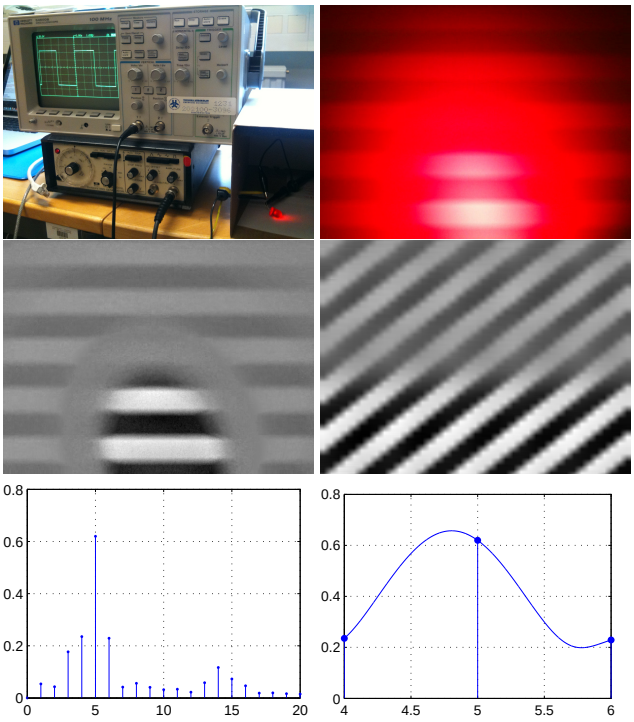

Fig. 21 Calibration of a readout time for a rolling-shutter camera. Top left: Used equipment; an oscilloscope, a function generator and a flashing LED. Top right: One image from the sequence. Middle left: Corresponding image after subtraction of the temporal average. Middle right: Image of average columns for each frame in sequence. Bottom left: 1D spectrum from (33), Bottom right: refinement of peak location.

by also evaluating the function for non-integer values of u, see figure 21, bottom right. The oscillation period is found from the frequency maximum u^*, as $T = N_r/u^*$.

As the Geyer calibration is a bit awkward (it requires a function generator, an oscilloscope and an LED), we have reproduced the calibration values we obtained for a number of different cell-phone cameras in table 1. The "<30" value for frame rate in the table signifies a variable frame rate camera with an upper limit somewhere below 30. The oscillator frequency is only estimated with three significant digits by our oscilloscope, and thus we have averaged obtained readout times for several different oscillator frequencies in order to improve the accuracy.

Reported readout times marked with an "*" are one out of several used by the camera. We have seen that a change of focus, or gain may provoke a change in readout time on those cameras. We have found that rectifications using the listed value look better than those from other readout times, and thus it appears that this value is used most of the time.

The Deshaker webpage (Thalin, 2010) reports rolling shutter amounts ($a = t_r \times f$) for a number of different rolling shutter cameras. The only cell-phone currently included is the iPhone4, which is reported as $a = 0.97 \pm 0.02$. Converted to a readout time range, this becomes $t_r \in [31.67\text{ms}, 33\text{ms}]$ which is the range where we find our two iPhone4 units.

Acknowledgements We would like to thank SonyEricsson Research Centre in Lund for providing us with a SonyEricsson Xperia X10, and Anders Nilsson at Computer Engineering for lending us the equipment for readout time calibration. This work is funded by the CENIIT organisation at Linköping Institute of Technology, and by the Swedish Research foundation.

Camera	resolution	f	t_r	N	σ
iPhone 3GS	640 × 480	30	30.84	7	0.36
iPhone 4(#1)	1280 × 720	30	31.98	6	0.25
iPhone 4(#2)	1280 × 720	30	32.37	4	0.27
iPhone 4(#1,front)	640 × 480	30	29.99	5	0.16
iTouch 4	1280 × 720	30	30.07	5	0.25
iTouch 4 (front)	640 × 480	30	31.39	5	0.07
HTC Desire	640 × 480	< 30	57.84*	3	0.051
HTC Desire	1280 × 720	< 30	43.834*	4	0.034
SE W890i	320 × 240	14.706	60.78	4	0.16
SE Xperia X10	640 × 480	< 30	28.386*	4	0.024
SE Xperia X10	800 × 480	< 30	27.117*	5	0.040

Table 1 Calibrated readout times for a selection of cell-phone cameras with rolling shutters. f (Hz) - frame rate reported by manufacturer. t_r (milliseconds) average readout time, N number of estimates, σ standard deviation of t_r measurements. Readout times marked by an "*" are variable. See text for details.

References

Ait-Aider O, Berry F (2009) Structure and kinematics triangulation with a rolling shutter stereo rig. In: IEEE International Conference on Computer Vision

Ait-Aider O, Bartoli A, Andreff N (2007) Kinematics from lines in a single rolling shutter image. In: CVPR'07, Minneapolis, USA

Apple Inc (2010) iMovie'09 video stabilizer. `http://www.apple.com/ilife/imovie/`

Baker S, Scharstein D, Lewis JP, Roth S, Black MJ, Szeliski R (2007) A database and evaluation methodology for optical flow. In: IEEE International Conference on Computer Vision (ICCV07), Rio de Janeiro, Brazil

Baker S, Bennett E, Kang SB, Szeliski R (2010) Removing rolling shutter wobble. In: IEEE Conference on Computer Vision and Pattern Recognition, IEEE Computer Society, IEEE, San Francisco, USA

Bernstein J (2003) An overview of MEMS inertial sensing technology. Sensors Magazine 2003(1)

Buehler C, Bosse M, McMillan L (2001) Non-metric image-based rendering for video stabilization. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'01), pp 609–614

Chang LW, Liang CK, Chen H (2005) Analysis and compensation of rolling shutter distortion for CMOS image sensor arrays. In: International Symposium on Communications (ISCOM05)

Cho WH, Kong KS (2007) Affine motion based CMOS distortion analysis and CMOS digital image stabilization. IEEE Transactions on Consumer Electronics 53(3):833–841

Cho WH, Kim DW, Hong KS (2007) CMOS digital image stabilization. IEEE Transactions on Consumer Electronics 53(3):979–986

Chun JB, Jung H, Kyung CM (2008) Suppressing rolling-shutter distortion of CMOS image sensors by motion vector detection. IEEE Transactions on Consumer Electronics 54(4):1479–1487

Forssén PE, Ringaby E (2010) Rectifying rolling shutter video from hand-held devices. In: IEEE Conference on Computer Vision and Pattern Recognition, IEEE Computer Society, IEEE, San Francisco, USA

Gamal AE, Eltoukhy H (2005) CMOS image sensors. IEEE Circuits and Devices Magazine

Geyer C, Meingast M, Sastry S (2005) Geometric models of rolling-shutter cameras. In: 6th OmniVis WS

Gramkow C (2001) On averaging rotations. International Journal of Computer Vision 42(1/2):7–16

Green R, Mahler H, Siau J (1983) Television picture stabilizing system. US Patent 4,403,256

Harris CG, Stephens M (1988) A combined corner and edge detector. In: 4th Alvey Vision Conference, pp 147–151

Hartley R, Zisserman A (2000) Multiple View Geometry in Computer Vision. Cambridge University Press

Liang CK, Chang LW, Chen H (2008) Analysis and compensation of rolling shutter effect. IEEE Transactions on Image Processing 17(8):1323–1330

Liu F, Gleicher M, Jin H, Agarwala A (2009) Content-preserving warps for 3D video stabilization. In: ACM Transactions on Graphics

Liu F, Gleicher M, Wang J, Jin H, Agarwala A (2011a) Subspace video stabilization. ACM Transactions on Graphics 30(1)

Liu F, Gleicher M, Wang J, Jin H, Agarwala A (2011b) Subspace video stabilization, supplemental video set 1. http://web.cecs.pdx.edu/~fliu/project/subspace_stabilization/

Lucas B, Kanade T (1981) An iterative image registration technique with an application to stereo vision. In: IJCAI81, pp 674–679

Morimoto C, Chellappa R (1997) Fast 3D stabilization and mosaic construction. In: Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR97), San Juan, Puerto Rico, pp 660–665

Nicklin SP, Fisher RD, Middleton RH (2007) Rolling shutter image compensation. In: Robocup 2006 LNAI 4434, pp 402–409

Online Resource 01 (2011) iPhone 3GS input sequence, ESM_1.mpg

Online Resource 02 (2011) iPhone 3GS 135% zoomed input, ESM_2.mpg

Online Resource 03 (2011) iPhone 3GS rectification and stabilisation, ESM_3.mpg

Online Resource 04 (2011) iPhone 3GS rectification and stabilisation with 135% zoom and extrapolation, ESM_4.mpg

Online Resource 05 (2011) iPhone 3GS Deshaker, ESM_5.mpg

Online Resource 06 (2011) iPhone 3GS Deshaker with 135% zoom and extrapolation, ESM_6.mpg

Online Resource 07 (2011) iPhone 3GS iMovie with 135% zoom and extrapolation, ESM_7.mpg

Online Resource 08 (2011) HTC Desire input sequence, ESM_8.mpg

Online Resource 09 (2011) HTC Desire rectification and stabilisation, ESM_9.mpg

Online Resource 10 (2011) HTC Desire stabilisation only, ESM_10.mpg

Online Resource 11 (2011) HTC Desire Deshaker, ESM_11.mpg

Online Resource 12 (2011) HTC Desire iMovie with 135% zoom and extrapolation, ESM_12.mpg

Online Resource 13 (2011) SonyEricsson Xperia X10 input sequence, ESM_13.mpg

Online Resource 14 (2011) SonyEricsson Xperia X10 rectification and stabilisation, ESM_14.mpg

Online Resource 15 (2011) SonyEricsson Xperia X10 Deshaker, ESM_15.mpg

Online Resource 16 (2011) SonyEricsson Xperia X10 iMovie with 135% zoom and extrapolation, ESM_16.mpg

Online Resource 17 (2011) Rectification and stabilisation results on sequence: Liu et al. 2011 Supplemental Video Set 1, example 1 , ESM_17.mpg

Park F, Ravani B (1997) Smooth invariant interpolation of rotations. ACM Transactions on Graphics 16(3):277–295

Ringaby E (2010) Rolling shutter dataset with ground truth. http://www.cvl.isy.liu.se/research/rs-dataset

Shi J, Tomasi C (1994) Good features to track. In: IEEE Conference on Computer Vision and Pattern Recognition, CVPR'94, Seattle

Shoemake K (1985) Animating rotation with quaternion curves. In: Int. Conf. on CGIT, pp 245–254

Thalin G (2010) Deshaker video stabilizer plugin v2.5 for VirtualDub. http://www.guthspot.se/video/deshaker.htm

Whyte O, Sivic J, Zisserman A, Ponce J (2010) Non-uniform deblurring for shaken images. In: IEEE Conference on Computer Vision and Pattern Recognition, IEEE Computer Society, IEEE, San Francisco, USA

Yao YS, Burlina P, Chellappa R, Wu TH (1995) Electronic image stabilization using multiple visual cues. In: International Conference on Image Processing (ICIP'95), Washington DC, USA, pp 191–194

Zhang Z (2000) A flexible new technique for camera calibration. IEEE Transactions on Pattern Analysis and Machine Intelligence 22(11):1330–1334