Propagation of Innovative Information in Non-Linear Least-Squares Structure from Motion

Drew Steedly Irfan Essa

GVU Center, College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280, USA
E-mail: {steedly|irfan}@cc.gatech.edu

Abstract

We present a new technique that improves upon existing structure from motion (SFM) methods. We propose a SFM algorithm that is both recursive and optimal. Our method incorporates innovative information from new frames into an existing solution without optimizing every camera pose and scene structure parameter. To do this, we incrementally optimize larger subsets of parameters until the error is minimized. These additional parameters are included in the optimization by tracing connections between points and frames. In many cases, the complexity of adding a frame is much smaller than full bundle adjustment of all the parameters. Our algorithm is best described as incremental bundle adjustment as it allows new information to be added to an existing non-linear least-squares solution.

1. Introduction

Structure from motion (SFM) is the process of determining the pose, or motion, of a camera and the structure of a scene from a sequence of camera images. SFM algorithms are useful for numerous applications, including robotic navigation, surveying, image-based-rendering, and model generation.

In this paper, we present a method for incrementally adding new frames to an existing SFM solution while ensuring optimality. There are many cases where adding frames incrementally is beneficial. For example, in robot navigation, decisions about how to move are made continuously, and an updated SFM solution is needed for each decision.

Solving SFM has been approached in a number of ways, from non-linear least-squares bundle adjustment [10, 12] using Levenburg-Marquadt (LM) to recursive methods using the extended Kalman filtering (EKF) [6, 11, 1, 3, 5] to linear methods on a few frames [4, 2, 9]. Bundle adjustment is the process of iteratively adjusting the camera poses and point positions in order to move toward the optimal least squares answer. As with all gradient descent methods, there is the possibility of getting stuck in local minima. If these local minima are avoided, the minimum least-squares error solution will eventually be reached. The main drawback of bundle adjustment is that it is very slow for substantial numbers of frames because it iteratively adjusts all the parameters for each point and frame in the sequence.

Recursive methods process frames one by one and update a portion of the system state with each new frame. The main benefit of the recursive methods is that they run in an approximately constant amount of time per frame. The drawback, though, is that they do not guarantee and, in general, do not generate the optimal least squares solution.

Lastly, linear methods such as fundamental matrix or trifocal tensor estimation do not minimize the least-squares reprojection error for the frames that the solution is computed with. The resulting matrices or tensors must be projected down on to a manifold of valid solutions. Chaining together the suboptimal answers that the linear methods generate has been looked at as a fast SFM algorithm[7, 10, 2], but the resulting solution is not guaranteed to be optimal.

In this paper, we first look at bundle adjustment algorithms in section 2 and discuss what causes them to be so slow. Next, we review EKF methods in section 3 and look at cases where they should give results that are nearly optimal and cases where they will not. Lastly, in section 4 we show how to modify the standard bundle adjustment algorithm to run in a comparable amount of time as EKF methods when they would give accurate results. If an EKF approach will not yield good results, we show how to detect this and still give the optimal answer at the expense of an increased number of computations. In section 5 we present results of our method on both synthetic and real data to validate our approach.

2. Bundle Adjustment

Bundle adjustment is non-linear least-squares minimization of an error function. A commonly used error function is the sum of the squared reprojection error, described mathematically as

$$\chi^2 = \sum_{f \in \mathbf{F}} \sum_{p \in \mathbf{P}_f} \mathbf{E}_{pf}^T \Lambda_{pf} \mathbf{E}_{pf}, \quad (1)$$

$$= \sum_{f \in \mathbf{F}} \sum_{p \in \mathbf{P}_f} \left(\mathbf{x}_{pf} - \mathbf{u}_{pf}\right)^T \Lambda_{pf} \left(\mathbf{x}_{pf} - \mathbf{u}_{pf}\right), \quad (2)$$

where $\mathbf{F}$ the set of all camera frames and $\mathbf{P}_f$ is the set of points visible in frame f. Λ_{pf} is a 2x2 covariance matrix that reflects the confidence of the tracked point location $\mathbf{u}_{pf}$ in frame f. $\mathbf{x}_{pf}$ is the reprojection of the 3D point, $\mathbf{M}_p$, into frame f as given by

$$\mathbf{x}_{pf} = Proj(H_f(T_f(\mathbf{M}_p))), \quad (3)$$

where T_f is the transformation from the reference coordinate frame into the camera space, H_f is a 3x3 matrix representing the camera internal parameters and

$$Proj\left([x, y, z]^T\right) = [x/z, y/z]^T. \quad (4)$$

Points in an image are extracted by feature tracking methods (e.g. [13]) with some or complete manual interaction. The feature noise is approximated as the zero mean, bivariate gaussian, $N(\mathbf{0}, \Lambda_{pf})$. Therefore, if a feature tracker is used, it must include outlier rejection [14]. In Euclidean bundle adjustment, H_f is given by the known internal camera parameters. In projective bundle adjustment, some of the internal camera parameters are unknown, and therefore H_f is also at least partially unknown. In both methods the pose of the frame, T_f, and the structure of the scene, $\mathbf{M}_p$, are unknowns that must be solved for.

For the Euclidean problem, the pose of a frame is represented with six parameters,

$$\mathbf{y}_f = [\omega_{xf}, \omega_{yf}, \omega_{zf}, t_{xf}, t_{yf}, t_{zf}]^T, \quad (5)$$

while, for the projective version where a single focal length needs to be estimated, a typical parameterization is

$$\mathbf{y}_f = [\omega_{xf}, \omega_{yf}, \omega_{zf}, t_{xf}, t_{yf}, t_{zf}, \beta_f]^T, \quad (6)$$

where $\beta_f = 1/f$ [1]. The position of a point is represented with three parameters,

$$\mathbf{M}_p = [M_{xp}, M_{yp}, M_{zp}]^T. \quad (7)$$

For mathematical succinctness, we combine all the unknowns into a vector of length $(6+i)m + 3n$,

$$\mathbf{Y} = [\mathbf{y}_1^T, \mathbf{y}_2^T, ..., \mathbf{y}_m^T, \mathbf{M}_1^T, \mathbf{M}_2^T, ..., \mathbf{M}_n^T]^T, \quad (8)$$

where m is the number of frames to be optimized, i is the number of internal camera parameters and n is the number of points.

2.1. Minimizing the Objective Function

The minimization of this objective function has typically been done using Levenberg-Marquadt (LM) [12, 8]. At each iteration, the gradient and the second gradient (ie. the Hessian) of the error function are computed and the following linear system is solved for the parameter step, δ:

$$\mathbf{A}\delta = \nabla\chi^2 \quad (9)$$

The gradient is given by

$$\nabla\chi^2 = 2 \sum_{f \in \mathbf{F}} \sum_{p \in \mathbf{P}_f} \frac{\partial \mathbf{x}_{pf}}{\partial \mathbf{Y}}^T \Lambda_{pf} \mathbf{E}_{pf} \quad (10)$$

and the Hessian is approximated by

$$\mathbf{A} \approx 2 \sum_{f \in \mathbf{F}} \sum_{p \in \mathbf{P}_f} \frac{\partial \mathbf{x}_{pf}}{\partial \mathbf{Y}}^T \Lambda_{pf} \frac{\partial \mathbf{x}_{pf}}{\partial \mathbf{Y}}. \quad (11)$$

The estimate of $\mathbf{Y}$ is updated at each iteration as $\mathbf{Y}_{n+1} = \mathbf{Y}_n + \delta_n$. An alternative to the method just described, simultaneous bundle adjustment, is interleaved bundle adjustment [4]. Interleaved bundle adjustment alternates between optimizing the points with the frames held constant and optimizing the frames with the points held constant. In this manner, each point optimization is independent of all other point optimizations and each frame optimization is independent of all other frame optimizations. Therefore, many small problems are solved in the place of one much larger one. Each iteration can be computed faster for interleaved than for simultaneous bundle adjustment, but more iterations are required.

2.2. Redundancies in Full Bundle Adjustment

So, what makes bundle adjustment slow? As the number of parameters increases, the size of the linear system in (9) increases. The computational cost of solving (9) becomes prohibitive as the size of $\mathbf{Y}$ gets large. However, there are scenarios where the problem can be solved less expensively. Consider the case where all the points are tracked in all of the frames. As more and more frames are added, the confidence of the point positions becomes higher and higher. Eventually, each new frame only increases the confidence of the point positions by a very small increment. Therefore, not much accuracy would be sacrificed by eventually switching to pure pose estimation. Once this confidence threshold was reached, the cost of adding a new frame would be constant. This redundancy in frame information has been exploited in [10] by using two "virtual key frames" to represent a set of frames.

Next, consider the case where the camera moves at a fairly constant velocity down a path. Point features are acquired, tracked for a number of frames and then lost. As the

camera gets a long distance down the path, the new frames have a smaller and smaller impact on the position of the first frame and points. This is because the latter frames are more and more decoupled from the first frames. One way to take advantage of this is to fix the state of all the parameters that are a long distance down the path and are only minimally coupled to the new frame. By only optimizing a window of frames around the new frame, the size, and therefore computational cost, of solving (9) decreases. As this window gets larger, the answer gets closer to that of full bundle adjustment. The accuracy gained by increasing the size of the moving window past some point should become small and not be worth the extra computational cost of increasing it.

3. Recursive Methods

Kalman filtering and other recursive techniques such as the Variable State Dimension Filter (VSDF) [6] make first order assumptions about the non-linear error function as a way to speed up processing. In this section, we look at where these approximations are valid and cases where they are not.

The EKF has been implemented in a couple of different ways [11, 1, 3, 5]. EKF methods typically have a state vector that consists of the parameters for the new frame and the parameters for the points that have been tracked in the new frame. In general, EKF methods are similar to taking a single LM step, updating only the parameters of the state vector and simultaneously enforcing a motion model for the camera.

The state vector used by [1] contains parameters for the position of the current frame and the points that were tracked in that frame. The point parameterization was reduced to one dimension per point by constraining it to lie along a ray from one of the cameras. This helped make the problem better conditioned, and therefore made the EKF implementation more stable. Another way to improve the stability of the EKF is to use the iterated EKF instead. If an EKF implementation can be approximated as taking a LM step on the parameters in the state vector, the the iterated EKF (iEKF) continues stepping until those parameters converge.

Even if the iterated EKF does not go unstable, it does not update parameters that are not in the current state vector even though LM would move them during reminimization. One way to correct this is to add in more parameters. [6] does just that by using a state vector that contains all the parameters for the past k frames and the points that are projected in those k frames. If most of the parameters that have to change are contained in this state vector, then the answer the VSDF gives is close to the answer that LM yields. The issue that is still left, though, is how to choose a window size. [6] suggests adaptively choosing the value of k, but does not give a method for doing so.

The main similarities between our method and these recursive methods are that we integrate one frame at a time into our solution and that we can sometimes optimize just a subset of the parameters. As we will discuss in section 4, we dynamically update the subset of parameters in our effective state vector. Our method will always optimize the same or smaller number of parameters than the VSDF would have to include in its state vector to yield the same answer as full bundle adjustment.

Recursive methods usually perform well on cases similar to the ones described in section 2.2. The amount of innovative information provided by each new frame is relatively small, which is the property that the recursive methods exploit. On the other hand, there are many relatively common camera motions that supply large amounts of innovative information in a small number of frames. These motions require that many parameters in the system are changed significantly to accommodate the large innovation.

Consider the case of a camera path that travels around a city block and ends up back where it started. Recursive methods can potentially remain stable all the way around the block. The frame to frame baseline is small so the innovation is also small. Invariably, though, there is error that accumulates as the camera moves around the block. The estimated position of the camera once it has made the circuit can be a significant distance from the estimated position of the initial frame. Once the points from the initial frame are re-acquired, all of the accumulated error must be reconciled in the span of a few frames. Most recursive methods will fail when enough error accumulates so that they cannot reconcile that error within a few frames.

Bundle adjustment algorithms, if given a good enough initialization, can converge to the correct solution. Our proposed method will handle this case only if it does not get stuck in a local minimum. If the error accumulation is large enough, the probability of getting stuck is also high. Our method could potentially be augmented to detect local minimums by looking for large reprojection errors. Once detected, it may be possible to converge to the global minimum. Regardless, the computational cost of our system will spike up when forced to reconcile a large amount of innovative information.

4. Incorporating Innovation

In this section, we detail our technique for choosing the subset of parameters to be adjusted when we add a new frame. We start with an optimized set of point and frame positions for all parameters up to frame $n-1$. Mathematically, this means that

$$\nabla \chi^2 (n-1) = \mathbf{0}, \tag{12}$$

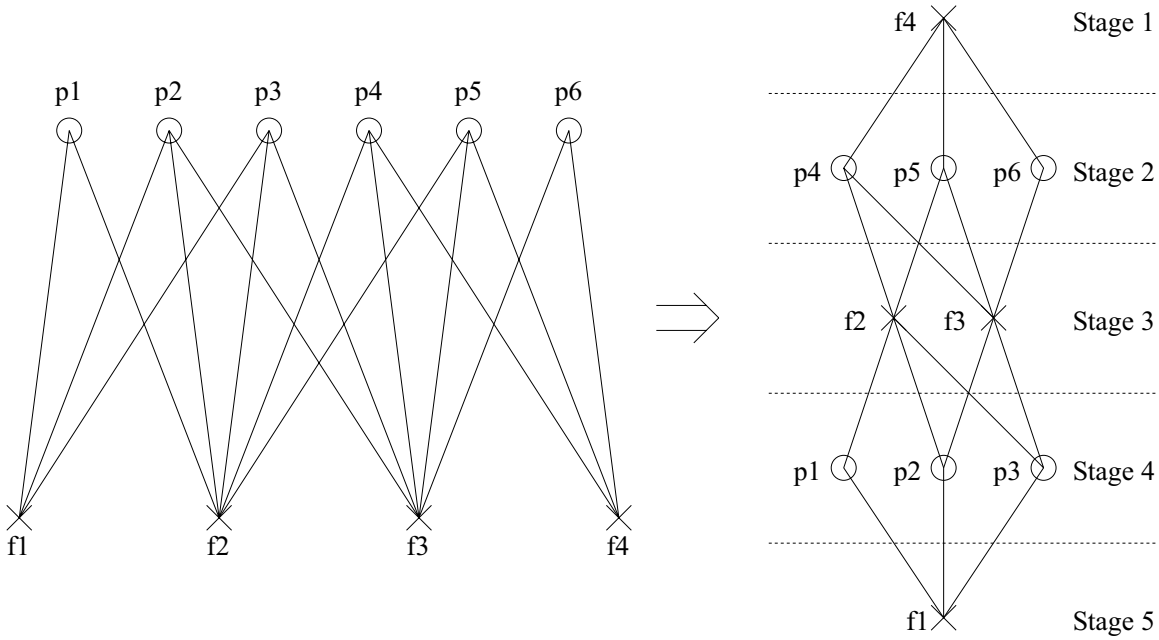


Figure 1. *A simple example used to illustrate our algorithm (left) The bottom crosses represent frames $f_1 \ldots f_4$, the top circles represent points $p_1 \ldots p_6$ and the lines between them represent projections of points into frames. Note that only a few points are visible from each frame. (right) This is an unraveled version of same structure as on the left with same connections, but showing the different stages of processing as $f4$ is added. Innovative information from a new frame is incorporated into the rest of the system by propagating down to stage 5 from stage 1.*

where

$$\chi^2(n-1) = \sum_{f=1}^{n-1} \sum_{p \in \mathbf{P}_f} \mathbf{E}_{pf}^T \Lambda_{pf} \mathbf{E}_{pf} \qquad (13)$$

The goal is to minimize $\nabla \chi^2(n)$ by adjusting only the parameters in $\mathbf{Y}$ that have to change. We will also illustrate our algorithm using a simple example in Figure 1 which shows four frames, $f1, \ldots, f4$ with six points $p1, \ldots, p6$.

Before each frame is added, the entire interconnected point-frame system is at its minimum energy state. One way to help visualize how the system changes when a frame is added is to make a spring-mass system analogy. The points and frames are masses and the connections between them (projections of points onto the frames) are springs. The starting state is where the system is at rest. Adding a frame is like adding a frame node and connecting it to a set of point nodes. This new frame node introduces a new force onto each of the point nodes it is connected to. If all of the nodes are allowed to settle to a new rest state, the force applied by the new frame will have been dissipated throughout the system by traveling through each of the spring connections. The path that the force is dissipated through is illustrated in Figure 1(right). The error propagates from the first stage at the top of the diagram down toward the bottom.

In full bundle adjustment, a linearized step would be calculated for the entire system and all the parameters would be iteratively adjusted until a new minimum, or rest state, is reached. It is possible, though, that the new frame only provides a very small amount of innovative information about the system. The final state of the system might be very close to the initial state. For example, imagine we take a system that is at its minimum energy state and perturb a frame slightly without changing all the other points and frames. The quickest way to get back to the minimum energy state is to only optimize the frame we perturbed. This would be equivalent to nailing down all the nodes other than the ones that were perturbed and letting the ones we perturbed settle back to their original rest state. However, full bundle adjustment would not do this. Instead, the error of the perturbed parameters will generate an update step for every other parameter. Therefore, many points and frames other than the ones we perturbed will end up being involved in a purely transient oscillation.

We try to avoid wasting computations on these transient oscillations by allowing as few parameters as possible to be adjusted. We initially only optimize the parameters that have introduced the innovation, the pose of the new frame. This corresponds to optimizing stage 1 of Figure 1. In our

spring-mass analogy, this would be equivalent to locking down all the nodes in the system, attaching the new frame node in stage 1 to the point nodes in stage 2 via springs, and letting the new node settle to a stable state. If, after the new node has settled, the stage 2 nodes we connected it to are not being pulled away from their current state, we could release all of the nodes and they would not move. If, on the other hand, some of them are being pulled, we would unlock only those nodes and the new node and let them settle to a stable state. We keep releasing nodes in subsequent stages until we eventually reach a stage that does not need to readjust. Once that stage is reached, the entire system has reached the new rest state.

In order to express this process mathematically, we separate the error function into sets of terms corresponding to the connections, or springs, between the stages of Figure 1.

$$\chi^2 = \chi^2_{1,2} + \ldots + \chi^2_{(m-1),m}, \tag{14}$$

$\chi^2_{i,j}$ is the set of terms corresponding to the connections between stages i and j, or,

$$\chi^2_{i,j} = \sum_{f \in \mathbf{Y}_i} \sum_{p \in \{\mathbf{Y}_j \cap \mathbf{P}_f\}} \mathbf{E}^T_{pf} \Lambda_{pf} \mathbf{E}_{pf} \tag{15}$$

where $\mathbf{Y}_i$ is the subset of parameters of $\mathbf{Y}$ that are in stage i.

We start by optimizing $\chi^2_{1,2}$ with respect to the parameters in stage 1, $\mathbf{Y}_1$. Just as the Newton-Raphson method fits a quadratic to the error surface and steps to where the predicted minimum is, we can also predict the percentage error reduction of that step. When the predicted percentage error reduction falls below some threshold, we stop optimizing. Once the minimization of $\chi^2_{1,2}$ is done, we use this same stopping criteria to evaluate whether the parameters in $\mathbf{Y}_2$ need to move. If not, then the minimization of χ^2 is done. If they do, then $\chi^2_{1,2}+\chi^2_{2,3}$ needs to be minimized with respect to $\left[\mathbf{Y}_1^T, \mathbf{Y}_2^T\right]^T$.

The set of error terms being minimized after including s stages is

$$\sum_{i=1}^{s} \chi^2_{i,i+1} \tag{16}$$

and it is being minimized with respect to $\left[\mathbf{Y}_1^T \ldots \mathbf{Y}_s^T\right]^T$.

The process just described will work for both simultaneous and interleaved optimizations. If the interleaved version is being used, a further parameter set reduction is possible. Since each point and frame is updated independently, not all of the points or frames in a stage may change. If some parameters of Y_i do not need to be adjusted, they are left out of the optimization and their connections to the next stage do not need to be checked. For example, if we have just finished optimizing stages 1 and 2 of Figure 1 we may find that $f2$ needs to be adjusted but $f3$ does not. We can leave $f3$ out of the next optimization and only check the connections from points $p2$ and $p3$ to $f3$. If we end up moving $p2$ or $p3$, then $f3$ needs to be checked along with $f1$, so it is possible that $f3$ could still be brought into the optimization.

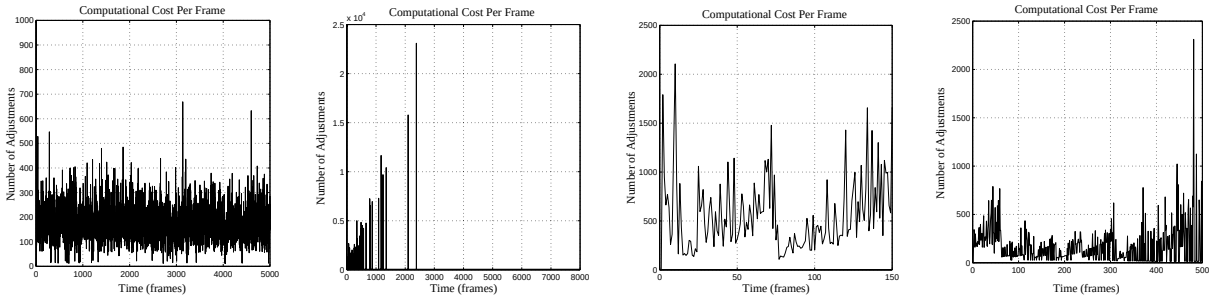
5. Results

Here we present the results of running an interleaved version of our algorithm on both synthetic and real video sequences. Each adjustment or stopping criterion evaluation of a frame or point requires a single matrix inversion. For point adjustments, the matrices are 3x3 and for frame adjustments they are 6x6. To evaluate the computational cost of each sequence, we count the total number of matrix inversions required. The computational cost of processing each sequence is plotted in Figures 2 and 3.

The first synthetic sequence was generated to model the case of a camera moving with constant velocity as described in Section 2.2. A sequence of 5000 images was generated by moving the camera with a constant velocity parallel to the image plane. Point features were uniformly distributed along the path of the camera so that each image contained approximately 20 projections and each point feature was tracked in approximately 20 images. The synthetic camera had a resolution of 720x480, a pixel aspect ratio of 0.91 and a 45 degree horizontal field of view. Gaussian noise was added to the point projections. Figures 2(a) and 3(a) show that, as expected, the computational cost per frame was roughly constant. On a 600MHz PC, the entire sequence took less than 20 minutes, or about 4-5 frames per second.

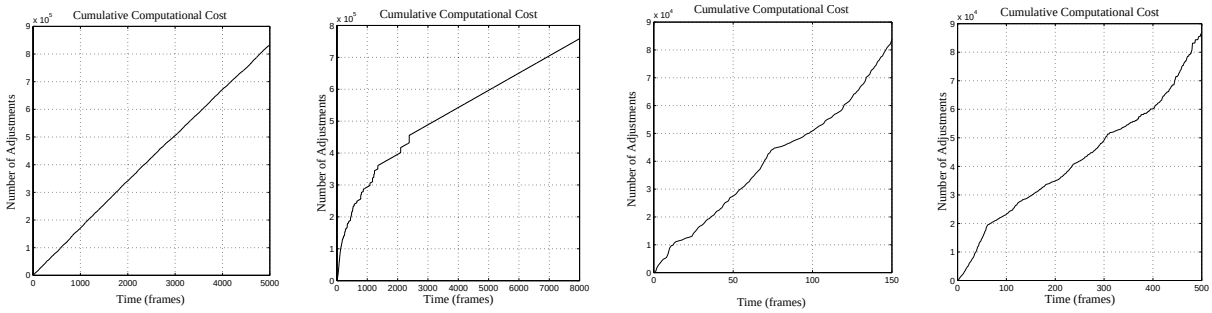
In the second synthetic sequence, we rotated a camera around a set of points so that all points were seen in all cameras. This was meant to demonstrate that, after enough cameras have been used to estimate the point positions, the remaining cameras do not contribute a significant amount of information about the point positions. We placed 50 points randomly on the surface of a sphere and rotated a camera around it 32 times, taking 250 images per revolution for a total of 8000 images. the radius of the sphere was f and the camera center of projection was kept a distance $4f$ from the center of the sphere. As the camera travelled around the sphere, it was rotated so that the sphere remained centered in the image. The internal camera parameters were kept the same as for the first synthetic sequence and Gaussian noise was added to the point projections.

After a few revolutions, the points only needed to be adjusted infrequently and most of the frames were being added in using purely pose estimation. The spikes in Figure 2(b) represent cases where some set of points no longer met the stopping criteria and were moved. Each time a set of points moves, all the frames that have been added must be checked against the stopping criteria. The last 22 revolutions (5500



(a) *(b)* *(c)* *(d)*

Figure 2. *Number of frame and point adjustments required to add each frame (a) linear motion synthetic sequence (b) circular motion synthetic sequence (c) flower garden sequence (d) lab sequence.*



(a) *(b)* *(c)* *(d)*

Figure 3. *Cumulative number of frame and point adjustments (integration of Figure 2) (a) linear motion synthetic sequence (b) circular motion synthetic sequence (c) flower garden sequence (d) lab sequence.*

images) were successfully added using only pose estimation, which only requires a few adjustments for the frame being added and 50 3x3 matrix inversions to check the points against the stopping criteria.

Our two sequences of real video footage contain approximately linear camera motion similar to that of the first synthetic sequence. The first video is the familiar flower garden sequence (Figure 4) [150 frames]. Thirty-eight point features were selected by hand and then tracked automatically through the sequence. The second video is footage we captured in a lab (Figure 5) [500 frames]. In the lab sequence, 54 point features were again selected by hand and then tracked automatically through the sequence. Some of the distant features were visible throughout most of each sequence. The closer features were acquired, tracked for a while and then lost.

In both real sequences, the computational cost is more erratic than in the linear motion synthetic sequence. In both sequences, a startup phase of increased computation is evident where the positions of points are being estimated from their initial positions. Eventually, the baseline has increased enough that the point positions in the first few frames become fairly stable. After that, the initial pose estimates of new frames become more reliable and the computational cost drops off. Throughout both sequences, the computational cost fluctuates as points enter and exit the frames.

Additionally, the computational cost of the flower garden sequence is about the same as that of the lab sequence even though it is less than one third the length of the lab sequence. There are a number of reasons for this. First, the number of projections per frame was higher in the flower garden sequence which tends to scale the computational cost up. Each time a camera position was modified, more points had to be checked and possibly adjusted. Also, the flower garden sequence was about half the resolution of the lab sequence in both dimensions. Because of this, there was more noise in the tracked features, which means more frames were needed for the point positions to settle down. Additionally, the intrinsic parameters of the lab sequence were extracted offline, including the radial distortion parameters. The intrinsic parameters of the flower garden sequence were approximated and radial distortion was ignored. This also led to higher reprojection error in the flower garden sequence.

6. Summary and Conclusions

The complexity of solving for the structure of a scene and the motion of the camera is highly dependent on the specific image sequence. There are many common camera motions for which existing recursive methods will remain

Figure 4. *Three selected frames from the flower garden sequence*

Figure 5. *Three selected frames from the lab sequence*

stable and yield close to optimal results. There are cases, however, where recursive methods will not give accurate results. We have presented a modified bundle adjustment algorithm that always guarantees results as accurate as full bundle adjustment, but will also run in linear time when recursive methods would work. We have demonstrated the viability of our method using real and synthetic sequences.

Our proposed method enables us to automatically and dynamically vary between the speed of recursive methods and the brute force approach of full bundle adjustment.

In the next stage of our research we intend to pursue applying this method to hierarchical parameterizations. Our attempt will be to address the problem of many highly correlated parameters bogging down the convergence. We also intend to work on methods to render the extracted models.

Acknowledgments

Thanks to Arno Schödl for various discussions on this topic.

References

[1] A. Azarbayejani and A. Pentland. Recursive estimation of motion, structure, and focal length. *PAMI*, 17(6):562–575, June 1995.

[2] P. Beardsley, A. Zisserman, and D. Murray. Sequential updating of projective and affine structure from motion. *IJCV*, 23(3):235–259, 1997.

[3] T. Broida, S. Chandrashekhar, and R. Chellappa. Recursive 3-d motion estimation from a monocular image sequence. *AeroSys*, 26(4):639–656, July 1990.

[4] R. Hartley. In defense of the eight-point algorithm. *PAMI*, 19(6):580–593, June 1997.

[5] L. Matthies, R. Szeliski, and T. Kanade. Kalman filter-based algorithms for estimating depth from image sequences. *IJCV*, 3(3. September 1989):209–238, September 1989.

[6] P. McLauchlan. A batch/recursive algorithm for 3d scene reconstruction. In *CVPR00*, pages II:738–743, 2000.

[7] D. Nister. Reconstruction from uncalibrated sequences with a hierarchy of trifocal tensors. In *ECCV00*, pages xx–yy, 2000.

[8] W. H. Press, B. P. Flannery, S. A. Teukoisky, and W. T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 1988.

[9] A. Shashua. Trilinearity in visual recognition by alignment. In *ECCV94*, pages A:479–484, 1994.

[10] H. Shum, Q. Ke, and Z. Zhang. Efficient bundle adjustment with key frames: A hierarchical approach to multi-frame structure from motion. In *CVPR99*, pages II:538–543, 1999.

[11] S. Soatto and P. Perona. Reducing structure-from-motion: A general framework for dynamic vision part 2: Implementation and experimental assessment. *PAMI*, 20(10):1117, October 1998.

[12] R. Szeliski and S. Kang. Recovering 3d shape and motion from image streams using non-linear least squares. *JVCIR*, 5(1):10–28, March 1994.

[13] C. Tomasi and J. Shi. Good features to track. In *CVPR94*, pages 593–600, 1994.

[14] P. Torr and A. Zisserman. Robust parameterization and computation of the trifocal tensor. *IVC*, 15(8):591–605, August 1997.

[15] W. Triggs, P. F. McLauchlan, R. I. Hartley, and A. Fitzgibbon. Bundle adjustment for structure from motion. In W. Triggs, A. Zisserman, and R. Szeliski, editors, *Vision Algorithms: Theory and Practice*, LNCS 1883. Springer Verlag, 2000.