A Flexible Software Architecture for Hybrid Tracking

Miguel Ribo*
*Christian Doppler Laboratory for Automotive Measurement Research
Graz University of Technology
Schiessstattg.14B, A-8010 Graz, Austria
e-mail: ribo@emt.tugraz.at*

Markus Brandner and Axel Pinz
*Institute of Electrical Measurement and Measurement Signal Processing
Graz University of Technology
Schiessstattg.14B, A-8010 Graz, Austria
e-mail: brandner@emt.tugraz.at,
pinz@emt.tugraz.at*

Received 4 November 2003; accepted 4 November 2003

Fusion of vision-based and inertial pose estimation has many high-potential applications in navigation, robotics, and augmented reality. Our research aims at the development of a fully mobile, completely self-contained tracking system, that is able to estimate sensor motion from known 3D scene structure. This requires a highly modular and scalable software architecture for algorithm design and testing. As the main contribution of this paper, we discuss the design of our hybrid tracker and emphasize important features: scalability, code reusability, and testing facilities. In addition, we present a mobile augmented reality application, and several first experiments with a fully mobile vision-inertial sensor head. Our hybrid tracking system is not only capable of real-time performance, but can also be used for offline analysis of tracker performance, comparison with ground truth, and evaluation of several pose estimation and information fusion algorithms.

*To whom all correspondence should be addressed.
Contract grant sponsor: Christian Doppler Laboratory for Automotive Measurement Research; Contract grant sponsor: Mobile Collaborative Augmented Reality project; Contract grant number: P14470-INF Contract grant sponsor: Tracking with Smart Sensors project; Contract grant number: P15748; Contract grant sponsor: Visual Active Memory Processes and Interactive REtrieval project; Contract grant number: IST-2001-34401.

1. INTRODUCTION

Fusion of vision-based and inertial pose estimation will play an important role in future high-potential mobile applications. Sensors are complementary, and no external signals are required. Forthcoming technological developments will lead to very compact mobile devices that will be used in navigation, augmented reality, and mobile communication, as well as on autonomous mobile platforms.

Many individual system components have to interact in real-time for a fully autonomous vision-inertial pose computation. Success has been reported, on hybrid and even on purely vision-based tracking, for cases where a carefully prepared environment can be assumed,[1] when the environment is simple and can be well controlled,[2] when there is a limited amount of sensor motion or a limited number of degrees of freedom,[3] when few objects are tracked,[4] or when many artificial landmarks can be placed in the scene.[5,6] As soon as natural landmarks have to be tracked, and when the scene gets more complex (changes in illumination, cluttered scene, occlusions, other moving objects obscuring the stationary background, outdoor applications) it is our experience that currently existing tracking algorithms tend to become more fragile and to break down every now and then. In summary, we have two reasons for the development of a new, flexible software architecture for hybrid real-time tracking:

Handling of many individual system components: CCD cameras, CMOS cameras, several camera interfaces (framegrabbers, firewire, USB), inertial sensors, additional sensors (compass, GPS), 2D image feature extraction, 2D prediction, 3D pose from 2D features, 3D pose from stereo, Kalman filter, inertial sensor raw data gathering, inertial pose computation, time stamps for individual sensor readings, target selection, etc. For many of these components, several different sensors, algorithms, and implementations may be available. Individual components have to be compared, replaced, and tuned. Other components, which are not required during real-time operation, but should be also integrated into the framework, are sensor calibration, pose initialization, logging of time stamps, and synchronized sensor raw data stream for offline analysis.

Testing, debugging, error handling: Whenever the real-time operation breaks down, it should be possible to do a detailed offline analysis of the case. Besides the data logging mode listed above, we require the access to all the individual modules to isolate a problem and to study the interaction between modules. It should be possible to process logged and time-stamped data in the identical manner as for real-time operation. For rapid prototyping, it would be desirable to have identical functionality, but slower execution times, on a development platform, e.g., MATLAB.

1.1. Related Work

Purely vision-based approaches to tracking and self localization have been reported in the literature. The XVision system[7] was designed to allow for fast and easy implementation of vision-based tracking experiments. It consists of an object oriented class design which is extendable and reusable. Concerning real-time tracking, XVision has introduced a windowing method to control processing time spent during feature computation. The V4R system as part of the ROBVISION project[2] is able to track a camera head's pose given a 3D CAD model. Tomasi and Kanade[8] describe a feature tracker based on examining the minimum eigenvalue of each 2 by 2 gradient matrix. Multi-resolution tracking allows for even large displacements between images. These systems have been reported to perform well within the environments they have been designed for. However, they are subjected to the limitations of vision based tracking.

Hybrid systems using vision and some sort of complementary sensors would allow us to alleviate the shortcomings of a vision-based tracking system. Azuma et al.[9] present a hybrid system combining gyro, compass, and tilt sensor. You et al.[10] report about an inertial/optical hybrid using a two-channel complementary motion filter. Two extended Kalman filters (EKF) sharing a common state prediction module are used. Jung and Taylor[11] present a system capable of estimating the affine trajectory given a structure from motion algorithm applied to a certain number of key frames and an inertial sensor. Naimark and Foxlin[6] use a system consisting of a vision-based fiducial tracker and an inertial sensor. In a Kalman filter based algorithm they are able to both fuse the sensor information and to incorporate a map-building process.

2. HYBRID TRACKER SOFTWARE ARCHITECTURE

From the previous section it is clear that hybrid tracking systems are needed to provide stable and precise pose readings in environments different from a controlled laboratory or indoor scene. The requirement of being able to rapidly prototype tracking applications and to conduct experiments led us to the design of a flexible software architecture for hybrid tracking applications. HT (*hybrid tracker*) is a set of object oriented C++ classes that have been designed using UML. HT's design was focused on the following key requirements:

Sensor classes: HT is capable of interfacing a variety of traditional camera interfaces such as firewire, USB, and selected framegrabbers. In addition, the design includes features to deal with nontraditional sensors such as CMOS cameras utilizing their random pixel-access capabilities and logarithmic pixel response. Apart from vision sensors, HT offers interfaces to inertial sources such as accelerometers and gyroscopes.

Scalability, Flexibility: Different applications require different configurations of vision and inertial sensors. HT's class structure is flexible enough to provide run-time configuration. Thus, setup information such as sensor calibration data, sensor configuration, and tracker configuration can be set at run-time. HT supports XML configuration files to aid this task.

Reusability: HT is planned as a long-term project with ongoing support by members of our research group. Thus, the design supports the separation of different modules into libraries which in turn can be maintained and developed further individually. The reusability of source code is a major design goal.

Complexity: The basic functionality offered by HT is designed to be lightweight in terms of computational complexity. The HT framework provides interfaces between different modules but does not incur unnecessary CPU power.

2.1. The HT Environment

Figure 1 gives an overview on the environment of the proposed HT design. The output of the tracking application is a stream of pose information (either self-pose of vision sensors and/or pose of distinct objects within a vision sensor's view). At system startup HT loads the sensor configuration and builds up the tracker class hierarchy. During the tracking process, a constant exchange of target data and corresponding features between the target selector and the tracker takes place (see discussion below for details). An administrator interface provides online status information, logging, and parameter tuning.

2.2. Hybrid Tracking in HT

The basic entity within HT is the *HybridUnit* (see Figure 2) which is built up of a *VisionUnit* and an *InertialUnit*. Each unit is assigned a coordinate system the relative poses of which are determined during a calibration process. A sensor reading of an individual vision or inertial sensor within a given coordinate system can be transformed to the coordinate system of the base class by means of appropriate member functions.

In order to reflect the physical properties of different inertial sources such as accelerometers (i.e., sensor pose w.r.t. the *InertialUnit*'s coordinate system) an *InertialUnit* is modeled as aggregation of *InertialSensors*. Similarly, a *VisionUnit* consists of one (mono-view) or more (stereo- and multiview, respectively) *VisionSensors*. A *VisionSensor* is in control of a physical vision sensor (e.g., camera plus grabber). For every feature type (e.g., blobs, fiducials, corners) a separate *FetaureTrackerController* is instantiated which is responsible for tracking of features of the given type. The *VisionSensor* controls the camera access and

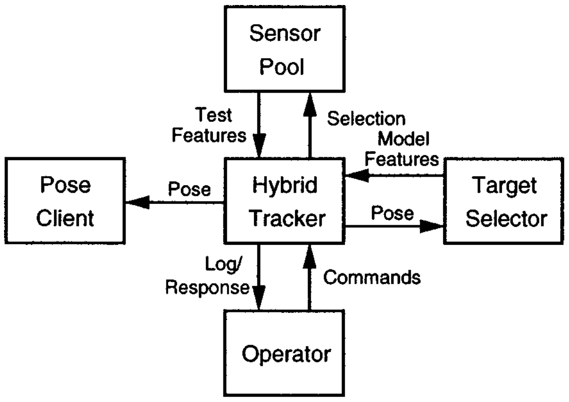

Figure 1. Overview of the hybrid tracking system.

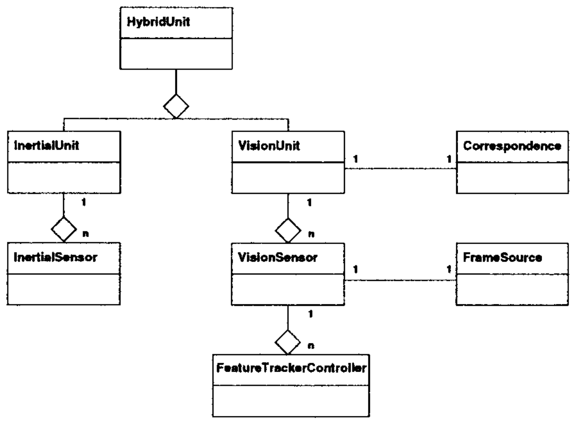

Figure 2. Subtree of HT's UML class diagrams. A *HybridUnit* consists of an aggregation of an *InertialUnit* and a *VisionUnit*. The correspondence search between model and test features takes place at the *VisionUnit* level.

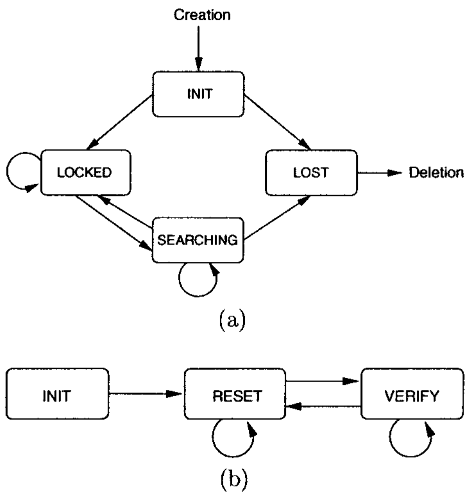

Figure 3. Possible states within HT: (a) depending on the result of the last tracker run, each feature is assigned a distinct state. (b) Feature states are propagated upwards within the class hierarchy and result in different states of the correspondence module.

can, therefore, avoid multiple readouts of a given region through different calls of the assigned feature trackers.

2.3. Treating Complexity: Target Selection

For any vision-based tracking algorithm that is required to autonomously *initialize* and *recover* from feature loss, the complexity of underlying search-spaces is a limiting factor for real-time operation. The HT design defeats its real-time capabilities via two distinct approaches:

Stateful Tracker Design: Within HT every visual feature involved in the tracking process carries an associated feature state. Figure 3(a) depicts the possible states and transitions of a single feature in 2D. *Init*-features are used to recover distinct areas of the input image (i.e., after feature loss). They are also used to identify potential search areas that are delivered by the correspondence module to enhance tracking performance (e.g., to avoid ill-posed feature configurations prior to computing the pose). During normal tracking operation a feature is either *locked*, i.e., has been successfully found within the current frame, or *searching*, i.e., was found in previous frames but has been lost since then. The feature state is propagated through HT's class hierarchy resulting in states for features in 3D (prior to the correspondence module) and states of corresponding model and test feature pairs (after the correspondence module). Given states of corresponding features the correspondence module itself is state driven. If enough model features and test features are found to correspond during the current frame, the correspondence module switches from the *reset* to the *verify* state. Thus, during further tracking frames no new correspondence search is initialized, rather the current correspondence is verified. This process is of considerably lower complexity.

Target Selection: In order to make the HT design as independent of its field of application as possible we decided to separate the tracking process from the target selection process (see Figure 1). Each vision sensor in the HT application is assigned a *Target Selector* which at a constant (but lower than framerate) rate delivers the set of features that are visible to the corresponding sensor. This process is similar to delivering a CAD model to the tracker as is done by the V4R application. However, the target selection not only takes into account the pose of the vision system but also the *context* of the application running the tracker. In the case of an augmented reality system this could be the introduction or removal of a new interaction device (such as a handheld PC) to the field of view of the vision sensor. In addition, HT provides the possibility to deliver features to the target selector that are not already stored within the database (map building) and to refine feature attributes (e.g., pose w.r.t. world coordinates).

2.4. Sensor Fusion

In HT data fusion of vision and inertial sensors takes places in the *HybridUnit*. The design of our tracking system does not restrict the fusion algorithm to be of any distinct type. Thus, the *HybridUnit* serves as a pose source to both the *InertialUnit* and the *VisionUnit*. This gives HT the possibility to correct sensor offsets in the case of inertial sensors and to utilize the inertial pose during a prediction step for the vision-based tracker.

2.5. Debugging HT

A number of test applications (see Section 4) have been implemented using the proposed HT design. However, HT is still under development which makes a proper debugging interface extremely important. The design of HT allows us to both process live input data (inertial sensor data and vision data) and to play-back previously stored experiments. Therefore, main classes in HT provide debugging input and output interfaces (e.g., video output, inertial data stream, pose streams, …).

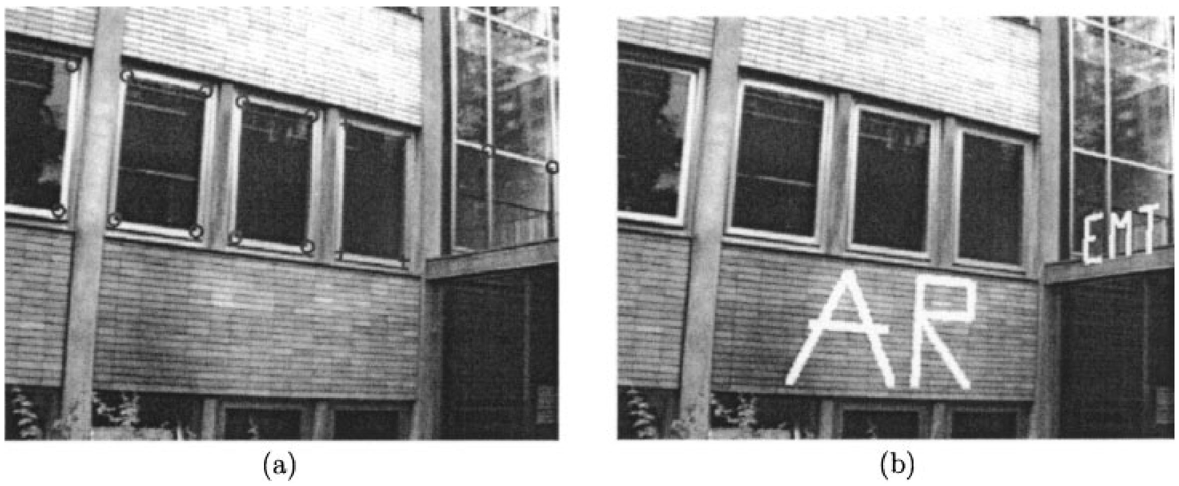

Figure 4. Processing of a simple facade. (a) Corner features tracked by the HT application, (b) augmented view of the scene.

2.6. Real-Time Tracking

The HT project is implemented on a standard Linux platform. We have run first experiments using a real-time kernel (Linux Real-Time Application Interface—RTAI). This platform enables us to run distinct parts of the HT application as a *real-time* process with guaranteed response times and latencies. We are currently investigating the use of *AnyTime* algorithms and a specialized tracker scheduling policy which prevents the vision-based tracker from spending too much time processing features that do not contribute to the tracking accuracy.

3. MOBILE AUGMENTED REALITY APPLICATION

The major current application of our tracking system is in mobile augmented reality (AR). While many solutions have been presented for indoor tracking, limited user motion, and prepared rooms, fully mobile tracking of natural landmarks (indoor and outdoor) still poses a challenge for current tracking systems. In ref. 12 we describe initial experiments with our hybrid tracker for a simple (backyard of an office building) and for a complex (historic city center) urban scene. The hybrid tracker works for a 13.5 s sequence in the simple scene, but fails for the complex scene. Figure 4 shows a frame from the backyard scene, the extracted ''promising features to track'' (mostly corners of the windows), and the resulting simple augmentation of the scene as it is perceived by the user. Figure 5 shows the current state of our mobile AR setup, which consists of two subsystems. The *tracking subsystem* runs on a single-board PC with interfaces to a camera and an inertial tracker, which are mounted to the user's helmet. Hybrid tracking delivers six degrees of freedom of head pose at rates of approximately 100 Hz. The *visualization subsystem* consists of a laptop with 3D graphics acceleration and a see-through head-mounted-display (HMD). The HMD delivers virtual 3D graphics as stereo video streams which are blended with the user's natural perception of the real scene using semitransparent mirrors. An additional firewire webcam is mounted at the front of the helmet and connected to the laptop. This camera can be used (but was not used in our current experiments) for 3D human–computer interaction, e.g., tracking of an interaction pad and pen, as described in ref. 13. There are many exciting applications of mobile outdoor AR: architectural visualization, city guide, maintenance support, navigation, rescue and emergency operations, to name just a few. Most of these applications require a reliable continuous tracking with high accuracy, low jitter, and no lag. Tracking requirements become even more stringent for multiuser AR scenarios, where several mobile users require an augmented perception which is consistent in space and time.

4. FIRST EXPERIMENTS AND RESULTS

We employ point-based visual pose estimation, so that point-like visual features have to be extracted from the individual 2D image frames. We have experimented with several visual targets (from infrared

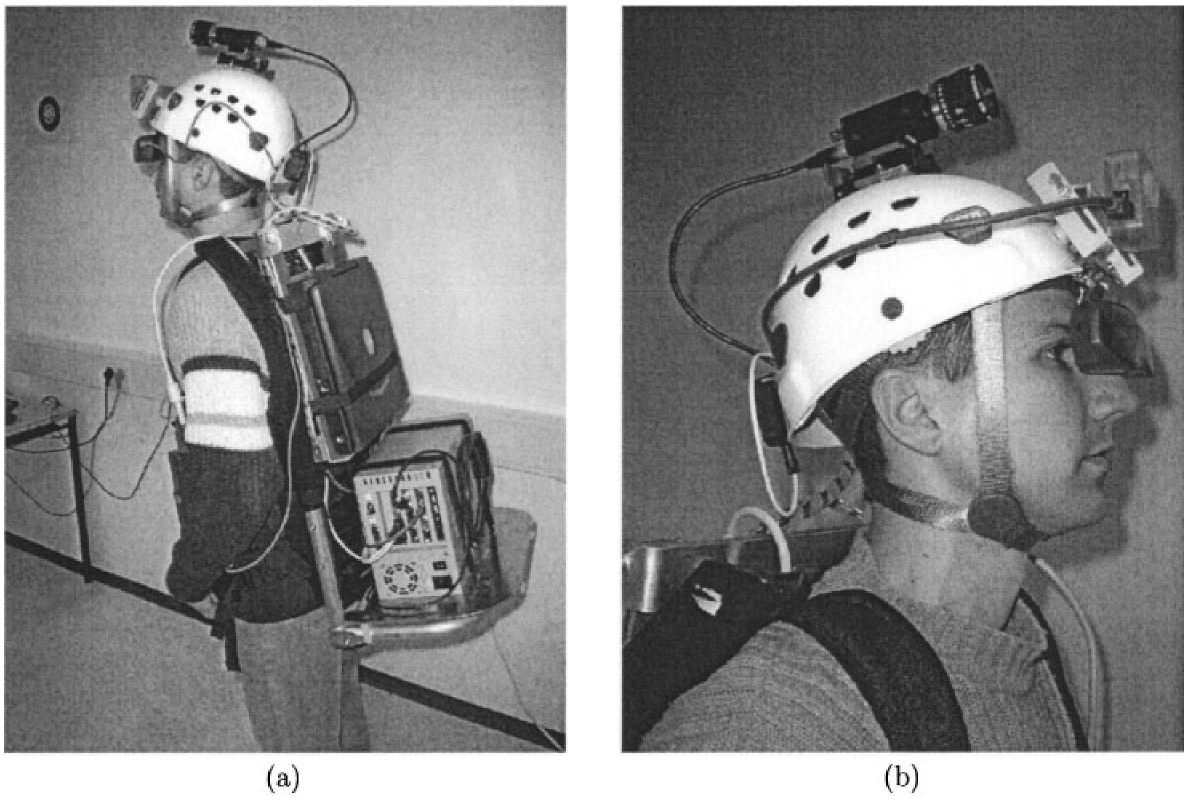

Figure 5. AR gear prototype. (a) Backpack with laptop and tracking PC. (b) Close-up of the helmet with inertial sensor (back), camera, and head mounted display (HMD).

LED blobs to "natural landmarks" like the corners of the windows in Figure 4). Figure 6 shows the type of features which were used for the experiments described below: Figure 6(a) shows three targets, each of them carrying seven individual simple corner features. The arrangement of these corners helps in identifying each target by a combination of two cross-ratios (see ref. 14). The design of our fiducial features [Figure 6(b)] that carry the position of the blob in 2D and a fiducial code for unmistakable identification was inspired by Naimark and Foxlin[6] but has been adapted to provide more robust recognition under strong perspective distortion. Furthermore, our detector has been extended to deal with the logarithmic characteristic of certain types of CMOS cameras (e.g., Fuga 1000 sensor).

4.1. Vision-Based Pose Estimation

After successful 2D localization and identification of point features on a target object, point-based pose estimation can be performed, which is commonly done by solving the perspective n-point problem (PnP).[15] A perspective vision model that assumes the projection of a 3D object onto a 2D image plane through a pinhole camera model is used. As a result, the minimum value of n that produces a finite number of solutions is three, although up to four solutions are possible. Four coplanar and non-collinear points give a unique solution. Four or five non-coplanar and non-collinear points give two solutions. For n greater than five non-collinear points, the result is unique and consists of an overdetermined set that can be solved using least squared error techniques. In general, as n increases, the accuracy of the pose estimation increases.

The pose algorithm used in our system is the one proposed by Lu et al.[16] The authors show that the pose estimation problem can be formulated as a metric error minimization based on collinearity in object space. As a result, they are able to derive an algorithm that operates by successively improving an estimate of the rotation (of the pose) and then estimates an associated translation. The proposed method is iterative, but it is efficient and robust. Moreover, when the system computes a valid pose, we use this information as prior initialization for the next pose

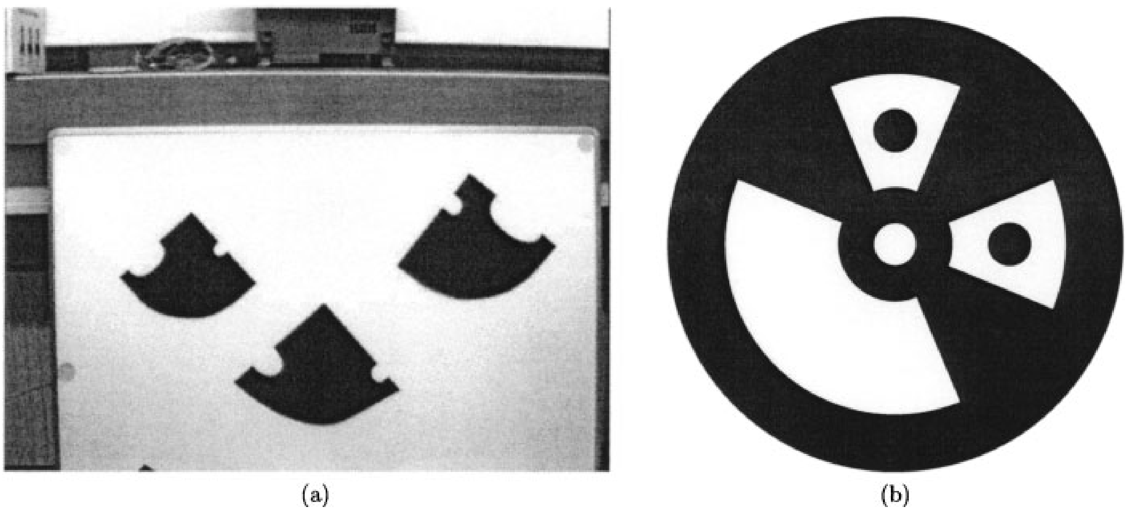
Figure 6. Different types of features used during our experiments: (a) corner features and (b) fiducial features.

computation. In this way, we improve the pose's accuracy and we reduce the computational costs.

For the evaluation of the system's self-localization abilities the results of the inside-out tracker (vision-based) were compared to the results obtained by an outside-in tracker[1] with the following experimental setup (see Figure 7): The outside-in tracker consisting of an IR sensitive stereo camera pair and IR light sources determines the pose of the camera by tracking IR reflectors mounted on the camera. The camera performs inside-out tracking and estimates its pose relative to the corner targets. The comparison of the trajectories recorded by the outside-in and the inside-out system is shown in Figure 8. Though neither corner targets nor IR targets are measured at high precision, the mean absolute error is approximately 1 cm which results in an absolute value of the error relative to the distance below 0.5%.

Beside the PnP-based pose estimation following Lu et al.[16] we have also implemented a combination of Fischler and Bolles[15] and Lu et al.[16] (see ref. 14), and the method proposed by Ansar and Daniilidis.[17] Moreover, thanks to the modularity and the flexibility of HT, all implemented pose estimation algorithms can be compared (e.g., speed versus accuracy) in a straightforward and intuitive manner.

4.2. Pose Estimation with Inertial Sensors

Inertial tracking systems compute the relative change in position $\Delta\mathbf{p_n}$ and orientation $\Delta\mathbf{q_n}$ from the appearing acceleration and angular velocity in the moving target ("moving-frame") with respect to an inertial (acceleration free) reference coordinate system ("reference-frame"). With a known absolute start position $\mathbf{p_0}$ and start orientation $\mathbf{q_0}$ the actual absolute position $\mathbf{p_n}$ and orientation $\mathbf{q_n}$ is further determined (see Figure 9). The parameter set $(\mathbf{p_n}, \mathbf{q_n})$ is called the pose at time n.

We have developed our own custom inertial tracker which provides higher update rate and better synchronization, but we also use the commercially available Xsens MT9 tracker, which provides updates at 100 Hz using a serial interface. Both inertial trackers are supported in HT, and can be used in a transparent way (i.e., loading an appropriate XML configuration file).

In the experimental setup our custom inertial tracker was used in conjunction with a vision-based

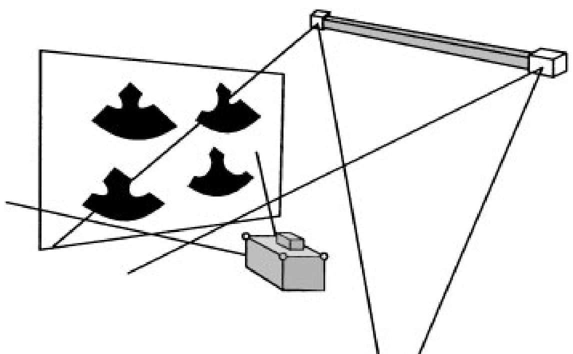
Figure 7. Sketch of the experimental setup: comparison of the outside-in versus hybrid inside-out tracker.

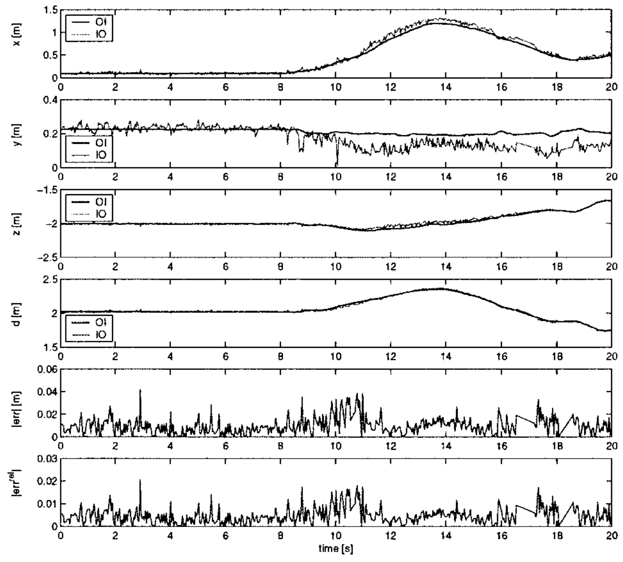

Figure 8. Comparison of inside-out (IO, thin line) and outside-in tracker (OI, thick line): Coordinates (corner target coordinate system) of the outside-in camera estimated from inside-out and outside-in tracking system x,y,z, distance from target to camera d, absolute and relative error of distance d using outside-in as reference data $|err|$,$|err^{rel}|$.

tracker that served as a ground-truth at a rate of 30 Hz. The inertial tracker rate was 500 Hz with drift updates at a rate of 10 Hz. Translation and rotation experiments were done. The translation experiments were carried out manually while the rotation experiments were performed with a pan-tilt unit that allows for a controlled simultaneous independent rotation in

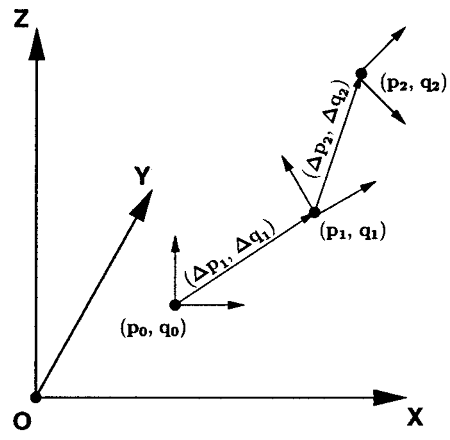

Figure 9. Inertial tracking process. The inertial tracker computes the relative changes in position $\Delta \mathbf{p}_n$ and orientation $\Delta \mathbf{q}_n$, and determines the actual pose $(\mathbf{p}_n,\mathbf{q}_n)$ relative to start configuration $(\mathbf{p}_0,\mathbf{q}_0)$.

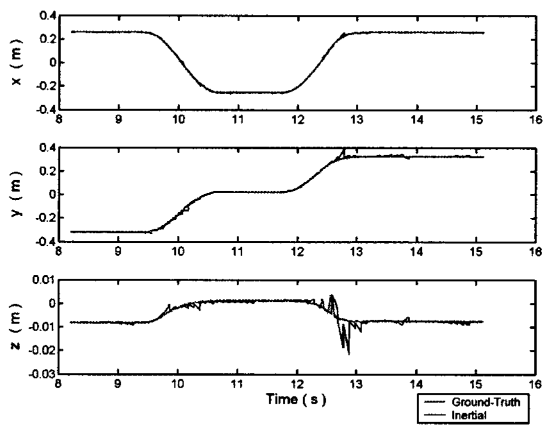

Figure 10. 2D Translation. Simultaneous motion in x- and y-direction.

two axes. Figures 10 and 11 show results from practical translation and rotation experiments, respectively.

Figure 10 depicts results from a translation experiment of a simultaneous movement in the x-direction and the y-direction. The position standard deviation is (4.1 mm, 9.4 mm, 1.8 mm). Figure 11 depicts results from a rotation experiment of a simultaneous 30° rotation around the y-axis and 90° rotation around the z-axis with an angular velocity of 50° s^{-1} in quaternion representation. The orientation standard deviation is (0.29°,0.29°,0.31°).

4.3. Fusion of Pose Estimates

Vision-based tracking often lacks fast motion tracking performance due to the rapid change of visual

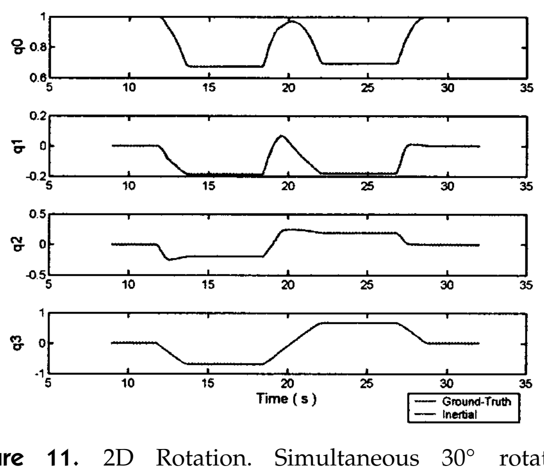

Figure 11. 2D Rotation. Simultaneous 30° rotation around the y-axis and 90° rotation around the z-axis with an angular velocity of 50 °s^{-1}.

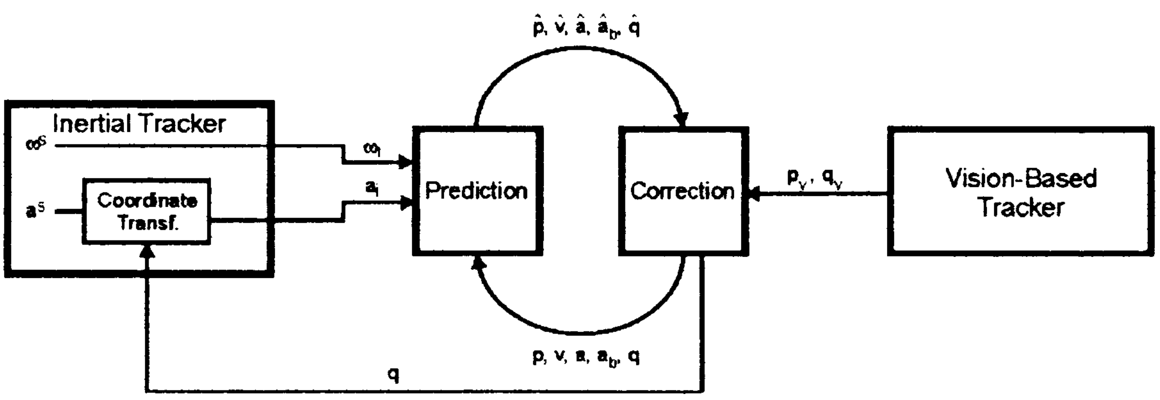

Figure 12. Fusion of vision-based and inertial tracker with a modified extended Kalman filter.

content, especially with fast rotations of the camera. Inertial tracking itself is fast but lacks long-term stability due to sensor noise and drift. This is in contrast to high precision and long-term stability of vision-based tracking with slow motion. Thus, sensor fusion is used to exploit the complementary properties of both technologies and to compensate for their respective drawbacks.

The sensor fusion process integrates an extended Kalman filter that operates in a predictor-corrector manner (see Figure 12). The state vector $\mathbf{x}$ maintains position $\mathbf{p}$, velocity $\mathbf{v}$, acceleration $\mathbf{a}$, acceleration sensor bias $\mathbf{a}_b$, and orientation $\mathbf{q}$ and is updated by sensor measurements ω_m, $\mathbf{a}_m$ from the inertial system and $\mathbf{p}_m$, $\mathbf{q}_m$ from the vision-based system. The corrected orientation $\mathbf{q}$ is also used for the necessary coordinate transformation of the acceleration since position computation takes place in the world coordinate system and acceleration is measured in the moving coordinate system. Operating inertial and vision-based trackers on the same CPU allows for simple temporal synchronization between both tracking systems. The experimental result illustrates the capabilities of our hybrid tracking system. It is based on the fusion between our Vision-based tracker and our inertial tracking system (see Section 4.2). The experiment was done by a person wearing our mobile AR setup (see Figure 5) and facing the target depicted in Figure 6(a). Our reference coordinate system is centered in the upper left corner of the white board, with the x-direction pointing right, the y-direction pointing down, and the z-direction pointing backward. The movement of the user is almost in the xz-plane (from the left to the right), and he mainly rotates his head in the y- and z-directions. Moreover, the scene (i.e., white board+user) is observed by our outside-in tracker which gives us the ground-truth of

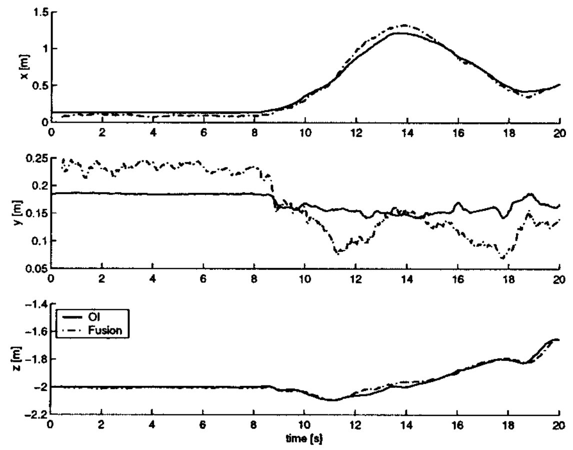

Figure 13. 3D position: Hybrid system (fusion) versus ground truth (outside-in, OI).

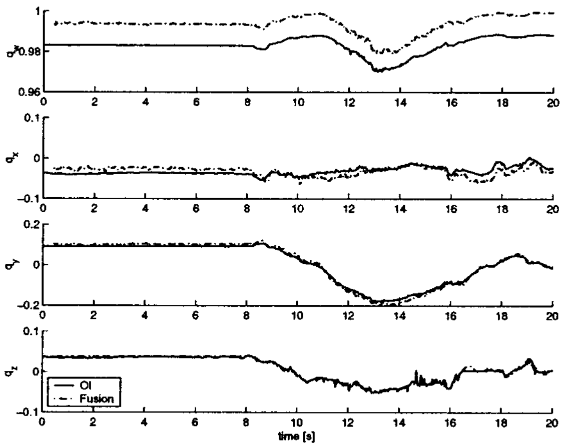

Figure 14. 3D rotation: Hybrid system (fusion) versus ground truth (outside-in, OI).

the user's 6 d.o.f. Figure 13 shows the user's 3D position, while Figure 14 depicts the four quaternion components of the user's 3D rotation. Notice the different scaling, which makes, e.g., a jitter of 5 cm in the y-direction (Figure 13) and an offset of 1.7°, over the value q_w (Figure 14) visible.

5. DISCUSSION AND OUTLOOK

We have presented a software architecture for hybrid tracking applications. HT, our hybrid tracking environment provides a scalable class structure for any combination of vision and inertial sensors. The aim of the design was reusability and flexibility of the tracking application. HT is configurable through XML configuration files and is able to support a variety of debugging interfaces used during implementation of new applications. The proposed design strictly separates the tracking process from the context dependent model feature selection. Thus, HT is independent of the pose client application (e.g., an augmented reality scenario with user interaction). The design inherently supports multisensor fusion and does not restrict the fusion algorithm to be of any distinct type. HT is therefore well suited for experimental as well as production hybrid tracking applications.

We have shown the feasibility of the proposed structure to act as an experimental platform for AR applications. Due to the modular design a comparison of different pose estimation techniques was easily implemented. Furthermore, we have been able to compare different fusion algorithms. HT currently supports our own custom-built inertial sensor as well as a commercial product. Again, the open design allows us to extend the range of supported hardware in a minimum time.

Still there remain open issues for future work: HT's prime application was targeted at AR scenarios. We plan to extend HT's interface toward an integration with the *OpenTracker*[18] platform. We are currently in the process of extending HT's real-time capabilities to achieve *hard* real-time performance.

REFERENCES

1. M. Ribo, A. Pinz, and A.L. Fuhrmann, A new optical tracking system for virtual and augmented reality applications, in Proc IEEE Instrumentation and Measurement Technology Conf, IMTC 2001, Budapest, Hungary, May 2001, vol. 3, pp. 1932–1936.
2. W. Ponweiser, M. Ayromlou, M. Vincze, C. Beltran, O. Madsen, and A. Gasteratos, Robvision—vision based navigation for mobile robots, IEEE Int Conf on Multisensor Fusion and Integration for Intelligent Systems (MFI), Baden-Baden, 2001, pp. 109–114.
3. K. Satoh, M. Anabuki, H. Yamamoto, and H. Tamura, A hybrid registration method for outdoor augmented reality, in Int Symposium on Augmented Reality, 2001, pp. 67–76.
4. V. Ferrari, T. Tuytelaars, and L. Van Gool, Markerless augmented reality with a real-time affine region tracker, in Proc IEEE and ACM Intl Symposium on Augmented Reality, October 2001, vol. I, pp. 87–96.
5. M. Billinghurst, H. Kato, and I. Poupyrev, The MagicBook: a transitional AR interface, Comput Graph, 25 (2001), 745–753.
6. L. Naimark and E. Foxlin, Circular data matrix fiducial system and robust image processing for a wearable vision-inertial self-tracker, in IEEE ISMAR, Darmstadt, Germany, 2001.
7. G.D. Hager and K. Toyama, XVision: A portable substrate for real-time vision applications, Comput Vision Image Understand 69:(1) (1998), 23–37.
8. C. Tomasi and T. Kanade, Detection and tracking of point features, Tech. Rep. CMU-CS-91-132, Carnegie Mellon University, April 1991.
9. R. Azuma, B. Hoff, and H. Sarfaty, A motionstabilized outdoor augmented reality system, in Proc IEEE VR'99, Houston, TX, March 1999, pp. 252–259.
10. S. You and U. Neumann, Fusion of vision and gyro tracking for robust augmented reality registration, in IEEE Conf on Virtual Reality, Yokohama, Japan, March 2001, pp. 71–78.
11. S.-H. Jung and C.J. Taylor, Camera trajectory estimation using inertial sensor measurements and structure from motion results, in IEEE CVPR, Kauai, HI, December 2001, vol. 2, pp. 732–727.
12. M. Ribo, H. Ganster, M. Brandner, P. Lang, C. Stock, and A. Pinz, Hybrid tracking for outdoor AR applications, IEEE Comput Graph Applic Mag 22:(6) (2002), 54–63.
13. Z. Szalavári and M. Gervautz, The personal interaction panel—A two-handed interface for augmented reality, Comput Graph Forum 16:(3) (1997), 335–346.
14. M.K. Chandraker, C. Stock, and A. Pinz, Real-time camera pose in a room, in 3rd ICVS, Graz, Austria, April 2003, pp. 98–110.
15. M.A. Fischler and R.C. Bolles, Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography, Commun ACM, 24:(6) (1981), 381–395.
16. C.P. Lu, G.D. Hager, and E. Mjolsness, Fast and globally convergent pose estimation from video images, IEEE PAMI, 22:(6) (2000), 610–622.
17. A. Ansar and K. Daniilidis, Linear pose estimation from points or lines, in ECCV, edited by A. Heyden et al., Copenhagen, Denmark, May 2002, Springer, New York, vol. 4, pp. 282–296.
18. G. Reitmayr and D. Schmalstieg, An open software architecture for virtual reality interaction, in VRST, Banff, Canada, 15–17 November 2001.