Fast Recovery of Weakly Textured Surfaces from Monocular Image Sequences

Oliver Ruepp and Darius Burschka

Institut für Informatik, Technische Universität München,
Boltzmannstraße 3, D-85748 Garching bei München, Germany
{ruepp,burschka}@in.tum.de

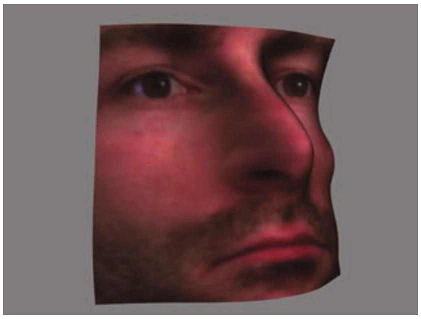
Abstract. We present a method for vision-based recovery of three-dimensional structures through simultaneous model reconstruction and camera position tracking from monocular images. Our approach does not rely on robust feature detecting schemes (such as SIFT, KLT etc.), but works directly on intensity values in the captured images. Thus, it is well-suited for reconstruction of surfaces that exhibit only minimal texture due to partial homogeneity of the surfaces. Our method is based on a well-known optimization technique, which has been implemented in an efficient yet flexible way, in order to achieve high performance while ensuring extensibility.

1 Introduction

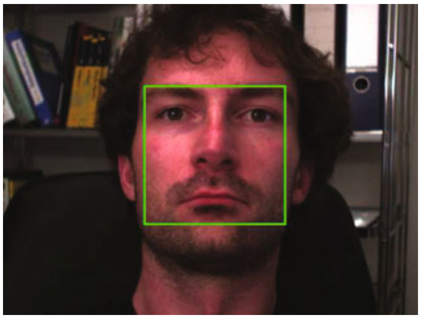
Dense recovery of 3D structures from video data is a problem that has been subject to extensive research work, and a number of methods have been developed for dealing with this problem. Some approaches are, e.g., the methods developed by Newcombe et al. [1], Pan et al. [2] and Palaanen et al. [3]. All of those methods rely on presence of salient image features, such as Good Features to Track [4], SIFT [5] features, FAST edges [6] and so on. In some settings, however, the objects do not exhibit much structure, which makes it very hard to find robust, dense feature sets using traditional methods. In such situations, it pays off to use intensity-based methods, which is what we have investigated.

Our method belongs to the family of intensity-based bundle adjustment techniques. An in-depth survey of the original bundle adjustment method is given in the book by Hartley and Zisserman [7]. The paper by Triggs et al. [8] provides a good overview of more recent developments in bundle adjustment and also briefly explains intensity-based approaches. There is also a more recent paper evaluating the status of real-time bundle adjustment methods [9] using sliding window approaches. The main contribution of our work in this context is the combination of sliding-window and intensity-based bundle adjustment.

Basically, the traditional bundle adjustment algorithm facilitates computation of the 3D position of some salient points in a scene from a number of images taken from different viewpoints. The basic idea is as follows: Coordinates of 3D points are associated with features that are recovered from a set of images using feature detection and matching schemes. This approach will obviously work



(a) Face reconstruction result.



(b) One of the used camera images.

Fig. 1. Example reconstruction result

only if a feature detection scheme is applicable at all. In intensity-based bundle adjustment, we do not assume that robust feature extraction is feasible, and thus we do not work with 2D feature positions, but directly with image intensities.

A number of offline methods for model-based bundle adjustment have been described with applications to face modeling [10,11]. In contrast to these methods, our method can be used on-line, since the time needed for computing a complete bundle update is comparatively small. For the example image in Figure 1a, the time required to process one frame of the sequence was under 1 second.

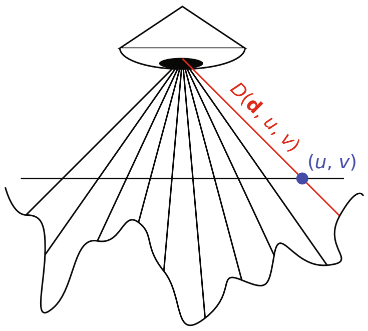
In Section 2, we give a detailed explanation of the mathematics involved. This will lead to the formulation of an optimization problem, which we are implementing as described in Section 3. Results have been obtained from real world data sets as well as synthetic data sets, and are presented in Section 4.

2 Mathematical Formulation

There are many possibilities for representing a model of a scene, with the most straightforward one being a point cloud. This is a very general representation that is actually used in the traditional bundle adjustment algorithm, where it works well under the assumption that those points can be reliably identified. As we have mentioned above, we do not assume that this is possible, since we are planning to work exclusively on intensity measurements. Locating a single point in an image simply because of the point's intensity is obviously infeasible, even if several frames are considered simultaneously. This observation disqualifies point clouds as scene representation for intensity based model-recovery algorithms.

2.1 Surface Model

Usually, some additional assumptions need to be made, usually in the form of a specific surface model that ultimately imposes a smoothness constraint on the observed points. Such a model would be a function of type $S : \mathbb{R}^k \times \mathbb{R}^2 \to \mathbb{R}^3$ that maps a set of k surface parameters together with surface coordinates u, v to



(a) Schematic 2D view of a depth map

(b) Depth map for reconstructed face from Fig. 1a

Fig. 2. The depth map concept

three-dimensional spatial coordinates. A model of this type is especially suitable for representation of scenarios that can be described with a small number of parameters k. This loss of generality is a compromise that seems to be necessary in the difficult situation of 3D reconstruction in scenes with low structure.

In this work, we are using surface models that describe the three-dimensional surface indirectly by specifying a per-pixel depth map $D : \mathbb{R}^k \times \mathbb{R}^2 \to \mathbb{R}$ for a given reference camera image. As illustrated in Figure 2a, the value $D(\mathbf{d}, u, v)$ is supposed to describe the depth for the pixel with coordinates (u, v). Here, $\mathbf{d}$ is the k-dimensional vector of surface parameters for the depth map.

To put this in a formal mathematical framework, we first need to define some basic characteristics of our camera. As is common, we assume a pinhole model with projection function

$$\pi(\mathbf{p}) = \left(\frac{p_1 f_x}{p_3} + c_x, \frac{p_2 f_y}{p_3} + c_y \right)^T \tag{1}$$

where f_x, f_y are focal lengths in terms of pixel dimensions, c_x, c_y describe the location of the camera center, and $(p_1, p_2, p_3)^T$ is a vector of Cartesian point coordinates. In case of significant radial distortions, the images will be rectified before usage.

Each pixel in the image now corresponds to a ray originating from the camera position that intersects the object surface at a certain depth. Assuming that the camera is located at the origin of the coordinate system, the ray corresponding to pixel coordinates (u, v) can then be parameterized by depth λ, yielding a function $r(u, v, \lambda)$:

$$r(u, v, \lambda) = \lambda \cdot \left(\frac{u - c_x}{f_x}, \frac{v - c_y}{f_y}, 1 \right)^T . \tag{2}$$

We can see now that the composite function $S(\mathbf{d}, u, v) := r(u, v, D(\mathbf{d}, u, v))$ describes a three-dimensional surface. Compared to our general definition of

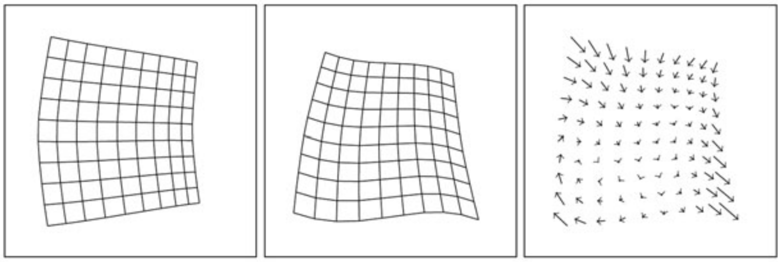


Fig. 3. Left, middle: Surface under two different camera positions. Right: Warping of surface coordinates from left to right image.

a 3D surface as stated earlier, this constitutes a slight restriction. Still, this representation is very well suited to the problem we want to address.

2.2 Optimization Formulation

Observing a static, three-dimensional smooth surface S under two different camera positions will essentially yield two images that are related to each other via a "warping" function. If, for two snapshots of a scene, we exactly know the corresponding extrinsic camera parameters and we have a perfect mathematical description of the surface that we are observing, we can, for each surface pixel in one image, determine the position of that pixel in the other image. In other words, we can formulate a coordinate warping function of type $\mathbb{R}^2 \rightarrow \mathbb{R}^2$ that transforms pixel coordinates from one image to another. Figure 3 shows an example for the coordinate warping function.

Consequently, we would then expect corresponding image intensities to be equal, thus the coordinate warping also defines an image warping, assuming that all pixels are visible. This is the case if there are no occlusions, and the surface does not move out of the camera image. To assure the latter, the depth map we are talking about is not applied to the whole reference frame, but only to a user-chosen rectangular region of interest within the image. As an example, the region of interest is marked with a green rectangle in Figure 1b.

The idea of our approach is now basically the same as in traditional bundle adjustment: Using a nonlinear optimization technique, we are able to compute parameters for the warping function that best explain the observations. Thus, we are able to determine a good approximation of the warping function itself.

To formulate the optimization problem, we need to define a cost function. Before we proceed with the description of that function, we will give a short summary of definitions and notations used. In the following, images are numbered consecutively, and the numbering starts with $n = 0$. Letters set in italic represent scalar-valued values, while bold-faced letters denote vector-valued quantities.

- $\mathbf{d}$ denotes the k-dimensional vector of parameters describing the depth map.
- $D(\mathbf{d}, u, v)$ denotes the depth map function itself.

- $\mathbf{c}_n = (\mathbf{t}_n, \mathbf{q}_n)$ denotes the extrinsic camera parameters corresponding to image n, consisting of translation vector $\mathbf{t}_n \in \mathbb{R}^3$ and rotation quaternion $\mathbf{q}_n \in \mathbb{R}^4$.
- $T(\mathbf{t}_n, \mathbf{q}_n, \mathbf{p}) : \mathbb{R}^3 \times \mathbb{R}^4 \times \mathbb{R}^3 \to \mathbb{R}^3$ is a transformation mapping 3D spatial coordinates $\mathbf{p}$ to 3D coordinates in the camera frame described by $\mathbf{c}_n$.
- $\pi(\mathbf{p})$ is the projection of a 3D point $\mathbf{p}$ to 2D image coordinates.
- $I_n(x, y)$ is the image function of image n. I_0 is the reference image function.

Using this notation, we can define the image coordinate warping function for a certain frame n as follows:

$$w(\mathbf{d}, \mathbf{c}_n, u, v) := \pi(T(\mathbf{c}_n, r(u, v, D(\mathbf{d}, u, v)))). \tag{3}$$

If we knew the perfect model parameters $\mathbf{d}$ and exact camera parameters $\mathbf{c}_n$ for image n, we would expect the relationship $I_n(w(\mathbf{d}, \mathbf{c}_n, u, v)) = I_0(u, v)$ to hold for all model surface coordinates (u, v).

This leads to the assumption that the correct camera position and the correct model parameters together minimize some difference measure c (e.g., least squares) on intensity values. The optimization process that determines camera and model parameters is computationally quite expensive. Thus, we will not include all possible pixel coordinates in the optimization process, but only the coordinates of m chosen reference points $(u_1, v_1), \ldots, (u_m, v_m)$. The corresponding objective function $o(\mathbf{d}, \mathbf{c}_n)$ can then be defined as

$$o(\mathbf{d}, \mathbf{c}_n) = \sum_{i=1}^{m} c(I_n(w(\mathbf{d}, \mathbf{c}_n, u_i, v_i))) - I_0(u_i, v_i)) \tag{4}$$

Our problem of finding a warping function from the template image I_0 to the current image I_n is now formulated as the problem of minimizing the error function with respect to camera and depth map parameters.

We can easily generalize above objective function to a set of views $V = \{\mathcal{V}_1, \mathcal{V}_2, \ldots\} \subseteq \{1, 2, \ldots, n\}$ (when n images have been acquired) by defining a new objective function

$$o'(\mathbf{d}, \mathbf{c}_{\mathcal{V}_1}, \mathbf{c}_{\mathcal{V}_2}, \ldots) = \sum_{j \in V} o(\mathbf{d}, \mathbf{c}_j) \tag{5}$$

This variant is used in our implementation, where we optimize simultaneously over the so-called sliding window of $|V|$ images. In our application, the size $|V|$ of the sliding window is fixed.

Note that above objective functions can easily be modified to formulate a traditional coordinate-based bundle adjustment problem. All we have to do is remove the functions I_j and I_0 from the formula. This is important to know since coordinate-based bundle adjustment will be used to initialize the system.

There are two minor issues that we should also address: Because quaternions are used to represent the rotation of the camera frame, we need to constrain the corresponding parameters $\mathbf{q}_n$ to represent a unit quaternion, and thus, a unit vector. This can trivially be formulated as a constraint $h_1(\mathbf{q}_n) = 0$ with

$h_1(\mathbf{q}_n) = |\mathbf{q}_n|^2 - 1$. Furthermore, it is well-known that reconstruction from monocular images can only be done up to scale. However, it is 0 then at least to enforce a constant scale during the reconstruction process. This can be achieved with the formulation of a constraint $h_2(\mathbf{d}) = 0$ with $h_2(\mathbf{d}) = D(\mathbf{d}, u_1, v_1) - l$ for some constant l.

Since through optimizing above function, we implicitly try to track point positions by intensity values, our approach could have difficulties tracking points in areas with completely homogeneous intensity. Thus, to improve the tracking results, the reference points are chosen from the ROI in such a way that they lie at pixel positions where the image derivative is non-zero.

Furthermore, reference points should be distributed in the region of interest such that the parameters determining the depth map are well constrained. This depends on the specific model used. For a B-Spline depth map model, one will, e.g., need at least a number of reference points that is equal to the number of control points used.

3 Optimization

It is clear that, to actually recover the model parameters from the scene, we need some method to minimize the cost function described above. Since we are dealing with a constrained problem, an adequate method for optimization is Sequential Quadratic Programming (abbreviated as SQP) [12].

The basic idea is as follows: Let $f : \mathbb{R}^k \to \mathbb{R}$ be a scalar function to be minimized, and let $h : \mathbb{R}^k \to \mathbb{R}^l$ be a function that describes a constraint of the form $h(x) = 0$ on solutions. It is well-known that for such problems, the so-called Karush-Kuhn-Tucker (KKT) conditions must hold for any value x^* that is a minimum. These conditions can be formulated in equation form as:

$$\begin{pmatrix} \nabla \mathcal{L}(x, \lambda) \\ h(x) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad \text{with} \quad \mathcal{L}(x, \lambda) = f(x) + \lambda^T h(x). \tag{6}$$

The term $\lambda \in \mathbb{R}^l$ is the Lagrange multiplier associated with the minimum. This is, in general, a nonlinear system of equations. The Lagrange-Newton Method can be applied to these equations, and we can compute an update Δx to x and a new Lagrange multiplier λ^+ by solving the equation system

$$\begin{pmatrix} \nabla^2_{xx} \mathcal{L}(x, \lambda) & \nabla_x h(x) \\ \nabla_x h(x)^T & 0 \end{pmatrix} \begin{pmatrix} \Delta x \\ \lambda^+ \end{pmatrix} = - \begin{pmatrix} \nabla_x f(x) \\ h(x) \end{pmatrix}. \tag{7}$$

When implementing this algorithm, we obviously need to compute the Hessian $\nabla^2_{xx}\mathcal{L}$ as well as the transposed Jacobian $\nabla_x h$ of h. Since f is, in our case, a quite complex composition of multi-dimensional functions, it is not feasible to compute the exact Hessian. Instead, it is common practice to use the Gauss-Newton approximation of the Hessian, as detailed below.

Note that the Hessian of $\lambda^T h(x)$ is, on the other hand, easy to determine: Since our constraints are all linear or quadratic, all second-order derivatives are reasonably easy to compute, so the exact Hessian can be established.

3.1 Implementation Details

In our case, the objective function f is the composition $c \circ g$ of a scalar cost function c with some multi-dimensional comparison function g. The typical choice for a cost function would be the least-squares cost $g(x) = x^T x$, but it is well-known that this cost function is very susceptible to outliers. Thus, we are instead using the pseudo-Huber cost function [7, p. 619], which is known to be very robust. Independent of the actual cost function used, the Hessian of the total cost is approximated as:

$$\nabla^2_{xx}(c \circ g)(x) \approx (\nabla_x g)(x) \cdot (\nabla^2_{xx} c)(f(x)) \cdot (\nabla_x g)^T(x)$$

We see that in order to estimate the Hessian, we need to compute the Jacobian of the comparison function g. That Jacobian is sparse, which means that the Hessian will also be sparse. As in the traditional bundle adjustment algorithm, exploiting the sparsity structure is extremely important.

The sparse Jacobians of f and h are computed using our own variant of Automatic Differentiation (AD) [13]. Our method can be described as a hybrid between traditional AD and symbolic differentiation. Traditional AD basically treats all functions as function compositions. AD is able to compute derivatives of elementary functions, such as basic arithmetic, cos, sin, exp and so on, directly, while the derivatives of composite functions are computed according to the chain rule.

The accuracy of AD is very good, especially when compared to finite difference approximation. Derivatives computed by AD are usually accurate up to machine precision. Additionally, the performance of AD is known to be good as well, since the time required to evaluate derivatives of a function is proportional to the time needed to evaluate the original function. However, when evaluating sparse Jacobians, plain AD is usually not optimal, and finding the most efficient way to compute sparse Jacobians within the framework of AD is actually an NP-complete problem [14]. That problem is solved in state-of-the-art AD implementations by means of heuristics.

Our approach for computing sparse Jacobians also relies on the chain rule, but it is applied at a different level: For a composition of vector-valued functions $f_1 \circ f_2 \circ \ldots \circ f_n$, the Jacobian can be computed as a sparse chained matrix product $J_{f_1} \cdot J_{f_2} \cdot \ldots \cdot J_{f_n}$. We choose to implement computation routines for the Jacobians J_{f_i} directly instead of starting with differentiating the most basic functions. The overall Jacobian is then evaluated as sparse matrix product with optimized bracketing. It is well-known that the bracketing of a matrix chain product is essential to evaluation performance [15, p. 331]. This observation also applies to sparse chain matrices, and is also the basis of a highly-efficient heuristic for traditional AD [16]. In our case, the sparsity structure of the Jacobians never changes, so the optimal bracketing that has been determined once for a certain objective function remains the same.

Implementing Jacobian evaluation code for vector-valued functions directly obviously involves additional work and is error prone, but our method also has two advantages over traditional AD:

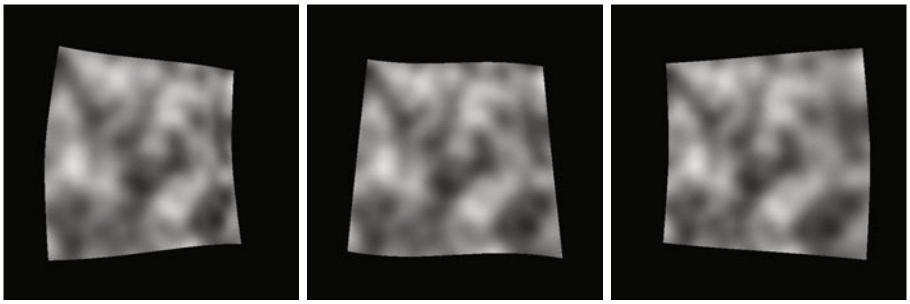


Fig. 4. Samples from artificial image sequence

- Speed: When used properly, our method can be substantially faster than traditional AD. For the coordinate transformation function T, as described in Section 2, we have compared the performance of our own implementation and that of an ADOL-C [17] based variant, and found that our method is about three times as fast.
- Flexibility: Some functions resist treatment by AD, especially in cases where the function to be differentiated is defined in a piecewise fashion. This is the case for our image functions, which are made continuous and differentiable using bicubic interpolation. However, it is reasonably simple for a human to implement efficient evaluation algorithms for the interpolated image function as well as its derivatives.

After computation of the Jacobian is finished, the approximate Hessian can be evaluated and the QP system is solved repeatedly. For increased robustness of this process, we add a damping term λI to the Hessian of $\mathcal{L}$. This method is well-known in the context of Levenberg-Marquardt optimization [18, 19] and can be applied to the SQP method as well. The equation system itself is then solved by employing a sparse Cholesky transformation on the whole system. The efficient Eigen library for Linear Algebra[1] is used to handle this. Finally, when a solution to the system has been found, a simple step size search according to the Armijo rule is performed.

3.2 Bundle Adjustment

Now that we have established the tools to solve the original optimization problems stated in Section 2, we can describe how the methods are actually applied to achieve model reconstruction.

First of all, we need to initialize the system. We assume that the user has chosen the template image and an ROI for the surface to be reconstructed. For a given window size $|V|$, we acquire images $\{1, \ldots, |V|\}$ and compute the optical flow [20] on these, which yields initial correspondences between the template image and the images to be used for initialization. One run of full coordinate-based bundle adjustment on those correspondences yields a starting point for further

[1] `http://eigen.tuxfamily.org/`

intensity-based optimization, which is carried out after the initialization through optical flow. While the coordinate-based optimization serves to provide a rough estimate of the surface and camera positions, the intensity based optimization is used to refine the initially found parameters.

After the initialization, the algorithm alternates between coordinate based optimization steps (optimizing towards optical flow results) and intensity based optimization. No full bundle adjustment on optical flow information is performed any more. Instead, the coordinate based optimization is used only to determine the camera position. With camera parameters initialized, a full intensity-based bundle optimization is carried out to refine the parameters.

To achieve usable results and assure that the algorithm is stable, one must also be careful which images to select for the optimization window V. Simply choosing consecutive images might be a bad idea, because if the camera stops moving for some time, a number of images from the same position will be taken. If those images are placed inside the window, the optimization process will become very unstable, since depth reconstruction is impossible without a certain minimal baseline.

In our implementation, new images in the window are only accepted if their difference of baseline to the previous image in the window is big enough. Whenever a new image is accepted, the oldest image in the window is discarded. This is obviously a rather crude algorithm, but it helps to avoid the most obvious problems. We are planning to implement more sophisticated methods in the future.

4 Results

We have tested our algorithm on a set of artificial rendered image sequences, as well as on sequences of real scenes. The artificial data set was useful for generating images with known ground truth, while the sequences of real images have been used to show that the approach also works in the "real world." As depth map model, we have used B-Spline surfaces of varying order and complexity.

Our first tests were on artificial images generated by a renderer. Here, we show results for one of the used sequences. Figure 4 shows an example image from the sequence, showing a surface with a texture that exhibits only intensity gradients, and almost no structure. Because we wanted to get a rough idea of how well traditional approaches would work on that sequence, we ran a SIFT feature detector on some of the images. The feature detection process resulted in about 20 features, depending on the actual image. Even when assuming that all features can be reliably identified through the whole sequence, and that no false feature matchings occur, this is by far not enough to fully describe the complexity of the actual surface. The surface is a quadratic spline surface determined by 25 control points (5 in each direction).

To compare the reconstructed surface to the ground truth, we have determined the normalized cross-correlation (NCC) between the ground truth surface of the sequence and the reconstruction result. The result of this comparison was that the reconstruction is extremely accurate, yielding surface models that achieved a NCC ratio of over 0.96, where 1 is the best possible value.

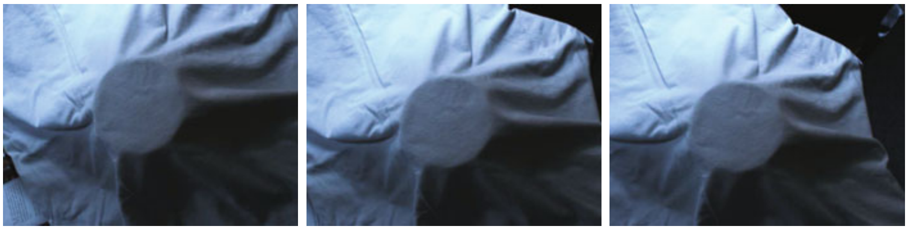


Fig. 5. Samples from real-world image sequence

Fig. 6. Template image, depth map, and 3D rendering

The artificial sequences have been used because it is really difficult in a real-world scenario to determine the ground truth. Still, it is important to show that our approach also works on actual data generated from a camera. Hence, we have tested our method an scene that was showing a piece of white cloth draped over a cup. You can see some images of the recorded sequence in Figure 5. Figure 6 shows the template image, the associated depth map, and the resulting 3D model.

As can be seen, the reconstruction quality is still good, despite the lack of texture in the scene. It is clearly possible to recognize the shape of the cup underneath the cloth.

As for running times: Our algorithm has been tried on a system with an Intel Core i7-820QM 1.73 GHz quad core CPU, using only one of the CPU cores. Running time generally depends on the resolution of the surface model, but generally, one frame was processed in under one second. The major time spent during reconstruction was due to intensity-based optimization. The convergence of the intensity-based optimization was rather slow, which is probably due to the non-convex nature of the cost function in case of large displacements of the tracked pixels to the optimal position. Still, the performance is promising, and we expect it to be possible to further improve performance by pursuing more elaborate optimization schemes.

5 Conclusion

The basis for further research has been established with our monocular model recovery and tracking algorithm. There are many possible extensions and improvements to this technique.

First of all, while the reference-point based reconstruction works well, it would probably constitute a major improvement if we were able to capture, in addition to point intensity values, a comparison of texture gradients in the area surrounding the reference points. We would expect this to further improve the stability and convergence speed of the optimization method.

While the algorithm is already quite fast, there is still a lot of potential for speed improvement. There exist more specialized algorithms that could be used for solving the QP equations [21]. GPU algorithms could be used for performing image subsampling, which constitutes the major part of the current computation time consumption. Finally, it should also be possible to speed up the involved geometry computations using the GPU.

Furthermore, we did not address the issue of changing illumination conditions. We would like to be able to deal with changes in brightness, but also with specularities, which would, in the current approach, both cause severe problems. However, some techniques for dealing with problems of that kind have already been developed, e.g., normalized cross-correlation matching for brightness-invariant matching. It should be possible to integrate them into our method.

We would also like to extend the approach such that deformable surfaces can be reconstructed and tracked. For tackling this problem, we intend to use a setup of two independently moving cameras. Based on such an idea, we would like to introduce a method for determining deformation parameters, allowing us also to predict and simulate deformations. We see applications for such a technique mainly in medical imaging.

Acknowledgement. This research has been funded by The German Heart Centre Munich.

References

1. Newcombe, R., Davison, A.: Live dense reconstruction with a single moving camera. In: IEEE Conference on Computer Vision and Pattern Recognition CVPR (2010)
2. Pan, Q., Reitmayr, G., Drummond, T.: ProFORMA: Probabilistic Feature-based On-line Rapid Model Acquisition. In: Proc. 20th British Machine Vision Conference (BMVC), London (2009)
3. Paalanen, P., Kyrki, V., Kamarainen, J.K.: Towards Monocular On-Line 3D Reconstruction. In: Workshop on Vision in Action: Efficient strategies for cognitive agents in complex environments, Marseille France, Markus Vincze and Danica Kragic and Darius Burschka and Antonis Argyros (2008)
4. Shi, J., Tomasi, C.: Good features to track. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR 1994), pp. 593–600 (1994)
5. Lowe, D.G.: Object recognition from local scale-invariant features. In: ICCV, pp. 1150–1157 (1999)
6. Rosten, E., Drummond, T.W.: Machine learning for high-speed corner detection. In: Leonardis, A., Bischof, H., Pinz, A. (eds.) ECCV 2006. LNCS, vol. 3951, pp. 430–443. Springer, Heidelberg (2006)
7. Hartley, R.I., Zisserman, A.: Multiple View Geometry in Computer Vision, 2nd edn. Cambridge University Press, Cambridge (2004) ISBN: 0521540518

8. Triggs, B., McLauchlan, P.F., Hartley, R.I., Fitzgibbon, A.W.: Bundle adjustment - a modern synthesis. In: Proceedings of the International Workshop on Vision Algorithms, ICCV 1999, pp. 298–372. Springer, London (2000)
9. Engels, C., Stewénius, H., Nistér, D.: Bundle adjustment rules. In: Photogrammetric Computer Vision (PCV), ISPRS (2006)
10. Fua, P.: Using model-driven bundle-adjustment to model heads from raw video sequences. In: Proceedings of the Seventh IEEE International Conference on Computer Vision, vol. 1, pp. 46–53 (1999)
11. Shan, Y., Liu, Z., Zhang, Z.: Model-based bundle adjustment with application to face modeling. In: Proceedings of Eighth IEEE International Conference on Computer Vision, ICCV 2001, vol. 2, pp. 644–651 (2001)
12. Nocedal, J., Wright, S.J.: Numerical Optimization. Springer, Heidelberg (2000)
13. Griewank, A., Walther, A.: Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation, 2nd edn. Other Titles in Applied Mathematics, vol. 105. SIAM, Philadelphia (2008)
14. Naumann, U.: Optimal jacobian accumulation is np-complete. Math. Program. 112, 427–441 (2007)
15. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. The MIT Press and McGraw-Hill Book Company (2001)
16. Griewank, A., Naumann, U.: Accumulating jacobians as chained sparse matrix products. Math. Program. 95, 555–571 (2003)
17. Griewank, A., Juedes, D., Utke, J.: Algorithm 755. ADOL-C: A package for the automatic differentiation of algorithms written in C/C++ 22, 131–167 (1996)
18. Levenberg, K.: A method for the solution of certain non-linear problems in least squares. Quarterly Journal of Applied Mathmatics II, 164–168 (1944)
19. Marquardt, D.W.: An algorithm for least-squares estimation of nonlinear parameters. SIAM Journal on Applied Mathematics 11, 431–441 (1963)
20. Lucas, B.D., Kanade, T.: An iterative image registration technique with an application to stereo vision (darpa). In: Proceedings of the 1981 DARPA Image Understanding Workshop, pp. 121–130 (1981)
21. Davis, T.A., Hager, W.W.: Dynamic supernodes in sparse cholesky update/downdate and triangular solves. ACM Trans. Math. Softw. 35, 1–23 (2009)