BUNDLE ADJUSTMENT RULES

Chris Engels, Henrik Stewénius, David Nistér

Center for Visualization and Virtual Environments, Department of Computer Science,
University of Kentucky
{engels@vis, stewe@vis, dnister@cs}.uky.edu
http://www.vis.uky.edu

KEY WORDS: Bundle Adjustment, Structure from Motion, Camera Tracking

ABSTRACT:

In this paper we investigate the status of bundle adjustment as a component of a real-time camera tracking system and show that with current computing hardware a significant amount of bundle adjustment can be performed every time a new frame is added, even under stringent real-time constraints. We also show, by quantifying the failure rate over long video sequences, that the bundle adjustment is able to significantly decrease the rate of gross failures in the camera tracking. Thus, bundle adjustment does not only bring accuracy improvements. The accuracy improvements also suppress error buildup in a way that is crucial for the performance of the camera tracker. Our experimental study is performed in the setting of tracking the trajectory a calibrated camera moving in 3D for various types of motion, showing that bundle adjustment should be considered an important component for a state-of-the-art real-time camera tracking system.

1 INTRODUCTION

Bundle adjustment is the method of choice for many photogrammetry applications. It has also come to take a prominent role in computer vision applications geared towards 3D reconstruction and structure from motion. In this paper we present an experimental study of bundle adjustment for the purpose of tracking the trajectory of a calibrated camera moving in 3D. The main purposes of this paper are

- To investigate experimentally the fact that bundle adjustment does not only increase the accuracy of the camera trajectory, but also prevents error-buildup in a way that decreases the frequency of total failure of the camera tracking.
- To show that with the current computing power in standard computing platforms, efficient implementations of bundle adjustment now provide a very viable option even for real-time applications, meaning that bundle adjustment should be considered the gold standard for even the most demanding real-time computer vision applications.

The first item, to show that bundle adjustment can in fact make the difference between total failure and success of a camera tracker, is interesting because the merits of bundle adjustment are more often considered based on the accuracy improvements it provides to an estimate that is already approximately correct. This is naturally the case since bundle adjustment requires an approximate (as good as possible) initialization, and will typically not save a really poor initialization. However, several researchers have noted (Fitzgibbon and Zisserman, 1998, Nistér, 2001, Pollefeys, 1999) that in the application of camera tracking, performing bundle adjustment each time a new frame has been added to the estimation can prevent the tracking process from failing over time. Thus, bundle adjustment can over time in a sequential estimation process have a much more dramatic impact than mere accuracy improvement, since it improves the initialization for future estimates, which can ultimately enable success in cases that would otherwise miserably fail. To our knowledge, previous authors have mainly provided anecdotal evidence of this fact, and one of

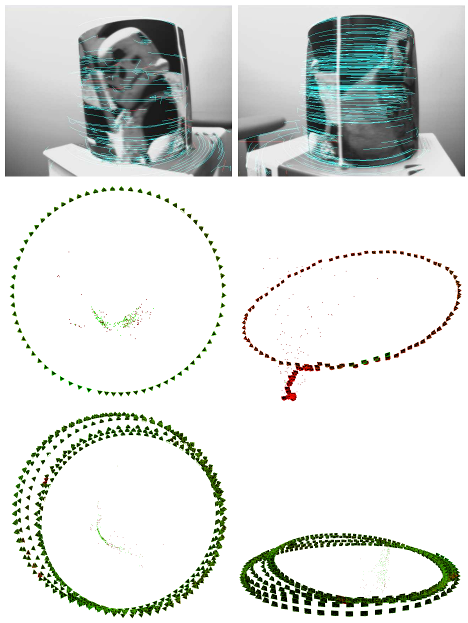

Figure 1: Top: Feature tracking on a 'turntable' sequence created by rotating a cylinder sitting on a rotating chair. Middle Left: When bundle adjusting the 20 most recent views with 20 iterations every time a view is added, the whole estimation still runs at several frames a second, and produces a nice circular camera trajectory, Middle Right: Without bundle adjustment, the estimation is more irregular, but perhaps more importantly, somewhat prone to gross failure. Here we show an example of the type of failure prevented by bundle adjustment. Bottom Left and Right: Although there is some drift, the bundle adjusted estimation is much more reliable and relatively long term stable. Here two views of multiple laps are shown. All the laps were estimated as full 6 degree of freedom unsmoothed motion and without attempting to establish correspondences to previous laps.

This work was supported in part by the National Science Foundation under award number IIS-0545920, Faculty Early Career Development (CAREER) Program.

our main contributions is to quantify the impact of bundle adjustment on the failure rate of camera tracking. In particular, we investigate the impact on the failure rate of n iterations of bundle adjustment over the last m video frames each time a frame is added, for various values of n and m.

The second item, to show that bundle adjustment can now be considered in real-time applications, is partially motivated by the fact that bundle adjustment is often dismissed as a batch-only method, often when introducing another ad-hoc method for structure from motion. Some of the 're-invention' and 'home-brewing' of ad-hoc methods for structure from motion have been avoided by rigorous and systematic exposition of bundle adjustment to the computer vision community, such as for example by (Triggs et al., 2000), but it is still an occurring phenomenon. Several researchers have previously developed systems that can perform real-time structure from motion without bundle adjustment, see e.g. (Davison and Murray, 2002, Nistér et al., 2006).

Admittedly, it is typically not possible in real-time applications to incorporate information from video frames further along in the sequence, as this would cause unacceptable latency. However, bundle adjustment does not necessarily require information from future video frames. In fact, bundle adjustment of as many frames backwards as possible each time a frame is added, will provide the best accuracy possible using only information up to the current time. If such bundle adjustment can be performed within the time-constraints of the application at hand, there is really no good excuse for not using it. We investigate the computation time required by an efficient implementation of bundle adjustment geared specifically at real-time camera tracking when various amounts of frames are included in the bundle adjustment. We then combine the failure rate experiments with our timing experiments to provide information on how much the failure rate can be decreased given various amounts of computation time, showing that bundle adjustment is an important component of a state-of-the-art real-time camera tracking system.

2 THEORETICAL BACKGROUND AND IMPLEMENTATION

In this section we describe the bundle adjustment process and the details of our implementation. For readers familiar with the details of numerical optimization and bundle adjustment, the main purpose of this section is simply to avoid any confusion regarding the exact implementation of the bundle adjuster used in the experiments. For readers who are less familiar with this material, this section also gives an introduction to bundle adjustment, which seems appropriate given that this paper argues for bundle adjustment.

A very large class of minimization schemes try to minimize a cost function $c(x)$ iteratively by approximating the cost function locally around the current (M-dimensional) position x with a quadratic Taylor expansion

$$c(x+dx) \approx c(x) + \nabla c(x)^\top dx + \frac{1}{2} dx^\top H_c(x) dx \qquad (1)$$

where $\nabla c(x)$ is the gradient

$$\nabla c(x) = \begin{bmatrix} \frac{\partial c}{\partial x_1}(x) & \dots & \frac{\partial c}{\partial x_M}(x) \end{bmatrix}^\top \qquad (2)$$

of c at x and $H_c(x)$ is the Hessian

$$H_c(x) = \begin{bmatrix} \frac{\partial^2 c}{\partial x_1 \partial x_1}(x) & \dots & \frac{\partial^2 c}{\partial x_1 \partial x_M}(x) \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 c}{\partial x_N \partial x_1}(x) & \dots & \frac{\partial^2 c}{\partial x_N \partial x_M}(x) \end{bmatrix} \qquad (3)$$

of c at x. By taking the derivative of (1) and equating to zero, one obtains

$$H_c(x) dx = -\nabla c(x), \qquad (4)$$

which is a linear equation for the update vector dx. Since there is no guarantee that the quadratic approximation will lead to an update dx that improves the cost function, it is very common to augment the update so that it goes towards small steps down the gradient when improvement fails. There are many ways to do this since any method that varies between the update defined by (4) and smaller and smaller steps down the gradient will suffice in principle. For example, one can add some scalar λ to all the diagonal elements of $H_c(x)$. When improvement succeeds, we decrease λ towards zero, since at $\lambda = 0$ we get the step defined by the quadratic approximation, which will ultimately lead to fast convergence near the minimum. When improvement fails, we increase λ, which makes the update tend towards

$$dx = -\frac{1}{\lambda} \nabla c(x), \qquad (5)$$

which guarantees that improvement will be found for sufficiently large λ (barring numerical problems).

Typically, the cost function is the square sum of all the dimensions of an (N-dimensional) error vector function $f(x)$:

$$c(x) = f(x)^\top f(x). \qquad (6)$$

Note that the error vector f can be defined in such a way that the square sum represents a robust cost function, rather than just an outlier-sensitive plain least squares cost function.

We use, as is very common, the so-called Gauss-Newton approximation of the Hessian, which comes from approximating the vector function $f(x)$ around x with the first order Taylor expansion

$$f(x+dx) \approx f(x) + J_f(x) dx, \qquad (7)$$

where $J_f(x)$ is the Jacobian

$$J_f(x) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(x) & \dots & \frac{\partial f_1}{\partial x_M}(x) \\ \vdots & \vdots & \vdots \\ \frac{\partial f_N}{\partial x_1}(x) & \dots & \frac{\partial f_N}{\partial x_M}(x) \end{bmatrix} \qquad (8)$$

of f at x. Inserting (7) into (6), we get

$$c(x+dx) \approx f^\top f(x) + 2 f^\top J_f(x) dx + dx^\top J_f^\top J_f(x) dx, \qquad (9)$$

which by equating the derivative to zero results in the update equation

$$J_f(x)^\top J_f(x) dx = -J_f(x)^\top f(x). \qquad (10)$$

By noting that $2J_f(x)^\top f(x)$ is the exact gradient of (6) and comparing with (4) one can see that the Hessian has been approximated by

$$H_c(x) \approx 2 J_f(x)^\top J_f(x). \qquad (11)$$

The great advantage of this is that computation of second derivatives is not necessary. Another benefit is that this Hessian approximation (and its inverse) is normally positive definite (unless the Jacobian has a nullvector), that is

$$dx^\top J_f(x)^\top J_f(x) dx > 0 \quad \forall dx \neq 0, \qquad (12)$$

which opens up more ways of accomplishing the transition towards small steps that guarantee improvement in the cost function. For example, instead of adding λ to the diagonal, we can multiply the diagonal of $J_f(x)^\top J_f(x)$ by the scalar $(1+\lambda)$, which leads to the Levenberg-Marquardt algorithm. This is guaranteed to eventually find an improvement, because an update dx with a sufficiently small magnitude and a negative scalar product with the gradient is guaranteed to do so, and when λ increases, the update tends towards

$$dx = -\frac{1}{\lambda} diag(J_f(x)^\top J_f(x))^{-1} J_f(x)^\top f(x), \quad (13)$$

(where $diag(.)$ stands for the diagonal of a matrix), which is minus the gradient times a small positive diagonal matrix. Yet another update strategy that guarantees improvement, but without solving the linear system for each new value λ, is to upon failure divide the update step by λ, resulting in a step that tends towards

$$dx = -\frac{1}{\lambda} (J_f(x)^\top J_f(x))^{-1} J_f(x)^\top f(x), \quad (14)$$

which is minus the gradient times a small positive definite matrix. With this strategy, only the cost function needs to be reevaluated when λ is increased upon failure to improve, which can be an advantage if the cost function is cheap to evaluate, but the linear system expensive to solve. In our implementation, we use the Levenberg-Marquardt variant.

The core feature of a bundle adjuster (compared to standard numerical optimization) is to take advantage of the so-called primary structure (sparsity), which arises because the parameters for scene features (in our case 3D points) and sensors combine to predict the measurements, while the scene feature parameters do not combine directly and the sensor parameters do not combine directly. More precisely, the error vector f consists of some reprojection error (some difference measure between the predicted and the measured reprojections), which can be made robust by applying a nonlinear mapping that decreases large errors, and the Jacobian J_f has the structure

$$J_f = \begin{bmatrix} J_P & J_C \end{bmatrix}, \quad (15)$$

where J_P is the Jacobian of the error vector f with respect to the 3D point parameters and J_C is the Jacobian of the error vector f with respect to the camera parameters. This results in the Hessian approximation

$$H = \begin{bmatrix} J_P^\top J_P & J_P^\top J_C \\ J_C^\top J_P & J_C^\top J_C \end{bmatrix}, \quad (16)$$

which in the linear system may possibly have an augmented diagonal. The whole linear equation system becomes

$$\begin{bmatrix} H_{PP} & H_{PC} \\ H_{PC}^\top & H_{CC} \end{bmatrix} \begin{bmatrix} dP \\ dC \end{bmatrix} = \begin{bmatrix} b_P \\ b_C \end{bmatrix}, \quad (17)$$

where we have defined $H_{PP} = J_P^\top J_P$, $H_{PC} = J_P^\top J_C$, $H_{CC} = J_C^\top J_C$, $b_P = -J_P^\top f$,$b_C = -J_C^\top f$ to simpify the notation, and dP and dC represent the update of the point parameters and the camera parameters, respectively. Note that the matrices H_{PP} and H_{CC} are block-diagonal, where the blocks correspond to points and cameras, respectively. In order to take advantage of this block-structure, a block-wise Gaussian elimination is now applied to (17). First we multiply by

$$\begin{bmatrix} H_{PP}^{-1} & 0 \\ 0 & I \end{bmatrix} \quad (18)$$

from the left on both sides in order to get the upper left block to identity, resulting in

$$\begin{bmatrix} I & H_{PP}^{-1} H_{PC} \\ H_{PC}^\top & H_{CC} \end{bmatrix} \begin{bmatrix} dP \\ dC \end{bmatrix} = \begin{bmatrix} H_{PP}^{-1} b_P \\ b_C \end{bmatrix}, \quad (19)$$

Then we subtract $H_{PC}^\top$ times the first row from the second row in order to eliminate the lower left block. This can also be thought of as multiplying by

$$\begin{bmatrix} I & 0 \\ -H_{PC}^\top & I \end{bmatrix} \quad (20)$$

from the left on both sides, resulting in the smaller equation system (from the lower part)

$$\underbrace{(H_{CC} - H_{PC}^\top H_{PP}^{-1} H_{PC})}_{A} dC = \underbrace{b_C - H_{PC}^\top H_{PP}^{-1} b_P}_{B} \quad (21)$$

for the camera parameter update dC. For very large systems, the left hand side is still a sparse system due to the fact that not all scene features appear in all sensors. In contrast to the primary structure, this secondary structure depends on the observed tracks, and is hence hard to predict. This makes the sparsity less straightforward to take advantage of. Typical options are to use profile Cholesky factorization with some appropriate on-the-fly variable ordering, or preconditioned conjugate gradient to solve the system. We use straightforward Cholesky factorization. As we shall see, for the size of system resulting from a few tens of cameras, the time required to form the left hand side matrix largely dominates the time necessary to solve the system with straightforward Cholesky factorization. This occurs because the time taken to form the matrix is on the order of $O(N_P l^2)$, where N_P is the number of tracks and l is a representative track length. Since N_P is typically rather large, and l on the order of the number of cameras N_C for smaller systems, this dominates the order of $O(N_C^3)$ time taken to solve the linear system, until N_C starts approaching N_P or largely dominating l.

Once dC has been found, the point parameter updates can be found from the upper part of (19) as

$$dP = H_{PP}^{-1} b_P - H_{PP}^{-1} H_{PC} dC. \quad (22)$$

Since an efficient implementation of the actual computation process corresponding to this description is somewhat involved, we find it helpful to summarize the bundle adjustment process in pseudo-code in Table 1.

The main computation steps that may present bottlenecks are

- The computation of the cost function (which grows linearly in the number of reprojections).
- The computation of derivatives and accumulation over tracks (linear in the number of reprojections).
- The outer product over tracks (which grows with the square of the track lengths times the number of tracks, or thought of another way, approximately the number of reprojections times a representative track length).
- Solving the linear system (which grows with the cube of the number of cameras, unless secondary structure is exploited).
- The back-substitution (linear in the number of reprojections).

Accordingly, these are the computation steps for which we measure timing in the experiments, as well as a total computation time.

We use a calibrated camera model in the experiments, so that the only camera parameters solved for are related to the rotation and translation of the camera. The parameterization used in bundle adjustment is straightforward, with three parameters for translation of each camera, and the sines of three Euler angles parameterizing an incremental rotation from the current position (notice

1 Initialize λ.

2 **Compute cost function** at initial camera and point configuration.

3 Clear the left hand side matrix A and right hand side vector B.

4 For each track p

{

Clear a variable H_{pp} to represent block p of H_{PP} (in our case a symmetric 3×3 matrix) and a variable b_p to represent part p of b_P (in our case a 3-vector).

(Compute derivatives) For each camera c on track p

{

Compute error vector f of reprojection in camera c of point p and its Jacobians J_p and J_c with respect to the point parameters (in our case a 2×3 matrix) and the camera parameters (in our case a 2×6 matrix), respectively.

Add $J_p^\top J_p$ to the upper triangular part of H_{pp}.

Subtract $J_p^\top f$ from b_p.

If camera c is free

{

Add $J_c^\top J_c$ (optionally with an augmented diagonal) to upper triangular part of block (c, c) of left hand side matrix A (in our case a 6×6 matrix).

Compute block (p, c) of H_{PC} as $H_{pc} = J_p^\top J_c$ (in our case a 3×6 matrix) and store it until track is done.

Subtract $J_c^\top f$ from part c of right hand side vector B (related to b_C).

}

}

Augment diagonal of H_{pp}, which is now accumulated and ready. Invert H_{pp}, taking advantage of the fact that it is a symmetric matrix.

Compute $H_{pp}^{-1} b_p$ and store it in a variable t_p.

(Outer product of track) For each free camera c on track p

{

Subtract $H_{pc}^\top t_p = H_{pc}^\top H_{pp}^{-1} b_p$ from part c of right hand side vector B.

Compute the matrix $H_{pc}^\top H_{pp}^{-1}$ and store it in a variable T_{pc}

For each free camera $c2 \geq c$ on track p

{

Subtract $T_{pc} H_{pc2} = H_{pc}^\top H_{pp}^{-1} H_{pc2}$ from block $(c, c2)$ of left hand side matrix A.

}

}

}

5 (Optional) Fix gauge by freezing appropriate coordinates and thereby reducing the linear system with a few dimensions.

6 **(Linear Solving)** Cholesky factor the left hand side matrix B and solve for dC. Add frozen coordinates back in.

7 **(Back-substitution)** For each track p

{

Start with point update for this track $dp = t_p$.

For each camera c on track p

{

Subtract $T_{pc}^\top dc$ from dp (where dc is the update for camera c).

}

Compute updated point.

}

8 **Compute the cost function** for the updated camera and point configuration.

9 If cost function has improved, accept the update step, decrease λ and go to Step 3 (unless converged, in which case quit).

10 Otherwise, increase λ and go to Step 3 (unless exceeded the maximum number of iterations, in which case quit).

Table 1: Pseudo-code showing our implementation.

that the latter has no problems with singularities since the parameterization is updated after each parameter update step, and the rotation updates should never be anywhere close to 90 degrees. For the 3D points, we use the four parameters of a homogeneous coordinate representation, but we always freeze the coordinate with the largest magnitude, the choice of which coordinate to freeze being updated after each parameter update step.

To robustify the reprojection error, we assume that the reprojection errors have a Cauchy-distribution (which is a heavy-tailed distribution), meaning that an image distance of e between the measured and reprojected distance should contribute

$$\ln(1 + \frac{e^2}{\sigma^2}), \tag{23}$$

where σ is a standard deviation, to the cost function (negative log-likelihood). To accomplish this, while still exposing both the horizontal and vertical component of error in the error vector f, the robustifier takes the input error (x, y) in horizontal and vertical direction and outputs the robustified error vector (x_r, y_r) where

$$x_r = \sqrt{\ln(1 + \frac{x^2 + y^2}{\sigma^2})} \frac{x}{\sqrt{x^2 + y^2}} \tag{24}$$

$$y_r = \sqrt{\ln(1 + \frac{x^2 + y^2}{\sigma^2})} \frac{y}{\sqrt{x^2 + y^2}}. \tag{25}$$

The key property of this vector is that the square sum of its component is (23), while balancing the components exactly as the original reprojection error.

3 EXPERIMENTS

We investigate the failure rate of camera tracking with n iterations of bundle adjustment over the last m video frames each time a frame is added, for various values of n and m. The frames beyond the m most recent frames are locked down and not moved in the bundle adjustment. However, the information regarding the uncertainty in reconstructed feature points provided by views that are locked down is still used. That is, reprojection errors are accumulated for the entire feature track lengths backwards in time, regardless of whether the views where the reprojections reside are locked down.

In the beginning of tracking, when the number of frames yet included is less than $m + 2$ so that at most one pose is locked, the gauge is fixed by fixing the first camera pose and the distance between the first and the most current camera position. Otherwise, the gauge fixing is accomplished by the locked views.

It is interesting to note that when we set $m = n = 1$, we get an algorithm that rather closely resembles a simple Kalman filter. We then essentially gather the covariance information induced on the 3D points by all previous views and then update the current pose based on that (using a single iteration), which should be at least as good as a Kalman filter that has a state consisting of independent 3D points, with the potential improvement that the most recent estimates of the 3D point positions are used when computing reprojection errors and their derivatives in previous views.

Note that bundle adjustment as well as Kalman filtering for the application of camera tracking can be used both with or without a camera motion model. We have chosen to concentrate on experiments for the particular case of no camera motion model, i.e. no smoothness on the camera trajectory is imposed, and the only

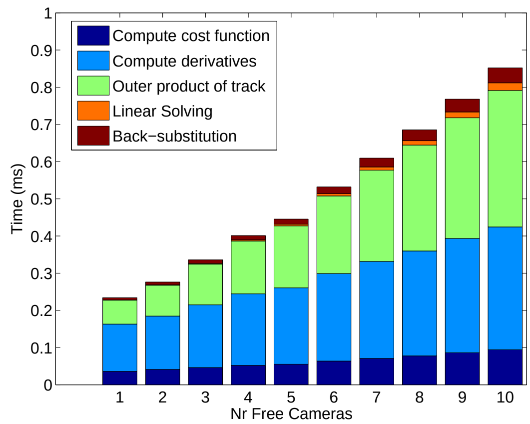

Figure 2: Time per iteration versus the number of free views. Time is dominated by derivative computations and outer products, while linear solving takes negligible time.

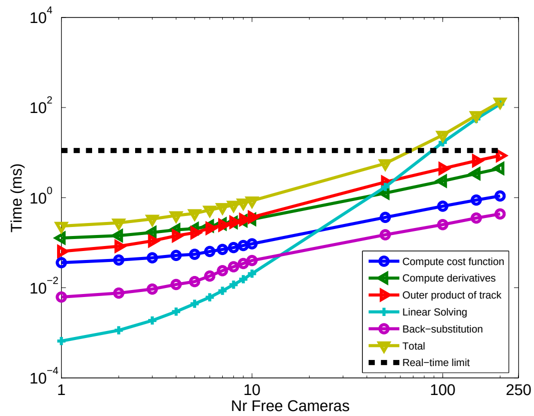

Figure 3: Time per iteration versus the number of free views for larger numbers of free views. Note that the cubic dependence of the computation time in the linear solving on the number of free views eventually makes the linear solving dominate the computation time. The real-time limit is computed as $1s/(30*3)$

constraints on the camera trajectory are the reprojection errors of the reconstruction of tracked feature points. The no motion model case is the most flexible, but also the hardest setting in which to perform estimation. It therefore most clearly elucidates the issue. While a motion model certainly simplifies the estimation task when the motion model holds, it unfortunately makes the 'hard cases even harder', in that when the camera performs unexpected turns or jerky motion, the motion model induces a bias towards status quo motion.

We measure the failure rate by defining a frame-to-frame failure criterion and running long sequences, restarting the estimation from scratch at the current position every time the failure criterion declares that a failure has occurred. Note that this failure criterion is not part of an algorithm, but only a means of measuring the failure rate. For our real-data experiments, we perform simple types of known motion, such as forward, diagonal, sideways or turntable motion, and require that the translation direction and rotation do not deviate more than some upper limit from the known values.

The initialization that occurs in the beginning and after each failure is accomplished using the first three frames and a RANSAC (Fischler and Bolles, 1981) process using the five-point relative

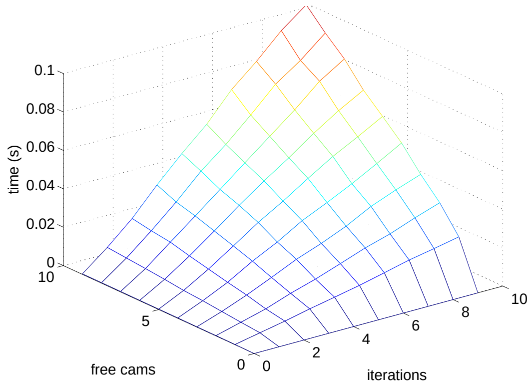

Figure 4: Computation time versus number of free views and number of iterations.

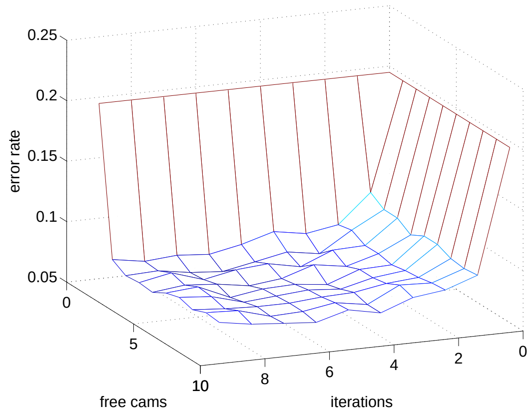

Figure 5: Failure rate as a function of the number of free cameras and the number of iterations.

orientation method in the same manner as described in (Nistér, 2004). In each RANSAC hypothesis, the five-point method provides hypotheses for the first and third view. The five points are triangulated and the second view is computed by a three-point resection (R. Haralick, 1994). The whole three-view initialization is then thoroughly bundle adjusted to provide the best possible initialization.

4 METHOD

For each new single view that is added, the camera position is initialized with a RANSAC process where hypotheses are generated with three-point resections. The points visible in the new view are then re-triangulated using the reprojection in the new view and the first frame where the track was visible. The bundle adjustment with n iterations of the m most recent views is then performed. In both the RANSAC processes and the bundle adjustment the cost function is a robustified reprojection error. Thus, we do not attempt to throw out outlying tracks before the bundle adjustment, but use all tracks and instead employ a robust cost function.

5 RESULTS AND DISCUSSION

Representative computation time measurements are shown in Figures 2, 3 and 4. In Figure 2, the distribution of time over cost function computation, derivatives, outer product, linear solving and back-substitution is shown as a bar-plot for a small number of free views.

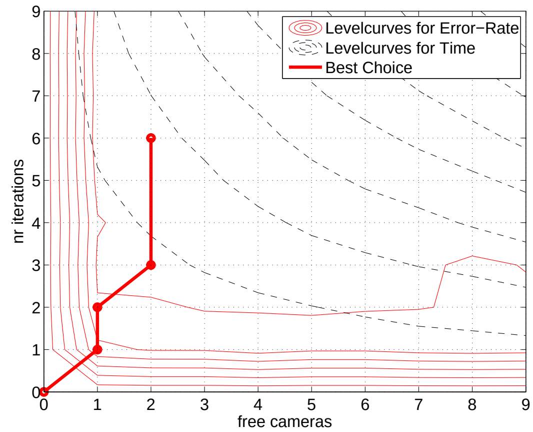

Figure 6: Level curves for time and error rate

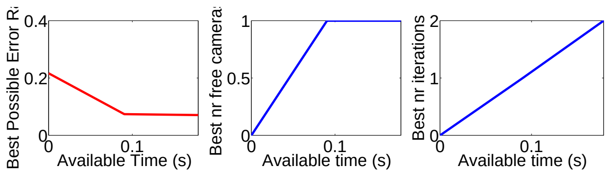

Figure 7: The first plot shows best possible quality for a given computation time. The second and third plots show the m and n values to achieve these results.

The time is per new view and iteration and was measured on the sequence shown in Figure 1, which has 640×480 resolution and 1000 frames in total. The computations were performed on an Alienware with Intel Pentium Xeon processor 3.4GHz, 2.37GB. The average number of tracks present in a frame is 260, and the average track length is 20.5. Note that the computation time is dominated by derivative computation and outer products. Note also that real-time rate can easily be maintained for ten free views. Figure 3 shows how the computation times grow with an increasing number of views. Note that the linear solving, due to the cubic dependence on the number of views eventually becomes the dominant computational burden and that real-time performance is lost somewhere between 50 and 100 free views. Note however that the other computation tasks scale approximately linearly (since the track lengths are limited). When the track length for each feature is constant, increasing the number of feature points results in a linear increase in computation time for all steps in the bundle adjustment except for the linear solver, which is independent of this number.

Some investigations of the failure rate as well as computation time for various amounts of bundle adjustment are shown in Figures 5, 6 and 7. Note that already one iteration on just the most recent view results in a significant decrease of the failure rate, but to get the full benefit of bundle adjustment and 'reach the valley floor', suppressing failures as much as possible, three iterations on three views or perhaps even four iterations on four views is desirable. Note that this does not necessarily mean that additional accuracy can not be gained with more iterations over more views, only that the gross failures have largely been stemmed after that. Also, with bundle adjustment very low failure rates can often be achieved. For example the sequence in Figure 1 can be tracked completely without failures for 1000 frames and over seven laps. Decreases in failure rate at such low failure rates require large amounts of data and computation to measure, but are clearly still very valuable.

In Figure 6 the level curves over the (n, m) space are shown for computation time and failure rate. The 'best path' in the (n, m) space is also shown there and in Figure 7, meaning that for a given amount of computation allowed, the n and m resulting in the lowest failure rate is chosen.

6 CONCLUSIONS AND FUTURE WORK

In this paper, we have investigated experimentally the fact that bundle adjustment does not only increase the accuracy of the camera trajectory, but also prevents error-buildup in a way that decreases the frequency of total failure of the camera tracking. We have also shown that with the current computing power in standard computing platforms, efficient implementations of bundle adjustment now provide a very viable option even for real-time applications.

We have found that bundle adjustment can be performed for a few tens of the most recent views every time a new frame is added, while still maintaining video rate. At such numbers of free views, the most significant components of the computation time are related to the computation of derivatives of the cost function with respect to camera and 3D point parameters, plus the outer products that the feature tracks contribute to the Schur complement arising when eliminating to obtain a linear system for the camera parameter updates. The actual linear solving of the linear system takes negligible time in comparison for a small number of views, but the linear solver, which has a cubic cost in the number of views, eventually becomes the dominating computation when a number of views approaching a hundred are bundle adjusted every time a new frame arrives. This can probably be improved upon by exploiting the secondary structure that still remains in the linear system, which is something we hope to do in future work.

Our results are, as expected, a strong proof that proper bundle adjustment is more efficient that any ad-hoc structure from motion refinement, and the results are well worth the trouble of proper implementation.

REFERENCES

Davison, A. and Murray, D. W., 2002. Simultaneous Localization and Map-Building Using Active Vision. IEEE Transactions on Pattern Analysis and Machine Intelligence 24(7), pp. 865–880.

Fischler, M. A. and Bolles, R. C., 1981. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Communications-of-the-ACM 24(6), pp. 381–95.

Fitzgibbon, A. W. and Zisserman, A., 1998. Automatic camera recovery for closed or open image sequences. In: ECCV (1), pp. 311–326.

Nistér, D., 2001. Automatic Dense Reconstruction from Uncalibrated Video Sequences. PhD thesis, Royal Institute of Technology, Stockholm, Sweden.

Nistér, D., 2004. An efficient solution to the five-point relative pose problem. PAMI 26(6), pp. 756–777.

Nistér, D., Naroditsky, O. and Bergen, J., 2006. Visual odometry for ground vehicle applications. Journal of Field Robotics 23(1), pp. 3–20.

Pollefeys, M., 1999. Self-Calibration and Metric 3D Reconstruction From Uncalibrated Image Sequences. PhD thesis, K.U.Leuven, Belgium.

R. Haralick, C. L. K. Ottenberg, M. N., 1994. Review and analysis of solutions of the three point perspective pose estimation problem. IJCV 13, pp. 331–356.

Triggs, W., McLauchlan, P., Hartley, R. and Fitzgibbon, A., 2000. Bundle adjustment: A modern synthesis. In: W. Triggs, A. Zisserman and R. Szeliski (eds), Vision Algorithms: Theory and Practice, LNCS, Springer Verlag.