Rectifying rolling shutter video from hand-held devices

Per-Erik Forssén and Erik Ringaby
Department of Electrical Engineering
Linköping University, Sweden
{perfo,ringaby}@isy.liu.se

Abstract

This paper presents a method for rectifying video sequences from rolling shutter (RS) cameras. In contrast to previous RS rectification attempts we model distortions as being caused by the 3D motion of the camera. The camera motion is parametrised as a continuous curve, with knots at the last row of each frame. Curve parameters are solved for using non-linear least squares over inter-frame correspondences obtained from a KLT tracker. We have generated synthetic RS sequences with associated ground-truth to allow controlled evaluation. Using these sequences, we demonstrate that our algorithm improves over to two previously published methods. The RS dataset is available on the web to allow comparison with other methods.

1. Introduction

Today consumer products that allow video capture are quite common. Examples are cell-phones, music players, and regular cameras. Most of these devices, as well as camcorders in the consumer price range, have CMOS image sensors. CMOS sensors have several advantages over the conventional CCD sensors: they are cheaper to manufacture, and typically offer on-chip processing [9], for e.g. automated white balance and auto-focus measurements. 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 distortions caused by using a rolling shutter, and how this frame is rectified by our proposed method, as well as two others.

Figure 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. Videos are available on the web and in the supplementary material.

1.1. Related work

A camera motion between two points in time can be described with a three element translation vector, and a 3DOF (degrees-of-freedom) rotation. For hand-held footage, the rotation component is typically the dominant cause of image plane motion. (A notable exception to this 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 cam-

corders lack MIS. There is thus a large volume of video out there, that 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 [5, 13, 16, 6, 7, 8]. Some algorithms assume that the image deformation is caused by a globally constant translational motion across the image [5, 16, 8]. After rectification this would correspond to a constant optical flow across the entire image, which is rare in practise. Liang et al. [13] 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 et al. [6, 7]. Here geometric distortion is modelled as a global affine deformation that is parametrised by the scan-line index.

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 preferred in the related problem of video stabilisation [14].

Other related work on RS images include structure and motion estimation. Geyer et al. [10] 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. 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 [1]. 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 [2].

1.2. Contributions

All the previous approaches to rectification of RS video [5, 13, 16, 6, 7, 8] model distortions as taking place in the image plane. We instead model the 3D camera motion using calibrated projective geometry. We introduce two models, one purely rotational, and one with rotation and translation with respect to an estimated plane in the scene. In projective geometry terms, these can be thought of as a sequence of parametrised homographies, one for each image row.

This far, no controlled comparison of RS algorithms have been published. Instead each new algorithm has just been published together with images that show how well images distorted by particular motions can be rectified. In related fields such as stereo, and optical flow computation [3], evaluation datasets have been important for ensuring that new algorithms actually improve on previous ones. For these reasons we have generated synthetic RS sequences, with associated ground-truth rectifications. Using these sequences, we compare our own implementations of the global affine model [6], and the global shift model [8] to the new method that we propose. Our dataset and supplementary videos are available for download at [18].

1.3. Overview

This paper is organised as follows: In section 2, we describe how to calibrate a rolling-shutter camera, and introduce models and cost functions for camera ego-motion estimation. In section 3 we discuss interpolation schemes for rectification of rolling-shutter imagery. In section 4 we describe our evaluation dataset. In section 5 we use our dataset to compare different interpolation schemes, and to compare our camera ego-motion approach to our own implementations of [13] and [8]. The paper concludes with outlooks and concluding remarks in section 6.

2. Camera motion estimation

In this paper, 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.

2.1. Rolling shutter camera calibration

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

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

where $\mathbf{K}$ is a 5DOF upper triangular 3×3 *intrinsic camera matrix*, and λ is an unknown scaling [12]. We estimate $\mathbf{K}$ using the Zhang method in OpenCV. [21].

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: $1/f = t_r + t_d$. The readout time can be calibrated by imaging a flashing light source with known frequency [10], see figure 2, left. If we measure the period T of the vertical oscillation in pixels, t_r can be obtained as:

$$t_r = N_r/(Tf_o)\,, \tag{2}$$

where N_r is the number of image rows, and f_o is the oscillator frequency. The inter-frame delay can now be computed as $t_d = 1/f - t_r$. For our purposes it is preferable to use rows as fundamental unit, and express the inter-frame delay as a number of *blank rows*:

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

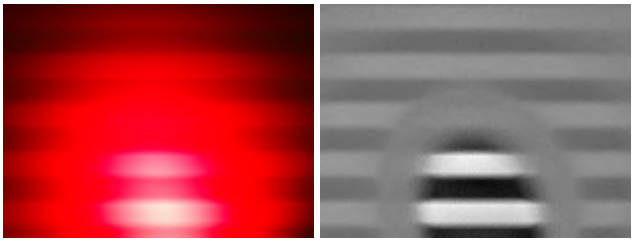

Figure 2. Calibration of a rolling-shutter camera. Left: Image of a flashing LED used for calibration of the readout time. Right: Corresponding image after subtraction of the temporal average.

Camera	f_o	t_r
W890i	5.02	60.94
	6.5	60.76
	7.5	60.88
	avg.	60.86

Camera	f_o	t_r
3GS	4.01	30.54
	5.01	31.22
	6.5	30.5
	avg.	30.75

Table 1. Used oscillator frequencies f_o, and obtained readout times t_r, for the SonyEricsson W890i and Apple iPhone 3GS camera phones. Units are milliseconds.

We have performed both Zhang [21], and Geyer [10] calibrations for the cameras built into the Apple iPhone 3GS, and the SonyEricsson W890i cell phones. As the Geyer calibration is a bit awkward (it requires a signal generator, an oscilloscope and an LED), we have reproduced the calibration values we obtained in table 1. The camera frame rates are $f = 30$ Hz for the 3GS, and $f = 14.7059$ Hz for the W890i (according to manufacturer specifications).

In his paper [10], Geyer suggests removing the lens, in order to get a homogeneous illumination of the sensor. This is difficult to do on cellphones, 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, see figure 2, right. This removes most of the shading seen in figure 2, left, and allows us to find the oscillation period from the first frequency above DC.

2.2. Rolling shutter camera under pure rotation

Our first model of camera motion is a rotation about the camera centre during frame capture, in a smooth, but time varying way. We represent this 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:

$$\mathbf{x} = \mathbf{K}\mathbf{R}(t_1)\mathbf{X}\,, \text{ and } \quad \mathbf{y} = \mathbf{K}\mathbf{R}(t_2)\mathbf{X}\,. \tag{4}$$

This gives us the relation:

$$\mathbf{x} = \mathbf{K}\mathbf{R}(t_1)\mathbf{R}^T(t_2)\mathbf{K}^{-1}\mathbf{y}\,. \tag{5}$$

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 $t_1 = x_2/x_3$. In the second image we get $t_2 = y_2/y_3 + N_r + N_b$, where N_r is the number of image rows, and N_b is defined in (3).

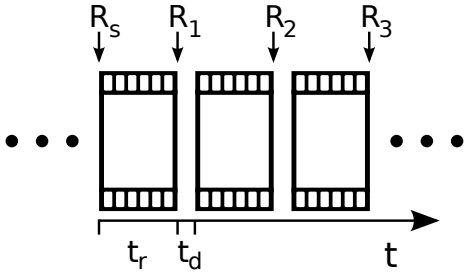

Figure 3. Rotations, $\mathbf{R}_1$, $\mathbf{R}_2, \ldots$ are estimated for last rows of each frame. Intermediate rotations are interpolated from these and $\mathbf{R}_s$. Readout time t_r, and inter-frame delay t_d are also shown.

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 spline with knots at the last row of each frame, see figure 3. Intermediate rotations are found using spherical linear interpolation [20].

As we need a reference world frame, we might as well fixate that to the start of frame 1, i.e. set $\mathbf{R}_s = \mathbf{I}$. This gives us $3N$ unknowns in total for a group of N frames. These can be resolved if we have at least three correspondences between each pair of views.

2.3. Rolling shutter imaging of a planar scene

In our second camera motion model, we assume that we are imaging a purely planar scene. We now model the motion as a sequence of translations $\mathbf{d}(t) \in \mathbb{R}^3$, and rotations $\mathbf{R}(t) \in \mathrm{SO}(3)$, with respect to a coordinate system located on the world plane. The world coordinate system needs not be explicitly estimated, it suffices to know that we can choose it such that the 3D points have a zero third coordinate, i.e. $(0\ 0\ 1)\,\mathbf{X} = 0$. The projection of such a point in the image, after a translation $\mathbf{d}(t_1)$, and a rotation $\mathbf{R}(t_1)$, can be written:

$$\mathbf{x} = \mathbf{K}\mathbf{R}(t_1)(\mathbf{X} + \mathbf{d}(t_1)) = \mathbf{K}\mathbf{R}(t_1)\mathbf{D}(t_1)\tilde{\mathbf{X}}\,, \tag{6}$$

$$\text{where} \quad \mathbf{D} = \begin{pmatrix} 1 & 0 & d_1 \\ 0 & 1 & d_2 \\ 0 & 0 & d_3 \end{pmatrix}\,, \tag{7}$$

and $\tilde{\mathbf{X}}$ is a three element vector containing the non-zero elements of $\mathbf{X}$, and a 1 in the third position.

Since (6) is invertible, in the same sense as (1), we can

relate the projections of the 3D point in two images as:

$$\mathbf{x} = \mathbf{K}\mathbf{R}(t_1)\mathbf{D}(t_1)\mathbf{D}(t_2)^{-1}\mathbf{R}(t_2)^T\mathbf{K}^{-1}\mathbf{y}\,. \quad (8)$$

Note that by setting $\mathbf{D}(t_1) = \mathbf{D}(t_2) = \mathbf{I}$ we obtain the pure rotation model (5) as a special case.

In contrast to the pure rotation case, we have now a variable origin, so we need also to find $\mathbf{R}_s$ and $\mathbf{d}_s$. However, we note that a point in the world plane, expressed in normalised camera coordinates $\mathbf{X}_c = \lambda\mathbf{K}^{-1}\mathbf{x}$, has to satisfy a plane equation:

$$\hat{\mathbf{r}}^T\mathbf{X}_c - \rho = \hat{\mathbf{r}}^T(\mathbf{X}_c - \hat{\mathbf{r}}\rho) = 0\,. \quad (9)$$

We now let this equation define the transformation from the camera to the third (zero valued) world coordinate:

$$\mathbf{X} = \mathbf{R}_s^T(\mathbf{X}_c - \hat{\mathbf{r}}\rho) \;\text{ for }\; \mathbf{R}_s = \begin{pmatrix}\hat{\mathbf{r}}_\perp & \hat{\mathbf{r}}\times\hat{\mathbf{r}}_\perp & \hat{\mathbf{r}}\end{pmatrix}. \quad (10)$$

This gives us the projection from the plane into the camera as:

$$\mathbf{X}_c = \mathbf{R}_s(\mathbf{X} + (0\;0\;\rho)^T)\,. \quad (11)$$

Finally, as a monocular reconstruction is only defined up to scale, we can fixate the plane at $\rho = 1$. This locks the translation to $\mathbf{d}_s = (0\;0\;1)^T$, and we thus only get the extra 3 parameters in $\mathbf{R}_s$.

Just like in the pure rotation case, each correspondence gives us two equations, but now we have $6N+3$ unknowns for a group of N frames. These can be determined if we have at least $3N+2$ correspondences, and at least 6 correspondences between each pair of frames (8 is required for the first pair).

2.4. Pure translational motion

It is also possible to constrain the planar scene model to translations only. For this we simply set all rotation matrices equal to the first, i.e $\mathbf{R}_n = \mathbf{R}_s\ \forall n \in [1, N]$. This gives us $3N+3$ unknowns, which again requires at least 3 correspondences between each pair of frames.

2.5. Motion interpolation

We interpolate the translational component of the camera motion, $\mathbf{d}(t) \in \mathbb{R}^3$ in-between two key translations $\mathbf{d}_1$, $\mathbf{d}_2$, using regular linear interpolation. Using a parameter $w \in [0, 1]$, this can be written as:

$$\mathbf{d}_{\text{interp}} = (1-w)\mathbf{d}_1 + w\mathbf{d}_2\,. \quad (12)$$

For the rotational component, the situation is more complicated, due to the periodic structure of SO(3).

We have chosen to represent rotations as three element vectors where the magnitude corresponds to the rotation angle, and the direction 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:

$$\mathbf{R} = \text{expm}(\mathbf{n}) = \mathbf{I} + [\hat{\mathbf{n}}]_x \sin\phi + [\hat{\mathbf{n}}]_x^2(1-\cos\phi) \quad (13)$$

$$\text{where } [\hat{\mathbf{n}}]_x = \frac{1}{\phi}\begin{pmatrix}0 & -n_3 & n_2\\ n_3 & 0 & -n_1\\ -n_2 & n_1 & 0\end{pmatrix}. \quad (14)$$

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 (13) are symmetric, and thus terms of the form $r_{ij} - r_{ji}$ will come from the anti-symmetric part alone. This allows us to extract the axis and angle as:

$$\mathbf{n} = \text{logm}(\mathbf{R}) = \phi\hat{\mathbf{n}}\,, \quad \text{where} \quad \begin{cases}\tilde{\mathbf{n}} = \begin{pmatrix}r_{32}-r_{23}\\ r_{13}-r_{31}\\ r_{21}-r_{12}\end{pmatrix}\\ \phi = \sin^{-1}(||\tilde{\mathbf{n}}||/2)\\ \hat{\mathbf{n}} = \tilde{\mathbf{n}}/||\tilde{\mathbf{n}}||\,.\end{cases} \quad (15)$$

It is also possible to extract the rotation angle from the trace of $\mathbf{R}$ [17]. We recommend (15), as it avoids numerical problems for small angles. Using (13) and (15), we can perform SLERP (Spherical Linear intERPolation) [20] between two rotations $\mathbf{n}_1$ and $\mathbf{n}_2$, using an interpolation parameter $w \in [0, 1]$ as follows:

$$\mathbf{n}_{\text{diff}} = \text{logm}\left(\text{expm}(-\mathbf{n}_1)\text{expm}(\mathbf{n}_2)\right) \quad (16)$$

$$\mathbf{R}_{\text{interp}} = \text{expm}(\mathbf{n}_1)\text{expm}(w\mathbf{n}_{\text{diff}}) \quad (17)$$

2.6. Optimisation

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

$$J = \epsilon(\mathbf{n}_1, \ldots, \mathbf{n}_N) \;\text{ or} \quad (18)$$

$$J = \epsilon(\mathbf{n}_s, \mathbf{n}_1, \ldots, \mathbf{n}_N, \mathbf{d}_1, \ldots, \mathbf{d}_N)\,, \quad (19)$$

for the pure rotation, and the planar scene models respectively. To this end, we choose to minimise the (symmetric) image-plane residuals of the set of corresponding points $\mathbf{x}_k \leftrightarrow \mathbf{y}_k$:

$$J = \sum_{k=1}^{K} d(\mathbf{x}_k, \mathbf{H}\mathbf{y}_k)^2 + d(\mathbf{y}_k, \mathbf{H}^{-1}\mathbf{x}_k)^2 \quad (20)$$

$$\text{where } \mathbf{H} = \mathbf{K}\mathbf{R}(\mathbf{x}_k)\mathbf{R}^T(\mathbf{y}_k)\mathbf{K}^{-1} \;\text{ or} \quad (21)$$

$$\mathbf{H} = \mathbf{K}\mathbf{R}(\mathbf{x}_k)\mathbf{D}(\mathbf{x}_k)\mathbf{D}(\mathbf{y}_k)^{-1}\mathbf{R}^T(\mathbf{y}_k)\mathbf{K}^{-1} \quad (22)$$

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

$$d(\mathbf{x}, \mathbf{y})^2 = (x_1/x_3 - y_1/y_3)^2 + (x_2/x_3 - y_2/y_3)^2 \,. \quad (23)$$

The rotation matrices are obtained as:

$$\mathbf{R}(\mathbf{x}) = \texttt{SLERP}(\mathbf{n}_1, \mathbf{n}_2, w) \,, \text{ for } \; w = x_2/(x_3 N_r), \quad (24)$$

where SLERP is defined in (16,17), and N_r is the number of image rows.

In our experiments, we have minimised (20) using the MATLAB optimiser `lsqnonlin`. Rewriting the optimisation in e.g. C should however be done if real-time operation is to be achieved.

For speed, we have chosen to optimise over short intervals of $N = 2$, 3 or 4 frames. For the pure rotation model, 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 3), i.e.

$$\mathbf{R}_o = \texttt{SLERP}(\mathbf{n}_1, \mathbf{n}_2, N_b/(N_r + N_b)) \,. \quad (25)$$

Then we shift the interval one step, correct for the change of origin, and use the previous rotations as initialisations

$$\mathbf{R}'_n = \mathbf{R}_o^T \mathbf{R}_{n+1} \,, \quad \text{for } \; n = 1, \ldots, N \,. \quad (26)$$

As initialisation of the rotations in newly shifted-in frames, we use identity rotations.

In the planar scene model, we initialise the rotations to identity rotations, and the translations to $\mathbf{d}_n = (0\ 0\ 1)^T \ \forall n \in [1, N]$.

2.7. Point correspondences

The point correspondences needed to estimate the rotations are obtained through point tracking. First, Harris-points [11] are detected in the current frame and these are tracked using the KLT tracker [15, 19]. 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 track-re-track procedure is used [3]. 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. Image rectification

Once we have found our sequence of rotation matrices, we can use them to rectify the images in the sequence. Each row gets its own rotation according to (24). We can then align them to a reference row $\mathbf{R}_o$ (typically the middle row), using:

$$\mathbf{R}'(\mathbf{x}) = \mathbf{R}_o \mathbf{R}^T(\mathbf{x}) \,. \quad (27)$$

This gives us the forward mapping as:

$$\mathbf{x}' = \mathbf{K}\mathbf{R}_o\mathbf{R}^T(\mathbf{x})\mathbf{K}^{-1}\mathbf{x} \quad (28)$$

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 in three steps: First, we create an all-zero RGBA image. Second, we apply (28) to each pixel in the RS image. The 3×3 closest grid locations are then updated by adding vectors of the form (wr, wg, wb, w). Here r, g, b are the colour channel values of the input pixel, and w is a variable weight that depends on the grid location $\mathbf{u}$, according to:

$$w(\mathbf{u}) = \exp(-.5||\mathbf{u} - \tilde{\mathbf{x}}'||^2/\sigma^2) \,. \quad (29)$$

Here $\tilde{\mathbf{x}}' = (x'_1/x'_3 \ \ x'_2/x'_3)^T$ is the sub-pixel location of the pixel, and σ is a smoothing parameter, which we set to $\sigma = 0.15$. Third, after looping through all pixels, we convert the RGBA image to RGB, by dividing the RGB values by the fourth element. This *forward interpolation* scheme is quite fast, and its parallel nature makes it well suited to a GPU implementation.

Alternatively, the irregular grid of pixels can be resampled to a regular grid, by defining a triangular mesh over the points, and sampling the mesh using bicubic interpolation. This is done by the function `griddata` in Matlab.

Finally, it is also tempting to use regular, or *inverse interpolation*, i.e. invert (28) to obtain:

$$\mathbf{x} = \mathbf{K}\mathbf{R}(\mathbf{x})\mathbf{R}_o^T\mathbf{K}^{-1}\mathbf{x}' \,. \quad (30)$$

We can now loop over all values of $\mathbf{x}'$, and use (30) to find the pixel locations in the distorted image, and cubically interpolate these.

4. Synthetic dataset

In order to do a controlled evaluation of algorithms for RS compensation we have generated six test sequences (available at [18]), 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 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 masks come in three variants: for rectification to the time instant when the first, middle and last row of the RS frame were imaged.

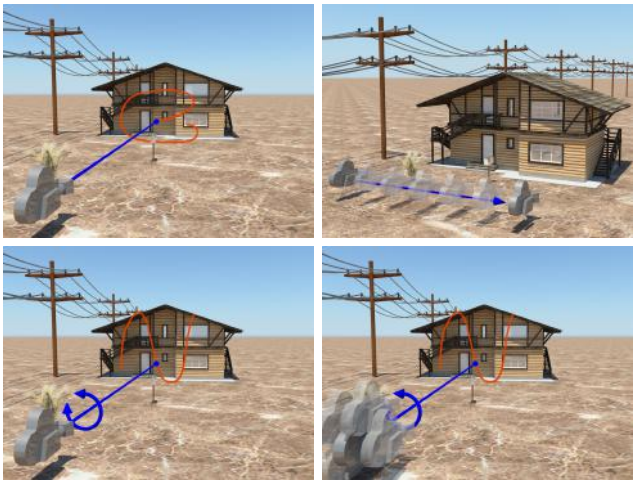

Figure 4. The four categories of synthetic sequences.

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 4.

In the first sequence type, the camera rotates around its centre in a spiral fashion, see figure 4 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).

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 4 top right.

In the last two sequence types the camera makes an up/down rotating movement, with a superimposed rotation from left to right, see figure 4 bottom left. There is also a back-and-forth rotation with the viewing direction as axis. The last sequence type is the same as the third except that a translation parallel to the image plane has been added, see figure 4 bottom right.

For each frame in the ground-truth sequences, we have created masks that indicate pixels that can be reconstructed from the corresponding RS frame, see figure 5. 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).

Figure 5. 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 mask with white for ground-truth pixels that were seen in the RS frame.

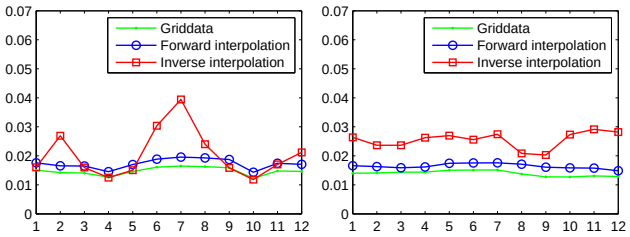

Figure 6. Comparison of interpolation scheme errors. The plots show the average Euclidean pixel distance between interpolated images and rendered ground truth for each frame in sequence type #1, $N_b = 0$ (left), and type #3, $N_b = 40$ (right).

5. Experiments

5.1. Interpolation accuracy

We have compared the errors of the three interpolation approaches described in section 3, in figure 6. Here we have used known ground-truth rotations to rectify each frame in two pure camera rotation sequences, sequence type #1, with $N_b = 0$, and sequence type #3, with $N_b = 40$ (see section 4 for a description of the sequences). We have used two pure rotation sequences, as for these an almost perfect reconstruction is possible, and thus the errors shown are due to interpolation only. The error measure used is average Euclidean distance to the RGB pixel values in the ground truth images, within the valid mask.

In some frames, the methods differ quite a bit, while in others they are very similar. The reason for this is that only for larger rotations, do the neighbours in the distorted and undistorted images start to differ. As can be seen in figure 6, `griddata` and our forward interpolation are superior to inverse sampling. Among the three methods, `griddata` stands out, by being approximately $40\times$ more expensive on 640×480 images. As our forward interpolation scheme is both fast and accurate, we recommend it over the other methods.

For very fast motions, and a slow rolling shutter, the 3×3 grid used in forward interpolation may be too small. The interpolated image would then have pixels where the value is undefined. In our experiments on real video we have however not experienced this. Should this effect occur, one could simply increase the grid size to 5×5.

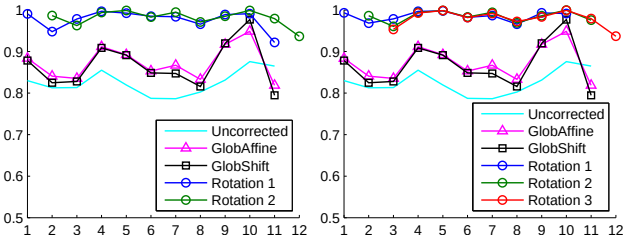

Figure 7. Sequence type #1 (rotation only), $N_b = 0$. Left: GA, GS, and uncorrected frames, against our **rotation model** with 2-frame reconstruction window. Right: GA, GS, and uncorrected frames, against our **rotation model** with 3-frame window.

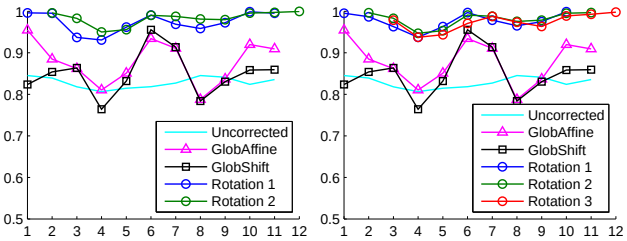

Figure 8. Sequence type #3 (rotation only), $N_b = 40$. Left: GA, GS, and uncorrected frames, against our **rotation model** with 2-frame reconstruction window. Right: GA, GS, and uncorrected frames, against our **rotation model** with 3-frame window.

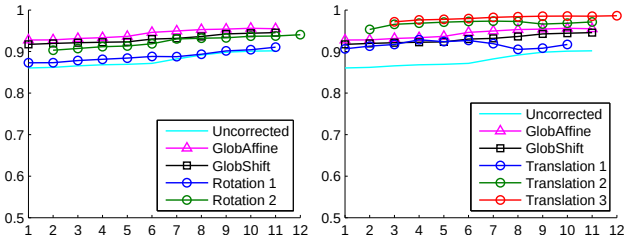

Figure 9. Sequence type #2 (translation only), $N_b = 40$. Left: GA, GS, and uncorrected frames, against our **rotation model** with 2-frame reconstruction window. Right: GA, GS, and uncorrected frames, against our **translation model** with 3-frame window.

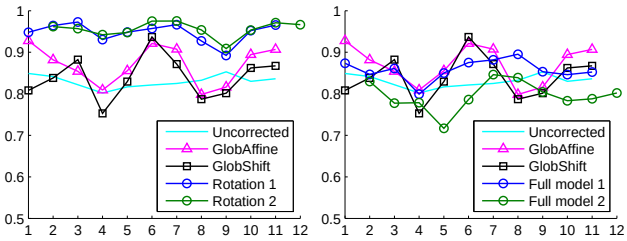

Figure 10. Sequence type #4 (translation and rotation), $N_b = 40$. Left: GA, GS, and uncorrected frames, against our **rotation model** with 2-frame reconstruction window. Right: GA, GS, and uncorrected frames, against our **full model** with 2-frame window.

5.2. Rectification accuracy

We have compared our methods to the *global affine model* (GA) [6], and the *global shift model* (GS) [8] on our synthetic sequences, see section 4. The comparison is done using thresholded Euclidean colour distance. Pixels that deviate more than $d_{\text{thr}} = 0.3$ are counted as incorrect. We have also tried other threshold values, and while the exact choice changes the locations of the curves, it does not change their order (for reasonable values of d_{thr}). As evaluation measure we use the fraction of correctly reconstructed pixels within the mask of valid locations. 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, with respect to the first frame ground-truth.

As our reconstruction solves for several cameras in each frame interval, we have simply chosen to present all of them in the following plots. E.g. Rotation 1, 2, and 3 in figure 7 are the three solutions in a 3-frame reconstruction.

In figure 7 we compare the GA, and GS methods with our pure rotation model. The sequence used is type #1 (rotation only), with $N_b = 0$. As can be seen, our methods do better than GA, GS, and the baseline.

In figure 8 we compare the GA, and GS methods with our pure rotation model on sequence type #3 (rotation only), with $N_b = 40$. As can be seen our methods do better than both GA, GS, and the baseline. GA, and GS, have problems with this sequence, and sometimes fall below the baseline. In general, other values of N_b give very similar results for our methods. For GA and GS the variability is larger, but we have not seen any consistent degradation or improvement.

In figure 9, left, we compare the GA, and GS methods with our pure rotation model. The sequence used is type #2 (translation only), with $N_b = 40$. As can be seen our methods do slightly worse than GA and GS, but they still improve on the uncorrected input. In figure 9, right, we compare GA, GS, and our translation-only model. The translation reconstruction for the first frame is still worse than GA and GS, but the other two do significantly better.

In figure 10 we have compared GA, GT, with our rotation only model (left) and with the full model (right). As can be seen, the rotation only model does consistently better than the others. Note that the full model currently does worse than the rotation only model. When we gave the optimiser different starting points, (e.g. the result from the rotation model) we obtained different solutions, thus we conclude that the cost function for the full model is not convex. A better initialisation may solve this problem, but this is out of the scope of this paper.

5.3. Stabilisation of rolling-shutter video

We have done a simple comparison of RS compensation algorithms on real imagery, using image stabilisation. Such a comparison requires that the imaged scene is static, and that the camera translation is negligible. We do this

Figure 11. Image stabilisation. Left: Uncorrected RS frame. Centre: Rectified frame, with tracked points indicated. Right: Frame stabilised by centring the tracked points along a vertical line.

by tracking two points through the RS frames, using the KLT-tracker [15, 19]. After rolling-shutter compensation, we perform a virtual rotation of the frames (using a global homography), such that two points in the scene are placed symmetrically about the image centre, along a vertical line, see figure 11. The only manual input to this approach is that the two points are indicated manually in the first frame.

We supply two such stabilised sequences as supplemental material (one for the iPhone 3GS and one from the W890i), together with the corresponding uncorrected RS sequences, and results for the GA and GS methods. A single frame comparison of the rectification step is also shown in figure 1, for the iPhone 3GS.

6. Concluding remarks

In this paper, 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 [8].

The method we currently see as the best one is the rotation only model. In addition to being the overall best method, it is also the fastest of our models. Note that even this model 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. We will also investigate better initialisations for the full model. 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.

In general, the quality of the reconstruction should benefit from more measurements. In MIS systems, camera rotations are measured by MEMS gyro sensors [4]. It would be interesting to see how such measurements could be combined with measurements from KLT-tracking when rectifying video. There are also accelerometers in many cellphones, and measurements from these could also be useful in ego-motion estimation.

References

[1] O. Ait-Aider, A. Bartoli, and N. Andreff. Kinematics from lines in a single rolling shutter image. In *CVPR'07*, Minneapolis, USA, June 2007. 36

[2] O. Ait-Aider and F. Berry. Structure and kinematics triangulation with a rolling shutter stereo rig. In *IEEE International Conference on Computer Vision*, 2009. 36

[3] S. Baker, D. Scharstein, J. P. Lewis, S. Roth, M. J. Black, and R. Szeliski. A database and evaluation methodology for optical flow. In *IEEE ICCV*, 2007. 36, 39

[4] J. Bernstein. An overview of MEMS inertial sensing technology. *Sensors Magazine*, 2003(1), February 2003. 42

[5] L.-W. Chang, C.-K. Liang, and H. Chen. Analysis and compensation of rolling shutter distortion for CMOS image sensor arrays. In *ISCOM'05*, 2005. 36

[6] W.-H. Cho and K.-S. Hong. Affine motion based CMOS distortion analysis and CMOS digital image stabilization. *IEEE TCE*, 53(3):833–841, August 2007. 36, 41

[7] W.-H. Cho, D.-W. Kim, and K.-S. Hong. CMOS digital image stabilization. *IEEE TCE*, 53(3):979–986, 2007. 36

[8] J.-B. Chun, H. Jung, and C.-M. Kyung. Suppressing rolling-shutter distortion of CMOS image sensors by motion vector detection. *IEEE TCE*, 54(4):1479–1487, 2008. 36, 41, 42

[9] A. E. Gamal and H. Eltoukhy. CMOS image sensors. *IEEE Circuits and Devices Magazine*, May/June 2005. 35

[10] C. Geyer, M. Meingast, and S. Sastry. Geometric models of rolling-shutter cameras. In *6th OmniVis WS*, 2005. 36, 37

[11] C. G. Harris and M. Stephens. A combined corner and edge detector. In *4th Alvey Vision Conference*, pages 147–151, September 1988. 39

[12] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000. 36

[13] C.-K. Liang, L.-W. Chang, and H. Chen. Analysis and compensation of rolling shutter effect. *IEEE Transactions on Image Processing*, 17(8):1323–1330, August 2008. 36

[14] F. Liu, M. Gleicher, H. Jin, and A. Agarwala. Content-preserving warps for 3D video stabilization. In *ACM Transactions on Graphics*, 2009. 36

[15] B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *IJCAI81*, pages 674–679, 1981. 39, 42

[16] S. P. Nicklin, R. D. Fisher, and R. H. Middleton. Rolling shutter image compensation. In *Robocup 2006 LNAI 4434*, pages 402–409, 2007. 36

[17] F. Park and B. Ravani. Smooth invariant interpolation of rotations. *ACM Transactions on Graphics*, 16(3):277–295, July 1997. 38

[18] E. Ringaby. Rolling shutter dataset with ground truth. `http://www.cvl.isy.liu.se/research/rs-dataset`. 36, 39

[19] J. Shi and C. Tomasi. Good features to track. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR'94*, Seattle, June 1994. 39, 42

[20] K. Shoemake. Animating rotation with quaternion curves. In *Int. Conf. on CGIT*, pages 245–254, 1985. 37, 38

[21] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000. 36, 37