Affordable Infrared-Optical Pose-Tracking for Virtual and Augmented Reality

Thomas Pintaric* Hannes Kaufmann†

Interactive Media Systems Group
Institute of Software Technology and Interactive Systems
Vienna University of Technology

Abstract

In this paper, we describe the hard- and software of a new low-cost infrared-optical pose-tracking system for room-sized virtual environments. The system consists of 4-8 shutter-synchronized 1394-cameras with an optical bandpass filter and infrared illuminator. All image-processing is done in software on an attached workstation. Preliminary results indicate low latency (20-40 ms), minimal jitter (RMS less than 0.05 mm/0.02°), submillimeter location resolution and an absolute accuracy of ±0.5 cm. Currently, ten independent 6-DOF targets can be tracked in real-time with up to 60Hz.

Keywords: Infrared-optical 6-DOF pose-tracking, low-cost.

Index Terms: I.4.0 [Computing Methodologies]: Image Processing and Computer Vision—General; I.3.8 [Computing Methodologies]: Computer Graphics—Applications

1 Motivation

Time and again, a lack of affordability has delayed or averted the adoption of an otherwise viable technology. In the context of Virtual and Augmented Reality (VR/AR), the single largest expense for building a room-sized virtual environment is almost always incurred by the acquisition of a wide-area 6-DOF pose-tracker.

Only a few vendors offer commercial systems that extend beyond 20 square meters in coverage and are capable of tracking at least six independent targets with a precision, latency and measurement frequency that is considered adequate for a range of VR/AR applications (see Section 2). Systems meeting those criteria include 3rd Tech's HiBall [1] (inside-out, infrared-optical), the Intersense IS-1200 [2] (inside-out, hybrid optical/inertial) and A.R.T. ARTrack[1] (outside-in, infrared-optical). A brief survey of available optical tracking systems is given in [3]. In a configuration tailored to a room-sized multi-user environment, all of the aforementioned systems have price tags in the range of tens of thousands of Euros. While corporate entities and well-funded research laboratories will not be deterred by such amounts, it is the authors' first-hand experience that many smaller educational institutions, especially secondary schools, operate on tightly constrained budgets that leave little, if any, room for an expense of this magnitude, even if third-party subsidies are available.

It was an attempt to build an affordable hardware platform for *Construct3D* [4], a collaborative Augmented Reality application designed to supplement the mathematics and geometry education curriculum of secondary schools, that prompted us to build our own wide-area tracker. Born out of necessity in early 2005, our tracking system has since undergone three design iterations and is currently deployed at locations in three different countries. At the time of this writing (January 2007), we are beginning to make the system commercially available to interested parties for a price of six to eight thousand Euros (depending on the configuration). Our goal is to provide an affordable, high-quality tracking to a broad range of smaller VR/AR application developers.

The remainder of this paper summarizes our requirement analysis, outlines hard- and software design choices, and gives preliminary performance results. It is intended as a high-level overview, not an in-depth discussion of algorithmic or constructional details.

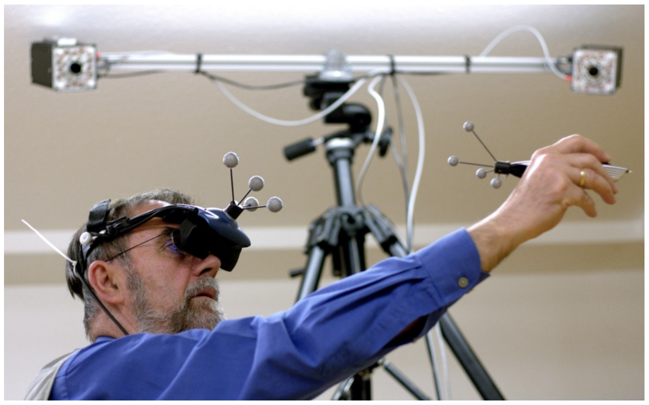


Figure 1: An early prototype of our pose-tracker in a mobile, tripod-mountable configuration for room-sized virtual environments.

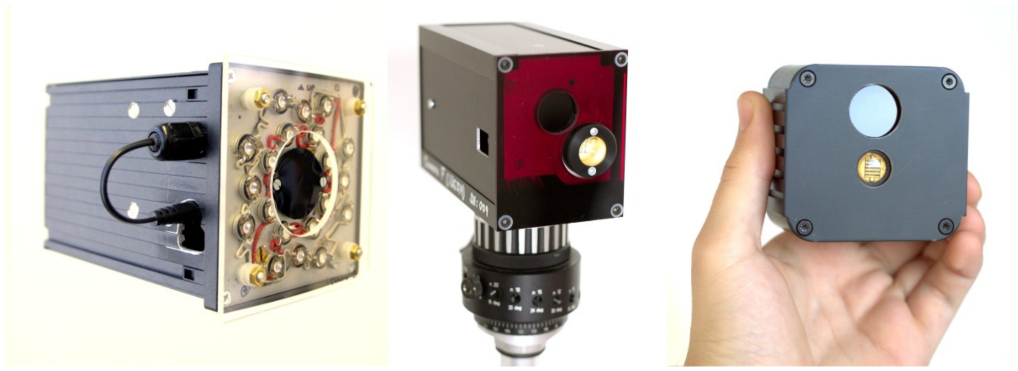
2 Requirement Analysis

Choosing commodity hardware over custom-built components has always been a reliable cost-minimization strategy. The ready availability of inexpensive electronic imaging devices with a standardized computer interface (e.g. IEEE-1394) prompted us to chose optical tracking over competing technologies, mainly because it would allow for all computation to be done on a regular PC workstation. Within the domain of optical tracking, outside-in trackers, usually composed of wall- or tripod mounted cameras directed towards the center of the interaction volume, exhibit better scalability (in a monetary, not a computational sense) to multiple targets than inside-out self-trackers. Additionally, the installation and configuration of outside-in trackers is usually faster and easier than that of existing inside-out systems.

The basic principle of optical outside-in tracking lies in estimating the pose of known rigid 3D feature-point constellations, commonly referred to as "targets", from their projections in different views. To simplify the task of identifying relevant features in a camera image, targets are usually built from clusters of light-emitting diodes ("active targets") or retro-reflective spheres ("passive targets"). In the latter case, which we opted for, all cameras must be equipped with a strobelight. The added expense is an initial dis-

*e-mail: pintaric@ims.tuwien.ac.at
†e-mail: kaufmann@ims.tuwien.ac.at

[1]`http://www.ar-tracking.de`


(a) Development prototypes from early 2005 (left), mid-2006 (middle) and final camera design (right).

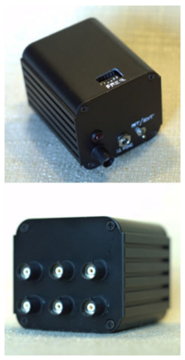

(b) Trigger box.

Figure 2: Hardware design.

advantage, but is made up for by the fact that passive targets are cheaper to build and maintain, because they do not require a power source. As long as the system is operating under controlled lighting conditions, which is usually the case in an indoor environment, the choice of near-IR strobelights (typically in the 750-950 nm range) combined with optical band-pass filters helps to avoid user irritation and increases the speed and accuracy of the image segmentation by reducing unwanted contributions from ambient light sources.

Accuracy and Precision: While of lesser importance for fully synthetic virtual environments, closely registered AR overlays impose rather stringent requirements on tracking precision and accuracy. We consider a location resolution of less than 1.0 mm, with a minimum accuracy of ± 5.0 mm, and an angular precision of better than 0.05 degrees as adequate for a wide range of AR applications.

Latency: It is commonly accepted that HMD rendering latencies beyond 40-60 ms will start to negatively affect user performance, possibly inducing symptoms of "simulator sickness" (i.e. fatigue, loss of concentration). According to recent experiments [5], trained users are able to discriminate variations of end-to-end latencies as small as 15 ms. To be suitable for use with an HMD, a tracker's intrinsic latency should under no circumstances exceed 30-40 ms.

Update rate: We attempted to match our tracker's measurement frequency to the update rates of the visual output devices it would eventually be combined with. Suitable cameras within our price range were limited to a maximum of 30-60 frames per seconds. Incidentally, the current generation of lower-end HMDs (e.g. eMagin's Z800[2]) offers refresh rates of no more than 60 Hz. Hence, no benefit would be gained from providing higher update rates.

3 Hardware Design

3.1 Cameras

Our most recent camera design (see Figure 2a) is based on the FireFly MV[3] monochrome OEM board camera from Point Grey Research, which is built around the Micron FFMV-03MTM 1/3" global-shutter CMOS-sensor and capable of delivering a video resolution of 640x480 at 60 frames per second. It has external trigger and strobe functionality, and costs under USD 200. After attaching a 3.6 mm f 2.0 microlens with an infrared band-pass filter (RG-780), the camera module is mounted inside a 71x65x40 mm aluminum enclosure, next to a 60-chip LED-array with a peak wavelength of 850 nm, radiant intensity of 0.4 W/sr (watts per steradian) and viewing half-angle of ± 60 degrees. The design further includes an opto-isolated trigger input, constant-current LED driver and the necessary circuitry for interfacing with the camera board's TTL I/O pins. The device is powered by an external 12 VDC (1A) power supply and connects to a PC workstation via 6-pin firewire (1394a) cable, which can reach a length of up to 10 meters.

Multiple cameras are shutter-synchronized from a 20 mA square-wave current loop signal generated by a synchronization unit (see Figure 2b) with a built-in programmable oscillator. The synchronization unit supports arbitrary phase-shifting between cameras, and can be locked onto an external stereo synchronization signal.

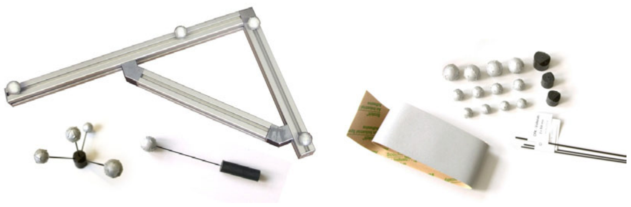


Figure 3: L-shaped room calibration target, four-marker target, single-marker extrinsic calibration target and building materials.

3.2 Targets

We build our passive retro-reflective targets from self-adhesive sheets of 3M ScotchLite Silver Transfer Film which are cut into shape and wrapped around solid nylon (PA 66) spheres of varying diameters (between 12 and 20 mm, depending on the intended use of the target). The reflective spheres are subsequently attached to a cone-shaped nylon base via thin carbon-fiber rods. We chose nylon over other plastics for its good machinability and superior adhesive properties, and carbon-fiber for its structural strength and resistance against breaking and permanent bending. Figure 3 shows the materials used, and resulting final targets.

4 Software Architecture

The diagram in Figure 4 depicts the organization of our system's online image-processing pipeline, with the exception of camera calibration, which is treated as a separate offline step. Given a calibrated camera rig, the runtime behavior is as follows:

After a new frame is received from an attached camera (*Video Input*), an image segmentation algorithm identifies the 2D coordinates of every blob feature (i.e. screen-space projection of a 3D marker) in the video image (*Feature Segmentation*). For every set of shutter-synchronized frames, the system optimally solves the feature correspondence problem across all views (*Multiple-View Correlation*).

[2]http://www.3dvisor.com
[3]http://www.ptgrey.com/products/fireflymv

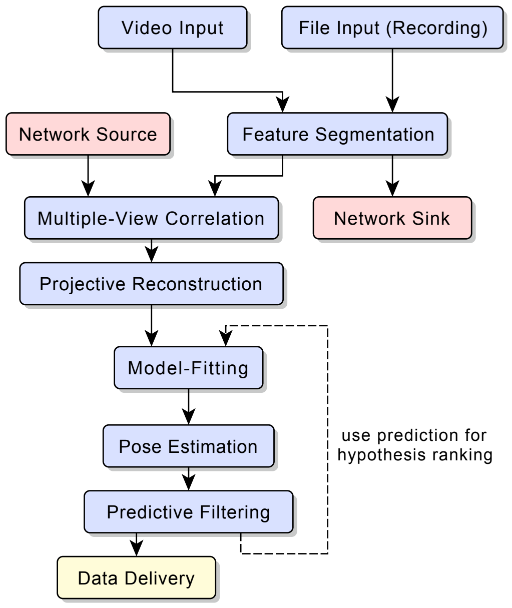


Figure 4: Runtime image-processing pipeline.

A 3D position is computed for each correlated 2D feature-tuple via projective triangulation (*Projective Reconstruction*). The resulting 3D point cloud is examined to find sub-structures that are similar to a pre-calibrated target (*Model-Fitting*). If a target is discovered, its position and orientation with respect to the tracker's world coordinate system is estimated (*Pose Estimation*). In order to reduce pose jitter and to compensate for the tracker's intrinsic latency, the resulting pose estimate is fed into a recursive filter (*Predictive Filtering*). Finally, the filtered pose estimate is handed to a middleware component responsible for streaming the tracking data to clients over a network (*Data-Delivery*).

5 Algorithms and Implementation

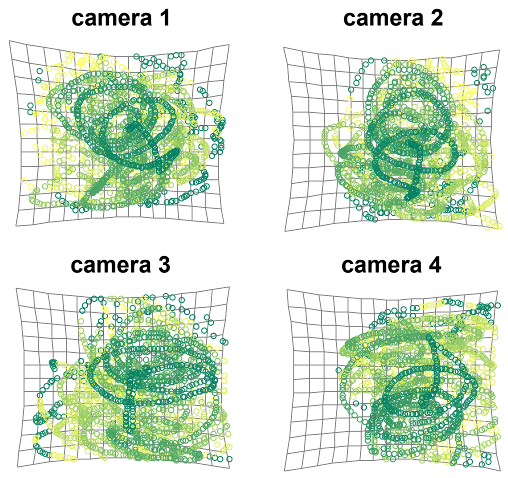
With the sole exception of intrinsic camera calibration, for which we currently use a MATLAB toolbox, our software framework was written entirely in C++. In addition to modularity and platform-independence, special emphasis has been placed on shared-memory parallelizations of the core algorithms in order to take maximal advantage of modern multi-core CPU architectures. In some instances, we made use of existing public-domain software libraries (such as VRPN [6]), but for the most part, we re-implemented standard textbook techniques. Since their in-depth discussion would lack particular merit, we refer the reader to the pertinent literature [7, 8] on projective geometry for a detailed and exhaustive treatment of the underlying mathematical theory. Noteworthy exceptions are multiple-view feature correlation and model-fitting, both of which do not lend themselves to straight-forward parallel implementations. Our approach was to re-state those problems as combinatorial optimizations on graphs and use a parallel maximum-clique enumeration algorithm to reduce the solution space. Unfortunately, limited space precludes us from giving anything more than a brief overview of those techniques.

5.1 Camera Calibration

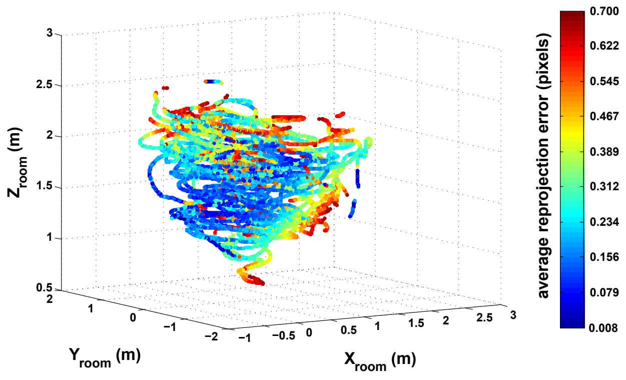
Although several multiple-camera calibration methods exist that simultaneously solve for intrinsic and extrinsic parameters (e.g. [9, 10] and others), we chose to treat the two problems separately – primarily because a camera's intrinsic parameters will have to be determined far less frequently (ideally only once) than its exterior position and orientation. Additionally, combined approaches tend to be computationally more expensive.

5.1.1 Intrinsic Parameters

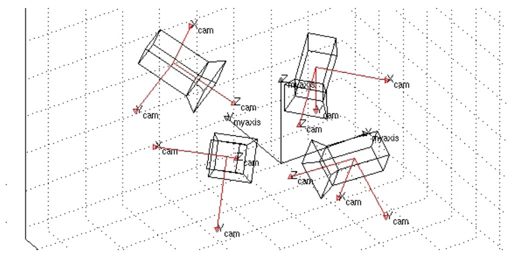
We currently use a variant of Jean-Yves Bouguet's well known and widely-used *Camera Calibration Toolbox for* MATLAB[4] with an extension[5] for the automated corner extraction of non-symmetric checkerbox calibration patterns to calibrate our cameras' intrinsic parameters. The toolbox uses a pinhole camera model with non-linear radial and tangential distortion compensation which was inspired by Heikkilä [10].



(a) Projections of the calibration target in multiple views.



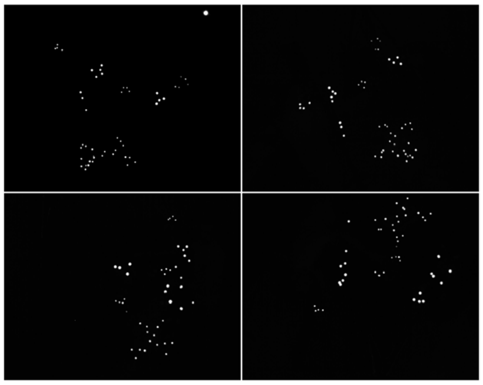
(b) Reconstructed 3D-path of the calibration target. Coloring indicates the average reprojection error at different points inside the volume.

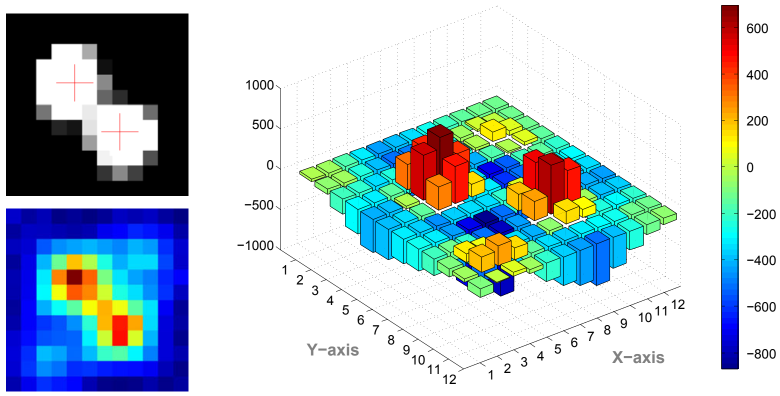


(c) Exterior pose of the cameras (not drawn to scale).

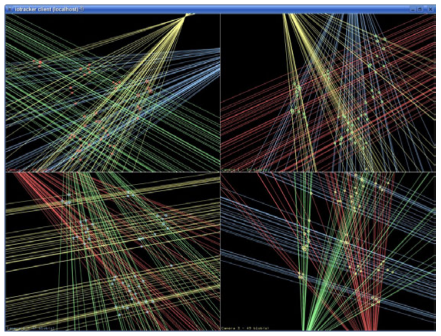
Figure 6: Extrinsic calibration (4 cameras).

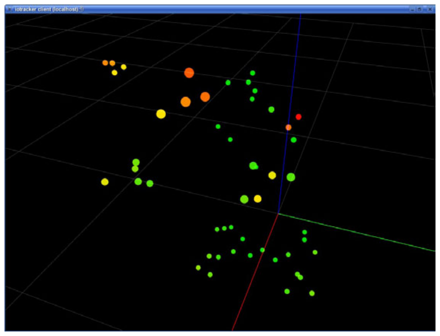
[4]http://www.vision.caltech.edu/bouguetj/calib_doc
[5]http://research.graphicon.ru/computer-vision


(a) Four shutter-synchronized video images.



(b) Recovered centers of overlapping blobs (via circular Hough transform).


(c) 2D features and corresponding epipolar lines.


(d) Reconstructed 3D marker positions.

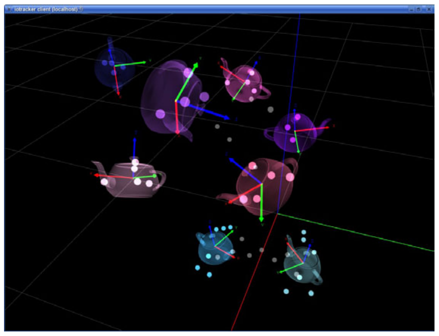

(e) Estimated 6-DOF pose of multiple rigid targets.

Figure 5: Stages of the data-processing pipeline: from raw video images to 6-DOF pose estimates.

5.1.2 Extrinsic Parameters

In order to accurately estimate a camera rig's exterior position and orientation, we implemented parts of the automated single-point self-calibration technique published by Svoboda et al. [11] in C++. Instead of using a laser-pointer, we move a single retro-reflective marker ("extrinsic calibration target"), as depicted in Figure 3, through the interaction volume, apply our segmentation algorithm (see Section 5.2) and record its resulting screen-space coordinates. After discarding outliers using pair-wise RANSAC analysis, the point cloud's projective structure is computed by rank-factorization and refined through bundle adjustment. The final step is Euclidean stratification that converts the projective reconstruction into Euclidean structures. The minimal average reprojection error (all cameras, all points) achieved by this method lies between 0.15 and 0.35 pixels. The running time for six cameras with pre-calibrated intrinsic parameters, and an average of 2500 marker reprojections per camera, is below five minutes. Figure 6 illustrates the calibration procedure.

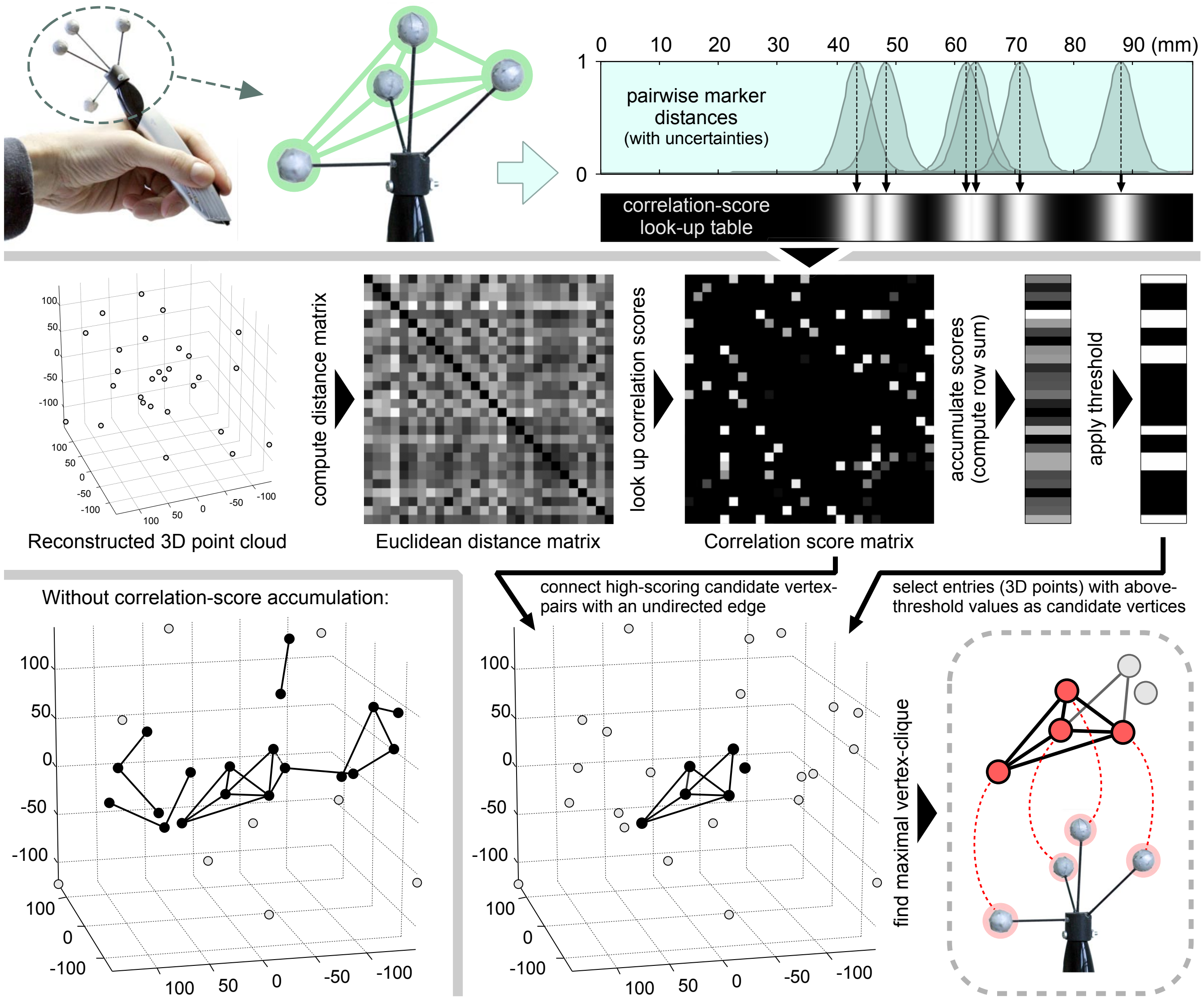
In a final step, the affine transformation between the (arbitrary) coordinate frame returned by Svoboda's algorithm and a user-defined room coordinate system with real-world distance units (mm) is computed. We place an L-shaped constellation of reflective markers ("room calibration target"), as shown in Figure 3, at the intended coordinate origin inside the working volume. The target's marker positions were previously measured to submillimeter accuracy with an industrial theodolite system (Leica TPS700). Using a scale-invariant version of our model-fitting algorithm (described in Section 5.4), we obtain the set of optimal correspondences between calibration target and observed point cloud. The scaling factor is calculated from the ratio of the sum of entries in both distance matrices (over multiple frames), and a 3D rigid transformation between the two sets is computed as outlined in Section 5.5.

5.2 Feature Segmentation

A fast, robust and accurate feature segmentation algorithm is essential to the stability of the subsequent projective reconstruction. We designed an adaptive algorithm that combines three standard segmentation techniques (thresholding, luminance-weighted centroid computation, and circular Hough transform) to efficiently locate the centers of all circular shapes in a monochrome image without sacrificing accuracy. Fortunately, the bright circular pixel-blobs formed by the light reflected from our spherical markers are extremely easy to locate in an otherwise dark camera image by simple means of binary thresholding. The application of a non-destructive block-wise threshold (via locical AND-operations) yields image regions containing at least one bright ($>$ 50% luminance) pixel. Only those regions are further examined by subjecting them to a weighted grayscale centroid computation, which recovers the center location of an 8-connected blob to subpixel precision. We use a heuristic (based on the bounding rectangle and sum of pixel luminances) to select certain blobs for further analysis, if they are deemed likely to be composed of overlapping marker reprojections. In this case, a circular Hough transform [12] is applied to recover the center of every circular shape within a confined region-of-interest around the blob. Figure 5b shows a typical case of two markers, whose projections overlap and their recovered center locations.

5.3 Projective Reconstruction

Minimizing the square-sum of 2D image reprojection errors of a 3D point is widely accepted as the optimal formulation of the projective reconstruction problem [13] (also known as "L2-optimal triangulation"). When the correspondence between projected points in



0 10 20 30 40 50 60 70 80 90 (mm)
pairwise marker distances (with uncertainties)
correlation-score look-up table
compute distance matrix
look up correlation scores
accumulate scores (compute row sum)
apply threshold
Reconstructed 3D point cloud
Euclidean distance matrix
Correlation score matrix
Without correlation-score accumulation:
connect high-scoring candidate vertex-pairs with an undirected edge
select entries (3D points) with above-threshold values as candidate vertices
find maximal vertex-clique

Figure 7: Quadratic-time complexity reduction (top). Formulation of the model-fitting problem as maximum-clique search (bottom).

different views is initially unknown, as is the case with our system, is becomes necessary to formulate a feature-correlation strategy.

5.3.1 Multiple-View Feature Correlation

In order to reduce the combinatorial size of the correlation problem, we take advantage of an observation from epipolar geometry: A (very) large algebraic distance between a 2D point in one image and the epipolar line corresponding to a point in another image (also known as "epipolar distance", see Figure 5c) will also yield (very) large reprojection errors after L2-optimal triangulation. This allows us to safely discard every n-tuple of 2D features (from different views), in which the epipolar distance between any two points exceeds a certain large threshold (usually in the range of several pixels). Since the (squared) algebraic epipolar distance can be computed very efficiently, this results in a significant reduction of processing time. We modeled this optimization as search on a graph, whose vertices correspond to the 2D features in all cameras. If we connect every vertex-pair, whose epipolar distance does not exceed said threshold with an undirected edge, we can apply a maximum-clique algorithm [14] to obtain a drastically reduced set of correspondence candidates. Optimal correspondences are then chosen from the remaining candidates by ranking them according to their reprojection error (computed via triangulation).

5.3.2 Triangulation

Recovering a 3D point from multiple 2D correspondences in calibrated camera images is a well-studied problem with multiple straight-forward solutions (see [13] for a comprehensive survey). In our system, we chose the following approach: After obtaining an initial estimate via the standard "linear" SVD method, we perform bundle adjustment [15] with a Levenberg-Marquardt nonlinear least squares algorithm.

5.4 Model-Fitting

The basic idea behind our model-fitting algorithm is to formulate the underlying combinatorial problem in a way that allows for the application of a maximum-clique search. Prior to running the maximum-clique search (which is NP-hard), we apply a polynomial-time complexity-reduction heuristic that takes advantage of an important constraint: Since we are in control over how our targets are designed, we can construct them in a way that optimizes the effectiveness of the complexity-reduction step (see Section 5.4.3 for a detailed discussion).

5.4.1 Complexity Reduction

Our complexity-reduction technique is based on a form of geometric hashing. At startup, we compute the Euclidean marker-distance matrix for every trackable target (i.e. rigid constellation of 3D markers) and store its entries in a one-dimensional look-up table (see Figure 7, first row). Measurement uncertainties are assumed to follow a Gaussian distribution whose standard deviation is chosen to coincide with the tracker's absolute accuracy (see Section 7.3). We refer to the entries stored in this look-up table as *correlation scores*, because they represent the probability with which any given marker-pair (or more precisely, their separating distance) is part of the same target.

At runtime, we calculate the Euclidean distance between every pair of observed 3D markers for a given frame. For every entry in the Euclidean distance matrix (EDM), we look up its corresponding *correlation score* and store it in a separate Correlation score matrix (CSM). From the CSM, we compute a vector containing the accumulated correlation scores for every observed marker through row- or column-wise summation.

All of these operations can be carried out in a single parallel pass without any inter-thread coordination overhead. A visualization of the procedure is shown in Figure 7 (second row). As can be seen in the image, the CSM will contain mostly near-zero entries, indicating that the majority of marker-pairs do not belong to the same solution-tuple. In other words: the optimal solution to the model-fitting problem is highly unlikely to contain both of the two markers.

Depending on the number (n) of model-markers, we also expect every member of the solution-set to have an accumulated correlation score of at least $T_n = t * (n-1)$, where $t \in [0.5, 1]$. The easiest way to enforce this constraint is to apply threshold T_n to the accumulation-vector. Afterwards, every marker whose thresholded entry is zero can be eliminated as a candidate.

According to our observations, this elimination procedure will reliably reduce the number of candidate markers by up to 90% (in a real-life scenario, not degenerate test-cases), without being prone to false negatives.

5.4.2 Graph Search

After complexity-reduction, the remaining candidate markers are modeled as vertices in an initially unconnected graph. Every vertex-pair whose correlation score lies above threshold t is connected with an undirected edge. Figure 7 (bottom row) shows the resulting graph structure. As can be seen, the optimal solution to the model-fitting problem for a target of n markers is a vertex-clique of size n, which can be obtained through a maximum-clique search. We have to account for three possible outcomes of the search:

- **Single solution**: The graph has exactly one vertex-clique of size n. This is the ideal (and most common) case. No further processing is necessary.

- **Multiple solutions**: The graph has multiple vertex-cliques of size n. We choose the solution with the minimal square-sum of point-wise model-alignment errors as "optimal fit". If an estimate of the expected target pose is available (from a previous frame or predictive filter), we can also use linear and angular distances from the expected pose as ranking criteria.

- **No solutions**: The graph does not have any vertex-cliques of size n, most likely due to occlusions or the absence of the target from the working volume. If $n = 3$, the search is stopped, because we need at least three point-matches to estimate the target's orientation. Otherwise (if $n > 3$), we start a new search for model-substructures of size $(n-1)$, which involves adding previously discarded markers with an accumulation score above $T_{n-1} = t * (n-2)$ to the graph.

The complexity-reduction procedure described in Section 5.4.1 needs to run only once for each frame, whereas the graph search is repeated at least once per trackable target. We generally search for targets in descending order of their size (i.e. number of model-markers), but prioritize targets that were discovered in a recent frame over those for which no recent observation exist. Once the solution to the model-fitting problem has been found for a specific target, its corresponding markers are removed from the 3D point cloud before the search for the next target continues.

5.4.3 Target Design

The computational performance of our model-fitting algorithm depends in large parts on how many candidates (i.e. edges and vertices) the complexity-reduction heuristic is able to eliminate prior to the graph search. Problems arise when multiple targets are too "similar" to each other, or if a single target has a high degree of "self-similarity". In this context, we define "similarity" as the *smallest difference between pair-wise marker distances across all targets*. For the target pictured in Figure 8, the similarity-measure would be equal to the minimum entry in the matrix of pair-wise distances between all d_{ij}.

Suppose our targets were built to minimize this similarity-metric. It follows that we simultaneously minimized the probability with which any (random) marker-distance can at the same time occur inside two or more targets. A visual assessment of the similarity-metric would require examining the extent of overlap between multiple targets' correlation-score look-up tables, where little overlap would indicate a low similarity.

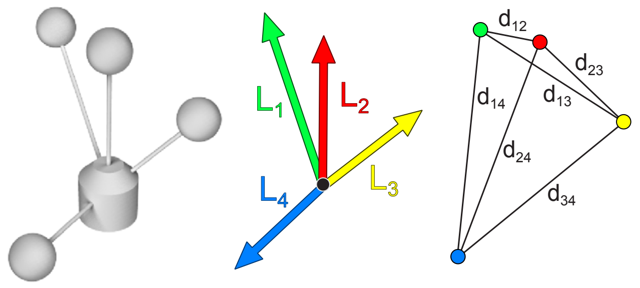


Figure 8: Target construction template.

Unfortunately, the way in which we build our targets does not allow for an arbitrary placement of markers. The position of a marker is determined by the length and angle of inclination of its mounting rod. In order to simplify the manufacturing process, we use the same drill-hole positions and angles for multiple base mounts ("target templates"), which leaves the variation of rod lengths as the only means of marker-placement. In Figure 8, those lengths are denoted $L_{1..4}$.

The problem of determining the optimal rod lengths for a set of targets amounts to a global maximization of the objective function $f(L_1, L_2, ..., L_n) =$ *minimum pair-wise distance* (d_i, d_j). Although there are are better global optimization techniques for this kind of problem, we achieved very good results with a simple genetic algorithm. In our experiment, the minimum difference of distances for a set of 6 targets with a total of 28 markers came out to 3 mm (with a 0.05-quantile of 5 mm).

5.5 3D-3D Pose Estimation

Several closed-form solutions exist for computing the 3D rigid transformation between two sets of point-correspondences. We chose to implement the approach developed by Arun et al. [16], which is based on computing the singular-value decomposition of a derived matrix and was found equivalent [17] to other closed-form pose estimators (e.g. Horn's quaternion-method [18]).

5.6 Predictive Filtering

In order to reduce pose jitter and to compensate for the tracker's intrinsic latency, we currently employ predictive double-exponential smoothing [19], which is a viable alternative to (non-extended) Kalman filtering. It is much faster, while performing only marginally worse.

5.7 Data-Delivery

For easy integration with existing VR/AR frameworks, we chose to base our system's server module the on the public-domain tracking middleware VRPN [6]. An extended VRPN protocol format lets client applications access additional information, such as the average marker reprojection errors or 2D blob feature coordinates.

6 Distributed Processing

When we wrote our tracking server software, we made two provisions for a future scenario in which several networked tracking systems would form an arrangement of multiple overlapping cells covering larger indoor working volumes:

- Every server instance can synchronously serialize its entire internal state (to a compressed buffer) at any point in the image-processing pipeline. The runtime penalty for such an operation is approximately 1 ms.

- Every non-temporary variable can be (synchronously) read or (asynchronously) written remotely via an XML-RPC communication protocol.

7 Preliminary Results

This section provides an initial assessment of our systems's accuracy, precision and latency for a room-sized four-camera setup. Currently, our tracking server application runs on GNU/Linux (using libdc1394) and Microsoft Windows (using either DirectShow or Point Grey's FlyCapture SDK for video acquisition). Unless otherwise indicated, all experiments were conducted on a system with two AMD Opteron 280 processors clocked at 2.4 GHz, running under SUSE Linux 10.1. The extrinsic camera calibration at the time of the experiments is the same as shown in Figure 6, yielding an approximate working volume of 2.75 x 2.75 x 2.25 m (about 17m^3).

7.1 Latency

To determine our system's intrinsic latency, we built a customized USB device (based on the FT2232D chip) to trigger the cameras from within our tracking software and used the kernel's internal high-resolution clock to time the average interval between the start of the shutter integration and the end of the 6-DOF pose estimation for the current frame. For this experiment, the shutter integration time was set to 1.5 ms. Figure 9 shows the latency measurements we obtained by moving a varying number of pre-calibrated 6-DOF targets through the tracker's working volume. The constellation depicted in Figure 5 represents one such test case.

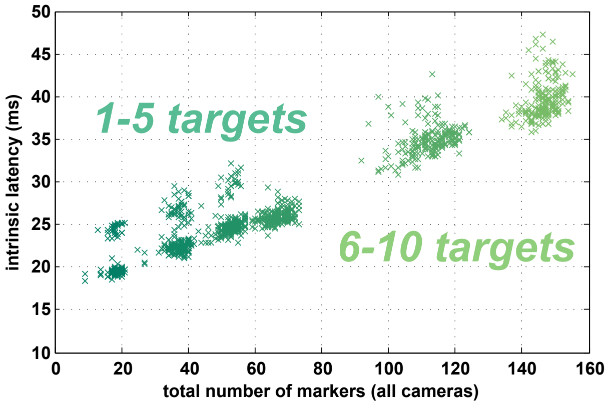


Figure 9: Intrinsic latency under varying computational load.

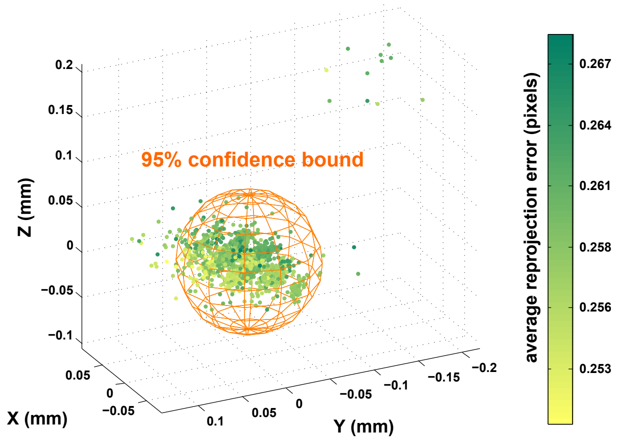


Figure 10: Distribution of position estimates for a static marker.

7.2 Static Jitter

Next, we placed a single marker near the center of the interaction volume and recorded its reconstructed 3D position for 36 seconds, resulting in 1800 measurements. The 3D scatter-plot in Figure 10 illustrates the spatial distribution of the reconstructed marker positions, whose RMS distance from their mean-normalized center (see Figure 11 for a histogram) equals 0.04 mm ($\sigma = 0.02$ mm). The 95% confidence radius of the distribution lies at 0.07 mm.

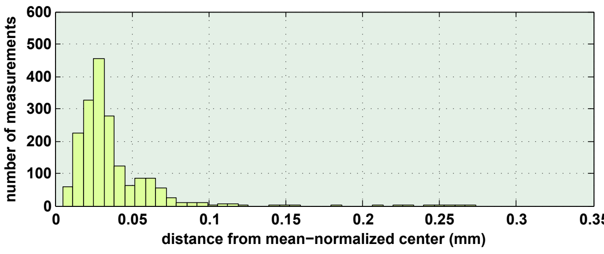


Figure 11: Static positional jitter of a single 3D marker.

Analogous to the previous experiment, we placed a pre-calibrated 5-marker target inside the tracking area and recorded its estimated 6-DOF pose for 2000 frames without any prediction or filtering. As depicted in Figure 12, the RMS angular distance from the mean-normalized pose was computed at 0.014° ($\sigma = 0.007°$).

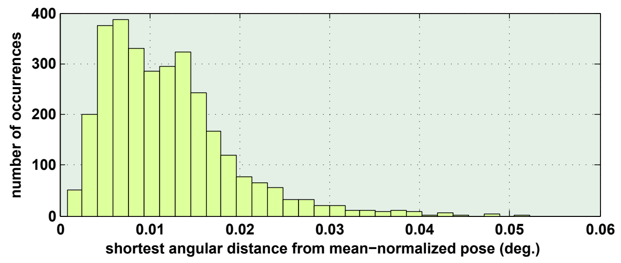


Figure 12: Static angular jitter of 5-marker target.

7.3 Accuracy

To assess the tracker's absolute accuracy, we rigidly attached two markers to the ends of a square-cut aluminum rod of 0.4 m length and surveyed their exact distance to submillimeter accuracy (using a theodolite). We then moved the rod through our tracking volume and recorded the variations in distance measurements between the two markers for 9700 subsequent frames. The resulting RMS distance error was measured at 5.3 mm ($\sigma = 5.2$ mm). The histogram in Figure 13 shows the distribution of distance errors.

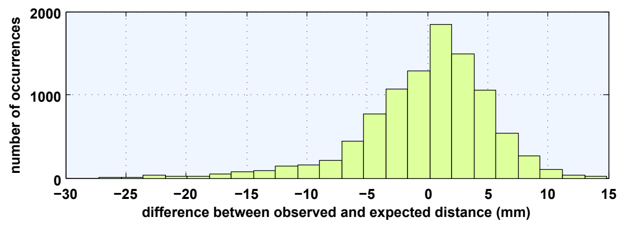


Figure 13: Accuracy of distance measurements.

8 Future Work

Ideas for the improvement of our system are manifold: Due to the significant cost-reduction, a range of new VR/AR applications becomes affordable and viable. Medium-term plans include the use of a larger number of cameras (> 20) in an arrangement of multiple overlapping cells to cover larger indoor volumes. We envision the use of multiple networked PCs with dynamic load-balancing, because there are architectural limits to the number of cameras that can be attached to a single PC. However, on a recently acquired workstation equipped with two Intel Xeon DP X5355 processors (a total of 8 cores), the mainboard theoretically allows to attach 36 cameras (with specifications as described in 3.1) using 2 on-board and 4 quad-channel add-on 1394-controllers.

Additionally, we plan to investigate whether general-purpose computing on modern GPUs would allow us to increase the performance of core algorithms.

9 Conclusion

This paper laid out the hardware and software-design for a low-cost infrared-optical pose-tracking system. Our goal was to minimize the cost of the system without sacrificing speed or accuracy. It is our expressed intention to make our system available not only to other members of the academic community, but also to artists, game designers and small commercial application developers with an interest in immersive Virtual and Augmented Reality.

Acknowledgements

This work was funded in part by the Austrian Science Fund FWF project P19265. The authors wish to extend a special note of appreciation to Christian Breiteneder for his support throughout the project.

References

[1] G. Welch, G. Bishop, L. Vicci, S. Brumback, K. Keller, and D. Colucci. The hiball tracker: high-performance wide-area tracking for virtual and augmented environments. In *VRST '99: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 1–10, New York, NY, USA, 1999. ACM Press.

[2] E. Foxlin and L. Naimark. Vis-tracker: A wearable vision-inertial self-tracker. In *Proceedings of the IEEE Conference on VR*, 2003.

[3] M. Ribo. State of the art report on optical tracking. Technical Report TR-VRVis-2001-025, 2001.

[4] H. Kaufmann and D. Schmalstieg. Mathematics and geometry education with collaborative augmented reality. *Computers & Graphics*, 27(3):339–345, Jun 2003.

[5] K. Mania, B. D. Adelstein, S. R. Ellis, and M. I. Hill. Perceptual sensitivity to head tracking latency in virtual environments with varying degrees of scene complexity. In *APGV '04: Proceedings of the 1st Symposium on Applied perception in graphics and visualization*, pages 39–47, New York, NY, USA, 2004.

[6] R. M. Taylor, T. C. Hudson, A. Seeger, H. Weber, J. Juliano, and A. T. Helser. VRPN: a device-independent, network-transparent VR peripheral system. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, pages 55–61, 2001.

[7] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition, 2004.

[8] O. Faugeras, Q. T. Luong, and T. Papadopoulou. *The Geometry of Multiple Images: The Laws That Govern The Formation of Images of A Scene and Some of Their Applications*. MIT Press, Cambridge, MA, USA, 2001.

[9] X. Chen, J. Davis, and P. Slusallek. Wide area camera calibration using virtual calibration objects. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 520–527, 2000.

[10] J. Heikkilä and O. Silvén. A four-step camera calibration procedure with implicit image correction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1106–1112, 1997.

[11] T. Svoboda, D. Martinec, and T. Pajdla. A convenient multi-camera self-calibration for virtual environments. *PRESENCE: Teleoperators and Virtual Environments*, 14(4):407–422, August 2005.

[12] C. Kimme, D. Balard, and J. Sklansky. Finding circles by an array of accumulators. *Commun. ACM*, 18(2):120–122, 1975.

[13] R. Hartley and P. Sturm. Triangulation. *Journal of Computer Vision and Image Understanding*, 68(2):146–157, 1997.

[14] P. R. J. Östergård. A fast algorithm for the maximum clique problem. *Journal of Discrete Applied Mathematics*, 120(1-3):197–207, 2002.

[15] B. Triggs, P. McLauchlan, R. Hartley, and A. Fitzgibbon. Bundle adjustment – A modern synthesis. In W. Triggs, A. Zisserman, and R. Szeliski, editors, *Vision Algorithms: Theory and Practice*, LNCS, pages 298–375. Springer Verlag, 2000.

[16] K. S. Arun, T. S. Huang, and S. D. Blostein. Least-squares fitting of two 3-D point sets. *IEEE Transactions On Pattern Analysis And Machine Intelligence*, 9:699–700, 1987.

[17] D. W. Eggert, A. Lorusso, and R. B. Fisher. Estimating 3-d rigid body transformations: a comparison of four major algorithms. *Journal of Machine Vision and Applications*, 9(5-6):272–290, 1997.

[18] B. K. P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America*, 4(4):629–642, April 1987.

[19] J. LaViola. Double exponential smoothing: an alternative to kalman filter-based predictive tracking. In *Proceedings of the Workshop on Virtual Environments (EGVE '03)*, pages 199–206, New York, NY, USA, 2003.