Management of Tracking for Mixed and Augmented Reality Systems

Peter Keitler, Daniel Pustka, Manuel Huber, Florian Echtler, and Gudrun Klinker

Abstract Position and orientation tracking is a major challenge for Mixed / Augmented Reality applications, especially in heterogeneous and wide-area sensor setups. In this article, we describe trackman, a planning and analysis tool which supports the AR-engineer in setup and maintenance of the tracking infrastructure. A new graphical modeling approach based on spatial relationship graphs (SRGs) eases the specification of known as well as the deduction of new relationships between entities in the scene. Modeling is based on reusable patterns representing the underlying sensor drivers or algorithms. Recurring constellations in the scene can be condensed into reusable meta-patterns. The process is further simplified by semi-automatic modeling techniques which automize trivial steps. Dataflow networks can be generated automatically from the SRG and are guaranteed to be semantically correct. Furthermore, generic tools are described that allow for the calibration/registration of static spatial transformations as well as for the live surveillance of tracking accuracy. In summary, this approach reduces tremendously the amount of expert knowledge needed for the administration of tracking setups.
Keywords: Mixed/Augmented Reality, spatial relationship graph, tracking, sensor fusion, authoring, calibration, registration, error analysis.

1 Motivation

Tracking the position and orientation of users and objects in the scene is one of the most critical issues for Mixed / Augmented Reality applications. A plethora of tracking methods based on various sensing technologies (e.g. optical, infrared, magnetic, ultrasonic, inertial, etc.) have been desribed for this purpose, all of them having their particular advantages and drawbacks [23]. Therefore, in order to achieve tracking at the expected level of robustness, speed and precision, rather complex tracking setups

Technische Universität München, Germany, e-mail: keitler@in.tum.de

are often necessary, potentially requiring heterogeneous multi-sensor environments that include both mobile and stationary sensors of various modalities. In this respect, we expect increasing numbers of sensors, such as cameras as well as ubiquitously available RFID readers and WiFi trackers, to become available in private and public buildings, coupled with increasing tracking and self-localization facilities in mobile devices. Until now there is no standardized way to accomplish the integration of such sensors into an application.

Typically, MR/AR applications have special solutions for special situations. This complicates the maintenance of tracking environments since changes in the infrastructure often require the source code of one or even several applications to be adapted, undermining system integrity. Developers of MR/AR applications, on the other hand, would rather focus on describing and modeling the application and its visualizations than on the tracking. Typically, applications only need to know the transformations between a few virtual or real world entities. For them, it is not important, how the tracking is achieved – as long as it is provided at expected quality levels. As a consequence, we expect a new breed of professionals, called *AR-engineers*, to emerge. They will be responsible for configuring and maintaining large, dynamically changing tracking setups. They will have general knowledge about trackers and the concept of spatial transformations, but neither will they be experts in all kinds of tracking systems, nor will they be experts in conceiving and programming complex calibration or fusion algorithms.

1.1 Requirements

A number of requirements exist for an application-independent facility to allow AR-engineers to set up and maintain a tracking environment.

- **R1: Separation of Tracking from Applications** To support flexible use of tracking facilities across many applications, a common standard is needed for the specification of relationships between sensors and objects and the exchange of tracking data associated with them.
- **R2: Accuracy and Performance** The standardization and abstraction must not lead to a degradation in tracking quality or speed, when compared to a hard-wired setup.
- **R3: Registration and Calibration** Tools must be provided to register and calibrate the increasing number of sensors and objects in tracking environments efficiently, reliably and precisely.
- **R4: Maintainability** The system should enable AR-engineers to describe and register new sensors and objects and to calibrate and continuously monitor them quickly and reliably. At any point in time, they need a clearly documented understanding of the current configuration. It should be impossible for AR-engineers to mistakingly configure physically impossible tracking setups.

1.2 Related Work

Many systems have been described which aim at the integration of multiple sensors and also at providing a layer of abstraction for the applications relying on them. However, they are mostly rather problem-specific and do not follow a general purpose approach, such as for example the perception and control techniques used in autonomous land vehicles [6], [21]. The architectures proposed for this field of application heavily rely on computer vision and make strong assumptions on the topology of the spatial relationships to be monitored [5], [1]. They were not designed with flexibility and maintainability in mind.

More generic architectures are based on so-called data flow networks, directed graphs whose nodes represent components for data acquisition from sensors or for data transformation. Components can be flexibly cross-linked, based on a small set of data types allowed on component inputs and outputs, such as 3DoF position, 3DoF orientation, or 6DoF pose, a combination of both. Such data flow networks constitute one of the most important building blocks of ubiquitous tracking environments and provide the necessary transformation, synchronization, and network transport of tracking data. We will come back to these issues in 2.

The two most prominent examples in the literature are *OpenTracker* [18] and *VRPN* [20]. Whereas *VRPN*, coming from the VR domain, focuses on how various sensors can be linked together and how tracking data can be synchronized and transfered via network, *OpenTracker* originated in AR and also includes rudimentary support for sensor fusion.

Also noteworthy is the *OSGAR* system [4]. It builds upon tracking data amongst others given by *VRPN* and models tracked, statically registered, or deduced spatial relationships in a scene graph [19]. Assumed tracking and registration errors can be propagated along its branches in terms of covariances, thereby providing the application with a measure of the expected tracking accuracy.

Modeling data flow networks using those systems is a great relief, compared to coding all drivers and algorithms from scratch. However, it is still a tedious task and maintainability (R4) is rather delimited by the fact that invalid data flows are not excluded conceptually. Furthermore, none of the described systems provides support for registration of statically aligned entities (R3), an obligation for real separation of tracking from the applications (R1).

1.3 The Ubitrack and trackman Approach

The *Ubitrack* tracking middleware adds a higher level of abstraction to the data flow concepts described above. AR-engineers can describe tracking setups as spatial relationships between sensors and objects. Corresponding data flow graphs are then generated partially or fully automatically. This higher level provides a more intuitive way of describing data flows. AR-engineers can focus on describing the physical sensor arrangements. Thereby, the risk of accidentally specifying physi-

cally impossible data flows is excluded conceptually. Spatial relationship graphs lend themselves to long-term documentation (and visualization) of the modeled sensor arrangement. To this end, the *trackman*[1] tool provides a graphical interface for the management, monitoring and quality analysis of complex and heterogeneous tracking environments. Ubitrack provides the foundation for fulfilling requirements R1 through R4. This chapter focuses on the presentation of trackman and how the Ubitrack functionality is refined in order to accomplish the requirements fully.

2 The Ubitrack Framework

Before diving into the presentation of *trackman*, we provide an introduction to the basic concepts of spatial relationship graphs, data flow networks and spatial relationship patterns. For more details, see [9], [13], [14], [15], [16], and [17].

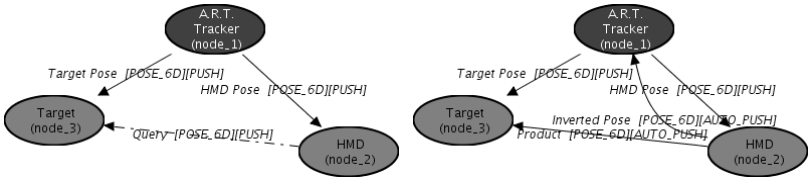
2.1 Spatial Relationship Graphs

The concept of *spatial relationship graphs* (SRGs) was proposed in [13]. A simple example is shown in Figure 1(a). SRGs are graphs which capture the structure of a tracking environment by describing the static and dynamic spatial properties of objects in the environment. The nodes of SRGs represent the local coordinate frames of real or virtual objects and sensors. Directed edges represent spatial relationships between nodes A and B, i.e., the pose (position and orientation, 6DoF) of coordinate frame B relative to frame A. In Figure 1(a), the top node refers to a tracker that tracks the poses of two objects, represented by the bottom two nodes. The right one represents a head-mounted display, the left one refers to a generic target. For instance, the applications might want to render some virtual object on top of it. The directed solid edges represent the (dynamically changing) spatial relationships between the tracker and the two objects.

SRGs are similar to scene graphs in computer graphics [19]. Yet, in contrast to scene graphs, SRGs do not imply a pre-defined hierarchical ordering of the nodes. Applications can request any node in an SRG to assume the role of a root node, requiring the traversal through parts of the SRG from this node to a specified sink node. For a detailed comparison of scene graphs and SRGs, see [7].

The dashed edge in Figure 1(a) describes a request to determine the spatial relationship between the tracked HMD and the tracked generic target. This relationship is not measured directly but can be derived by considering the known spatial relationships of both objects to the tracker. Figure 1(b) show the resulting extended SRG. Two new edges have been added to the SRG: one reversing the spatial relationship between the tracker and the HMD, and the other one replacing the dashed

[1] The Ubitrack runtime library and trackman can be downloaded from http://campar.in.tum.de/UbiTrack/WebHome



(a) Initial SRG (b) Final SRG

Fig. 1 (a) Exemplary SRG with a sensor tracking two targets. The solid lines depict tracking information that is continuously measured and updated. The dashed line depicts spatial information that can be derived from the tracking data. (b) Additional solid lines indicate the respective mathematical operations (inversion, concatenation) on the tracking data.

edge by concatenating the inverted edge from the HMD to the tracker with the edge from the tracker to the generic target.

2.1.1 Use of Cycles for Sensor Calibration and Object Registration

SRGs typically contain multiple edges between nodes, as depicted in Figure 1(b). For example, if multiple independent trackers are tracking the same objects (and the transformations between those trackers are known), the SRG might contain multiple paths to determine the spatial relationship in question. Thus, as another difference to scene graphs, SRGs may contain cycles. These cycles are essential for exploiting redundancy in tracking setups. They provide a means to calibrate or register sensors or objects by using complementary tracking information (i.e., an alternate path in the graph). They also allow system monitoring and the detection of faulty (miscalibrated) sensors (in support of requirement R3).

2.1.2 Edge Characteristics

SRG edges represent a number of sensing characteristics. Most importantly, tracking information can have varying degrees of freedom (DoF), ranging from 2 or 3 translational DoF for wide-area sensors such as GPS or WiFi based trackers, over 3 rotational DoF for mobile sensors such as gyroscopes and compasses to full 6 DoF poses of high-precision trackers for small-area VR or AR setups [17]. Other edge characteristics involve the explicit modeling of sensing errors [2], [11]. A third set of edge properties involves timing and synchronization issues with the special simple case of static edges [14]. The properties provide important criteria for selecting optimal (w.r.t. accuracy or speed) sensor arrangements when setups offer a multitude of options. They are essential to the derivation of good data flow networks (see below) and support requirement R2.

2.2 Data Flow Networks

SRGs are a descriptive rather than an operational specification of a certain setup and are not directly usable by an application. Rather, for efficient use by the Ubitrack runtime system that is included into the applications, they have to be converted into *data flow networks* (DFNs). DFNs consist of computational units that operate on tracking data.

DFNs are instances of *data flow graphs* (DFGs). DFGs are directed graphs and their nodes represent the components to be instantiated in the DFN. Edges represent the flow of tracking data between these components. Sources in a DFG generally represent sources of tracking data (i.e. tracking devices). Sinks correspond to interfaces to applications or to other data flow graphs.

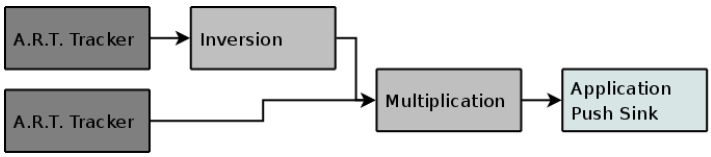


Fig. 2 Data flow network corresponding to the SRG shown in Figure 1(b). An inversion, followed by a multiplication is needed to compute the dashed edge of Figure 1(a) from the two solid edges.

Figure 2 shows the data flow graph that computes the spatial relationship between the HMD and the target in the SRG of Figure 1. The tracking data of the HMD is inverted and then concatenated with the tracking data of the target.

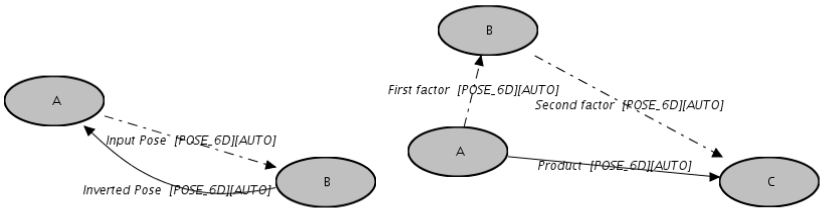
The Ubitrack runtime environment uses such data flow networks. At this level, Ubitrack is comparable to the approaches taken in other systems, such as OpenTracker [18]. In fact, we are able to export data flow networks that have been generated from SRGs into applications that use OpenTracker, using the Ubiquitous Tracking Query Language (UTQL) data exchange format [16],[12] (in support of R1).

2.3 Spatial Relationship Patterns

This section describes how SRGs can be transformed into DFGs. To this end, Pustka introduced the concept of *spatial relationship patterns* [15]. A pattern corresponds to a computational unit, i.e. a node, in a DFG. Tracking data is provided to the computational unit via inputs. It is transformed and then sent out via the output. Good examples are the Inversion and Multiplication nodes in the DFG of Figure 2.

2.3.1 Basic Concept

Spatial relationship patterns are depicted as template SRGs. They describe the effect of a computational unit on an SRG. For example, the Inversion pattern in Figure 3(a) states that, for a given SRG edge from node A to B, inversion adds a new edge to the SRG going in the reverse direction. Similarly, the Multiplication pattern in Figure 3(b) states that an SRG edge from A to B, and an edge from B to C can be concatenated by multiplying the transformations. As a result, a new edge from A to C can be added to the SRG. Component inputs in the DFG correspond to *input edges* of the pattern. They are shown as dashed lines, and the associated *input nodes* as light gray ellipses. The resulting component output in the DFG corresponds to an *output edge* of the pattern. It is shown by a solid line, and the associated *output nodes* by darker ellipses. If the input edges and nodes of a pattern match a part of an SRG, the output edges and nodes can be added to the graph. At the same time, the corresponding computational unit can be added to the DFG.



(a) Inversion (full pattern) (b) Multiplication (full pattern)

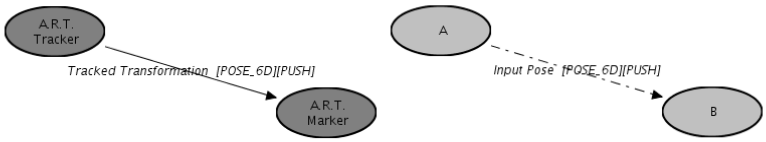
Fig. 3 Basic spatial relationship patterns for the inversion and multiplication of tracking data.

In addition to *full patterns* that have both input edges and output edges, there are *base patterns* that have only output nodes and edges, such as the Tracker pattern in Figure 4(a). Since it has an empty input section, it can be applied any time. It is used to add tracking devices as source computational units to the DFG.

Similarly, *query patterns* have only input nodes and edges (Figure 4(b)). They connect the tracking setup to an application, in form of a query for information about a specific spatial relationship in a scene. An example is the pose of the target relative to the HMD in Figure 1(a). Generally, all base/query patterns have the same structure, an output/input edge with output/input source and sink nodes.

2.3.2 Synchronization Issues

To properly handle measurements that are generated asynchronously by independent sensors, each edge in an SRG is attributed with its synchronization mode, *push* or *pull*. *Pushed* measurements travel downward from source toward sink through a DFN, e.g., when a tracker such as a camera sends new data into the network at its own speed. *Pulled* measurements are pulled upward in a DFN. A pull operation may



(a) A.R.T. Tracker (base pattern) (b) Application Push Sink (query pattern)

Fig. 4 Spatial relationship patterns for a tracking device and to connect the Ubitrack runtime system to an application.

be initiated for example by an application requesting measurements with a specific time stamp via the Application Pull Sink component. Push as well as pull events are propagated recursively through the data flow network.

Synchronization problems occur when two or more unsynchronized inputs have to be combined by a computational unit, such as the Multiplication component. The measurements then need to be valid for the same point in time. When a pull request occurs on the output, measurements have to be pulled for this time stamp on all inputs. When a push event occurs for one input edge, measurements for the same time stamp have to be pulled on the other inputs. The result can then be computed and pushed onwards on the output. Generally, it is not possible to have more than one input in push mode, except when both measurements come from the same tracker or are otherwise synchronized in hardware. Therefore, all except one of them should be in pull mode. To this end, suitable conversion facilities must be included. The Buffer (constant interpolation), LinearInterpolation and Kalman Filter components convert measurements from push to pull whereas the Sampler component converts from pull to push. Refer to [14] for more details.

2.3.3 Pattern Categories

A large number of patterns and associated computational units have already been integrated into Ubitrack [15]. For documentation purposes and for interactive use in trackman, they are categorized with respect to their structure (i.e., as base, full and query patterns) and also with respect to their semantics:

- **Sensor patterns** describe how tracking data is provided to the data flow network. This mainly comprises driver components retrieving data from hardware.
- **Basic patterns** describe trivial transformation steps such as inversion or interpolation of a transformation or concatenation of two transformations.
- **Calibration patterns** represent algorithms such as the hand-eye calibration or absolute-orientation algorithm which are used to determine static spatial relationships under certain boundary conditions.
- **Fusion patterns** represent algorithms which can be used to somehow fuse tracking data to obtain a better, more accurate or a more general result.
- **Persistence patterns** represent components that write tracking or calibration data to a file or read it from there.

- **Network patterns** represent components that send/receive tracking data to/from the network. They are needed to link independent DFNs.
- **Application patterns** represent components that transfer or receive tracking data to/from an application. This principally enables applications not only to consume tracking data but also to transform it somehow and reinject it into the data flow network. Other patterns in this category include render components, which create OpenGL-based 3D graphics output based on tracking data.

Table 1 presents a representative subset of patterns[2], classified according to their structure and semantics. For the application and persistence patterns, there exist pairs of corresponding base and query patterns, such as Player and Recorder for logging and replaying tracking data or Calibration Reader and Calibration Writer maintaining the calibration or registration data of a static transformation in files.

The Application Source and Sink patterns represent endpoints in the DFN which interface it to the application. The Application Pull Source is one of few data flow sources having type pull. It retrieves current tracking data at any time via a callback interface from the application. Similarly, the Application Push Sink pushes data into the application via a callback interface. In both cases, the DFN initiates the flow of tracking data. Application Push Source and Pull Sink, on the contrary, work without a callback mechanism and the application initiates the flow of tracking data. Some calibration patterns will be introduced in section 5.1. For more details, refer to [15].

2.4 SRG Design Activities

Pattern modeling consists of three major activities:

- **A1: Description of the tracking environment** All mobile and stationary sensors and all real and virtual objects are identified. Their known or tracked spatial relationships to one another are described. This activity mainly uses base patterns.
- **A2: Deduction of indirect spatial relationships** Full patterns are applied to suitable parts of an SRG either by an automatic pattern matching process or interactively by the AR-engineer.
- **A3: Definition of the runtime interface to the application** On the basis of query patterns, application interface(s) are inserted into the SRG.

trackman can assist AR-engineers in generating SRGs for a given tracking environment (A1 and A3). Ubitrack can use pattern matching techniques to automatically derive a DFG from a given SRG [9], [17] (A2). AR-engineers can influence the creation of DFGs manually in trackman, supporting requirements R2 and R4 (A2). The interactive variants of these activities are described in the subsequent sections.

[2] A comprehensive reference is provided at http://campar.in.tum.de/UbiTrack/WebHome

Table 1 Pattern categorization matrix showing a subset of the existing Ubitrack patterns. The transformation types are neglected for the sake of readability.

		Syntax		
		Base pattern	Full pattern	Query pattern
Semantics	Sensor	A.R.T. Tracker Static Transformation		
	Basic		Multiplication Inversion Buffer Interpolation Collector Gate Sampler	
	Calibration		Hand-Eye Calibration Absolute Orientation Tip Calibration	
	Fusion		Kalman Filter Functional Fusion	
	Persistence	Player Calibration Reader		Recorder Calibration Writer
	Network	Network Source		Network Sink
	Application	Application Push Source Application Pull Source		Application Push Sink Application Pull Sink X3D Object Background Image

3 trackman: Interactive Modeling of Spatial Relationships

trackman is a configuration and monitoring tool for tracking setups. It has a graphical interface, showing the current configuration of a setup in terms of both, SRG and DFG. It also provides interactive means to access all patterns that are known to the Ubitrack runtime system and to integrate them into the current configuration.

3.1 System Architecture

In order to keep trackman independent from Ubitrack development and to ensure its compatibility with upcoming patterns, it was designed as a lightweight and generic tool. Architecturally, it is organized as a Ubitrack application.

trackman does not have special knowledge of patterns but rather imports the current set of available patterns from external description files that come with the Ubitrack runtime library. The description language is based on the UTQL[3] data exchange format [16]. In addition to the mere graph structure of the patterns, it also allows to specify important meta information. Type information for node/edge/pattern attributes (see 2.1.2 for the most important ones) as well as pattern documentation

[3] http://ar.in.tum.de/static/files/ubitrack/utql/utql.xsd

have to be provided. The resulting *pattern template specification language* XML schema[4] allows for the formal description of available patterns. trackman uses the meta information to allow for convenient configuration of node, edge, and pattern attributes in its property editor and to display documentation to the user, as can be seen in Figure 5.

3.2 Graphical Layout

Figure 5 presents a screen dump of trackman showing the interactive construction of the SRG and DFG of Figures 1 and 2. The tree on the left shows excerpts of the list of all patterns, accessible both with respect to semantic and structural (layout) categories. Below, the property editor allows to inspect and edit settings associated with the selected node, edge, or pattern. On the top left, a search facility allows the AR-engineer to restrict the displayed elements in the tree to only those patterns that contain all specified strings.

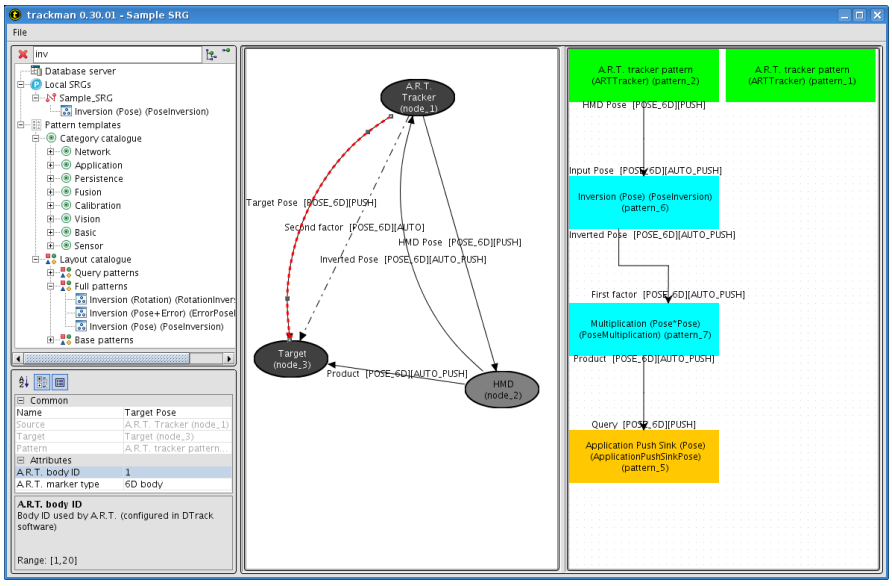


Fig. 5 The *trackman* graphical modeling tool for spatial relationship graphs.

The central area is tiled, showing the current SRG and/or the DFG. In the DFG pane on the right, data sources (corresponding to base patterns, green) are the uppermost components, followed by intermediate computational units (full patterns, cyan), and finally the lowermost data sinks (query patterns, orange). The latter ones

[4] http://ar.in.tum.de/static/files/ubitrack/utql/utql_templates.xsd, http://ar.in.tum.de/static/files/ubitrack/utql/utql_types.xsd

represent interfaces to applications. AR-engineers can alter the tracking setup in the SRG window. Resulting updates are automatically brought to the DFG window. At intermediate stages of the configuration process, not all nodes in the DFG window need to be integrated into the data flow network. For example, the right green node in Figure 5 has not yet been connected to other modules.

3.3 Interactive SRG Generation

Starting with an empty work area in trackman, we use base and query patterns similar to those shown in Figure 4 to describe the directly existing spatial relationships in the tracking environment and the application requests. To this end, they are dragged from the tree view on the left to the SRG editing workspace.

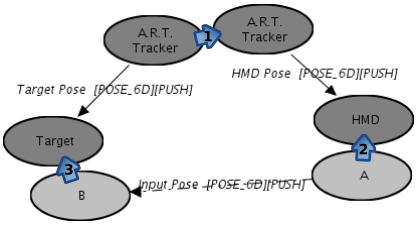


Fig. 6 Identification of coordinate systems via node unification. The indicated unification steps result in the SRG shown in Figure 1(a).

For the SRG in Figure 1(a), the A.R.T. Tracker pattern is dragged twice into the work area – once for each of the two targets. Names, IDs, and other attributes are modified by selecting the respective node, edge, or pattern and applying the settings in the property editor. The query pattern Application Sink is dragged into the work space to describe the request that an edge be provided which describes the spatial relationship between the HMD and the target (see Figure 6).

Using the *node unification* interaction scheme, all three patterns can be merged to form a single graph. To express that two nodes from different patterns are identical in the SRG, AR-engineers can drag one node on top of the other one. As a result of this operation, the subgraphs are merged at this node. Node unification can be applied to all combinations of input and/or output nodes. Nodes that result from unification of at least two output nodes are shown darker than normal output nodes and with a white label. Nodes within a single pattern cannot be unified (principle of pattern atomicity). Figure 6 shows the node unification steps which lead to the SRG shown in Figure 1(a).

3.4 Interactive Deduction of Spatial Relationships

Another interaction scheme is needed to let AR-engineers specify which operations should be applied to the tracking data such that additional spatial relationships can be derived. To this end, full patterns have to be integrated into the SRG, thereby adding further (deduced) edges in terms of their output edges.

By *edge matching*, an edge from the input section of a new pattern is matched against an edge that exists already in the SRG or that is part of the output section of another pattern. The operation also immediately updates the corresponding DFG, linking the input of the computational unit with the output of another component.

The edge matching operation is performed by selecting the two edges, and then invoking the match operation from the menu. Both edges must have the same edge characteristics (according to 2.1) and illegal matchings are inhibited.

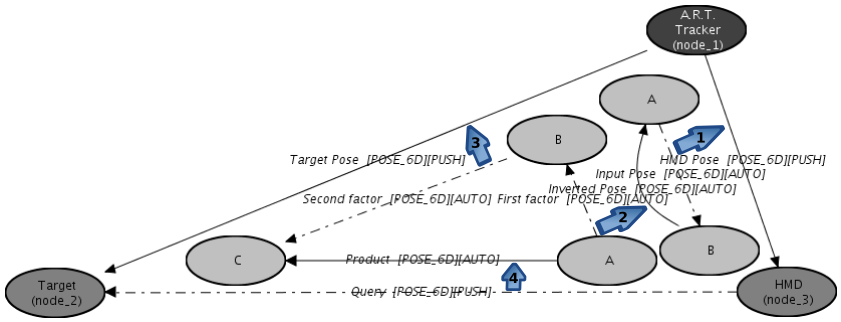


Fig. 7 Identification of data input and output of patterns via edge matching. The indicated matching steps result in the SRG shown in Figure 1(b).

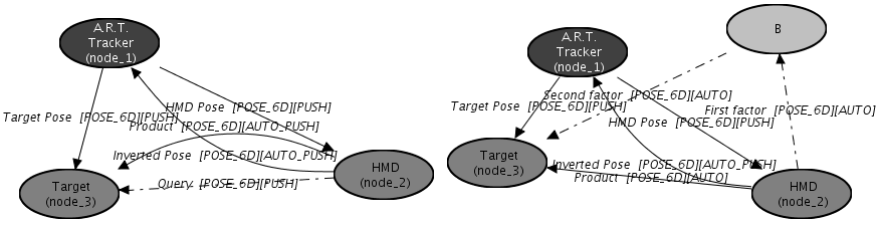
Edge matching implies node unification on the source and sink nodes, respectively, if necessary. Edges belonging to the same pattern cannot be matched (again due to pattern atomicity). The edge matching steps which lead to the SRG shown in Figure 1(b) are depicted in Figure 7.

trackman supports the analysis of synchronization issues. For many full patterns, it can perform the recursive propagation of synchronization mode flags on-the-fly (according to 2.3). For other full and query patterns that only allow for a specific constellation of mode flags, still a consistency check can be performed. Edge matchings with incompatible sync flags are inhibited. The conflict can be resolved manually by converting some push edges to pull mode.

3.5 More Modeling Functionality

With node unification and edge matching, SRGs can be constructed from scratch. Additional functionality is needed when dealing with existing SRGs. This is impor-

tant for the maintenance of existing setups (R4) and also to recover from modeling mistakes. Therefore, trackman also provides the following interaction schemes.



(a) Isolate pattern outputs (b) Isolate pattern inputs

Fig. 8 Result of the isolate pattern outputs and inputs operations, invoked on the Multiplication pattern contained in Figure 1(b).

- **Isolate Pattern Outputs**: trackman is able to separate the output edges of the pattern from connected nodes/edges in the input section of other patterns. The effect of this operation is shown in Figure 8(a). The output edge of the Multiplication pattern contained in the sample SRG (Figure 1(b)) is isolated from matched input edges. Concretely, the matching step 4 of Figure 7 is revoked.
- **Isolate Pattern Inputs**: To complement the previous scheme, trackman is also able to separate the input edges of a pattern from its context, effectively annulling all dependencies between these input edges and corresponding output nodes/edges of other patterns. Invoking this operation on the same Multiplication pattern results in Figure 8(b). The edge matching steps 2 and 3 as depicted in Figure 7 are revoked. This implies also the separation of those input nodes that are neither source nor sink of an output edge of the pattern, such as node “B” of the Multiplication pattern.
- **Isolate Entire Pattern**: This operation combines the two operations above and brings the pattern back to its atomic form.
- **Delete Pattern**: The pattern is removed from the current SRG. It does not matter whether the pattern was integrated in some larger SRG structure or existed in its atomic form. In the former case, an isolate pattern step is implicitly performed first.
- **Hide Pattern**: Parts of the SRG are hidden to provide an abstracted, clearer view of the SRG in the editor window. trackman provides this functionality on a per-pattern basis. This helps to maintain clarity in large SRGs such as the one shown in Figure 12 which consists of approximately 100 patterns.

3.6 Ordering of Design Activities

It is up to the AR-engineer to decide about a suitable design approach. Patterns may be added to the SRG in any sequence. Furthermore, patterns may be combined using the node unification and edge matching metaphors in any sequence. The output edge of a selected pattern may therefore be associated with a subsequent input edge even though it is currently unclear how the output edge can be deduced since the input edges of that pattern haven't been matched yet. A valid DFG, of course, requires all input edges to have been matched properly.

The design process may therefore be started either with the environment, some basic fusion algorithm or also the application interface. The first case might also be denoted a *bottom-up approach* since the AR-engineer starts with physical entities (A1) and refines information step-by-step (A2), resulting in an application-level (A3) piece of information. In the opposite *top-down approach* the engineer could start with the application interface (A3) and drill down through various algorithms (A2) to finally reach real-world sensors and objects (A1). For clarification, going *up* according to the degree of abstraction from raw sensor measurements towards application-level data comes along with going *down* in the data flow from data sources to data sinks.

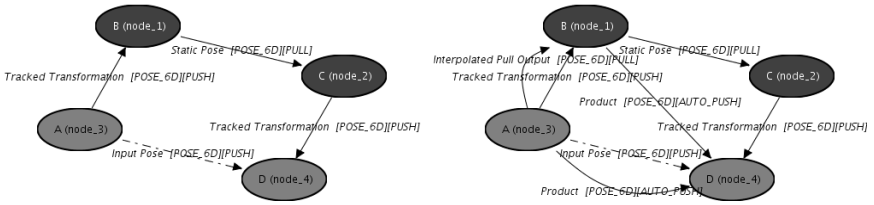
4 Advanced Interactive Modeling Concepts

This section describes two techniques which can further ease the SRG modeling process. Semi-automatic modeling automates simple operations and lets the user focus on the essential deduction steps. Meta patterns provide best-practice solutions to well known problems, reducing the modeling problem to the addition of a few patterns only. Both techniques significantly reduce the amount of modeling operations that have to be performed manually.

4.1 Semi-automatic Modeling

Manual pattern matching can become a very tedious procedure. In more complex setups, the amount of patterns to be integrated in the SRG increases quickly. A concatenation of n edges requires $n-1$ applications of the Multiplication pattern. In addition to that, some edges have to be inverted. In practice, approximately half of all matchings of full patterns fall upon the Inversion and Multiplication patterns (e.g. 22 out of 42 in Figure 12). Automatic pattern matching can relieve the user from the trivial aspects of these and other modeling operations. Figure 9 depicts a typical modeling situation. The transitive transformation from A to D shall be deduced using known transformations from A to B, B to C, and C to D, respectively. Three full patterns are necessary to solve this simple problem and overall six solutions

exist, one of them is shown in Figure 9(b). It first deduces a transitive transformation from B to D (Multiplication), then converts the mode of transformation A to B from push to pull and finally concatenates both to the desired result.



(a) Deduction problem (b) Possible Solution

Fig. 9 Typical modeling situation which requires many applications of full patterns.

On the opposite side of options, fully automatic pattern matching is also offered by Ubitrack [9]. Yet it has its limitations in selecting optimal patterns for every purpose. It is not easy to ensure that the chosen deduction steps meet the AR engineer's notion of the solution. Particularly the many push/pull variations may require fine-tuning by the engineer, once the overall setup has been configured. Assuming that the two tracked edges in Figure 9 offer comparable tracking quality at different frequencies, the position of the Interpolation influences the resulting quality and one would want to interpolate between measurements of the faster tracker. Differing tracking qualities between both trackers further complicates the consideration.

To exploit the best of both options, trackman provides semi-automatic modeling facilities. During manual operations, AR-engineers can enable automatic pattern matching for individual groups of patterns (e.g., for all variations of the Multiplication pattern or the Inversion pattern) while keeping other, more specialized patterns under strict manual control.

The automatic pattern matcher can be invoked in two ways. The first is to select the source and then the sink node of the transformation to be deduced and then to activate the matcher in the menu. The second method is to first insert a query pattern by unifying its source and sink nodes and then invoke the matcher on the corresponding input edge.

4.2 Meta Patterns

Another approach to simplify the modeling task is to provide template solutions for common, recurring problems in terms of *meta patterns*. Basically, they are incomplete SRGs and contain only those patterns that belong to the reusable core of the solution. Interchangeable parts such as the specification of a certain tracker are left open. A meta-pattern can be embedded into an SRG like any other pattern.

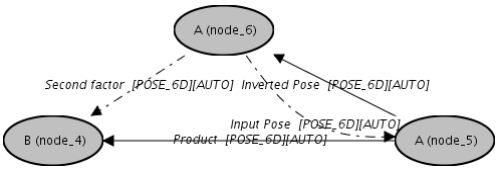


Fig. 10 Meta pattern describing the principal layout and application interface of the sample SRG in Figure 1(b).

To illustrate the idea behind meta patterns, Figure 10 shows the sample SRG from Figure 1(b) with all base patterns removed, i.e. nodes are gray rather than black (see also Figure 8). This meta pattern still conveys the basic structure of the sample application, with HMD and target being tracked by a single tracking system, as well as the application interface. It can be completed by simply matching an arbitrary tracker pattern (providing a push measurement of type 6DoF pose) twice with the input edges of the meta pattern in order to obtain a valid data flow description again.

5 Tools to Analyze and to Interact with Data Flows

Sections 3 and 4 dealt with SRG modeling, showing how DFGs can be provided to applications using trackman. In addition to creating such data flows, a major task of AR-engineers involves the calibration of sensors and the registration of objects, i.e. the careful estimation of fixed spatial relationships between groups of sensors and/or objects. For example, in multi-sensor setups, the poses of all stationary sensors have to be determined. Similarly, groups of mobile sensors and objects that move together as a package need to describe their fixed spatial relationships to one another. AR-engineers need to measure these static relationships accurately during configuration time. Such relationships are represented by static edges in the SRG.

During daily use of the tracking setup, the engineers further have to monitor these relationships to determine whether changes due to wear and tear cause systematic errors that require re-registrations. Another concern is tracking accuracy. It is critical to be able to continuously evaluate a given sensor setup according to its current precision and accuracy [2] such that applications can deal appropriately with different levels of tracking quality.

This section presents the tools provided by trackman and Ubitrack to support these needs. In this respect, trackman behaves like any other application using Ubitrack. This implies that, in contradiction to 3.1, trackman needs to know some interface patterns that allow him to interact with the data flow. The implementation of this functionality in trackman still has a preliminary status. Yet, it already proved to useful for several scenarios, e.g. [10], [11].

5.1 Tools for Calibration and Registration

trackman provides a generic means to carry out calibration procedures including the necessary user interactions. As a matter of principle, calibration does not differ from any other tracking application; it can be described fully by SRGs. They can be directly instantiated in trackman so that no additional implementation is necessary to solve the calibration problem.

In a typical calibration process, the AR-engineer has to move an object that is tracked by sensors in the environment. The tracking data of each sensor is stored, and the relative pose of the sensors can be determined from these data sets. Other processes require AR-engineers to align several objects (e.g. for HMD calibration) or to point with the tip of a tracked pointer at a specified location in the world. In both cases, the engineer has to signal when he is ready to take a measurement. When enough measurements are selected, the pose of the object can be registered.

Ubitrack provides a collection of patterns for state-of-the art registration and calibration algorithms [15], among them solutions for the absolute orientation [8], hand-eye calibration [22] and tip calibration (cf. Table1). These patterns need to be embedded into an SRG context that supports capturing and recording of data as well as user interactions such as a button press event, as needed by the calibration task. trackman provides a generic user interface for this purpose.

When AR-engineers calibrate or register objects, tracking measurements are streaming (pushed) from one or more sensors into the DFN. They can be conditioned in three different ways before being passed to the registration or calibration algorithm:

- **Continuous Measurements**: Measurements are continuously collected. They can either be directly fed into the calibration algorithm or stored using the Collector and Calibration Writer patterns (cf. Table1).
- **Discrete Measurements**: Samples are collected at regular time intervals. For this purpose, the collection mechanism for continuous measurements is extended by adding the Buffer (or another interpolation component) and Sampler components upstream in the data flow. The desired frequency can be specified as an attribute of the Sampler component.
- **User-Triggered Measurements**: Instead of sampling at regular time intervals, measurements are taken when the user presses a button. This asynchronous event triggers a gate to accept a suitable tracking measurement (either shortly before or after the button event). Typically, a number of user-triggered measurements, e.g. using calibration points, are collected. This setup uses the Gate pattern in combination with an Application Push Source pattern (for Button events) upstream in the data flow. A button is provided by trackman to interact with the Application Push Source component.

Some calibrations (such as tip calibration or registering a tracker coordinate frame with a known CAD model) just need measurements from one tracking modality as input. Data can then flow directly from the data collection components to the parameter estimation components. This allows for *on-line incremental parameter*

estimation as soon as the minimal number of measurements is provided. If desired, more measurements can be taken to incrementally improve the registration until the residual error is considered to be small enough.

In many calibration and registration processes, however, data streams of different trackers need to be associated with one another to obtain pairs of corresponding measurements. There are mainly two interaction methods for AR-engineers to establish such correspondences between measurements from several trackers:

- Use **simultaneous measurements** of the same entity, e.g. a pointing device, having the same timestamp. Both measurements can then be directly fed into the calibration pattern and one SRG is still sufficient to describe the entire calibration. This probably means running two tracking systems in parallel. Balancing the timestamps using interpolation might be necessary if the trackers are not synchronized in hardware. Using this solution, we can still benefit from on-line incremental parameter estimation.
- Exactly **reproduce the measurement**. This corresponds for example to conducting corresponding measurements (e.g. a set of points in space) sequentially using two different tracking systems. Thus, two SRGs are needed for the whole calibration step. The first SRG aggregates a list of measurements made by tracking modality A and stores it in a file (see above). In a second SRG, the actual calibration takes place, using both, measurements from the file via the Calibration Reader pattern and measurements from tracking modality B. This method does not work well with the first and second method of recording measurements described above since it is rather difficult to exactly reproduce a complete trajectory of an object, except if you have a robot at your disposal. Therefore, sets of corresponding pairs of measurements often have to be acquired manually.

Both of these approaches can be assembled from the meta-patterns that gather continuous, discrete or user-triggered data streams. At yet a higher level, they can be flexibly combined (loaded) with patterns to execute the mathematical calibration or registration algorithms. All-together, trackman and Ubitrack thus provides a very flexible runtime environment within which different data collection routines can be bundled with mathematical algorithms as needed to quickly determine static relationships between objects and/or sensors in tracking environments. AR-engineers can perform this task in combination with their efforts towards configuring descriptions (SRGs and DFGs) of the tracking environment they are in charge of. They can also revisit and anew static spatial relationships during daily use whenever this deems necessary - as described next.

5.2 Tools for Online Analysis of Tracking Environments

The functionality provided in trackman for the direct instantiation of data flows is not only useful for registration steps but also for the live evaluation of application data flows and the comparison of different alternatives against a trusted reference.

When trackman is running during the daily use of a tracking environment, it can continuously audit any spatial relationship between some nodes A and B by issuing an Application Push Sink request with respect to that spatial relationship (dashed edge in Figure 1(a)). Thus, trackman will be informed immediately of any new measurements for that edge. Measurements to be compared against the reference value are interfaced to trackman via the Application Pull Sink component; i.e. whenever a new reference measurement is pushed upon the application, alternative measurements of the same type w.r.t. other sensors are pulled. With this setup, it is possible to observe in real-time the deviations of position and orientation tracking, as estimated by the alternative data flows.

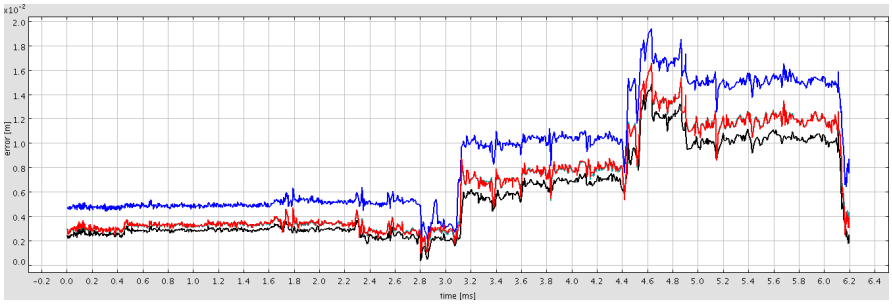


Fig. 11 Exemplary comparison plot showing deviations between four alternative measurements

This online measurement tool has been very helpful in investigating tracking setups for several industrial AR applications [3], [10], [11]. Figure 11 shows an exemplary comparison. The x-axis of the plot represents time. The y-axis represents positional deviations (euclidean distance) of three alternative tracking data flows relative to the reference data flow. Similarly, orientational deviations can be plotted.

6 Application Examples

trackman is actively used in our group for modeling and evaluation of tracking environments. Use cases range from rapid prototyping of small demo setups up to large-scale evaluations. We also used trackman for the evaluation of an indirect tracking setup [11],[10]. It consists of a static camera setup mounted to the ceiling of a room as well as a mobile stereo camera setup on a tripod which is tracked by the static setup. The mobile setup can be placed on-the-fly such that tracking is also possible in areas that are hidden from the static cameras. One might expect a strong degradation of tracking quality of the indirect tracking approach, as compared to direct tracking using only the cameras mounted to the ceiling. Our evaluation shows that the main source of error in this setup is the systematically wrong detection of the orientation of the mobile setup which propagates to large positional errors in the

region of interest. The evaluation furthermore shows that by an appropriate error correction, indirect tracking can be almost as good as direct tracking. The used correction mechanism is based on common reference points in the scene which can be seen by both tracking systems.

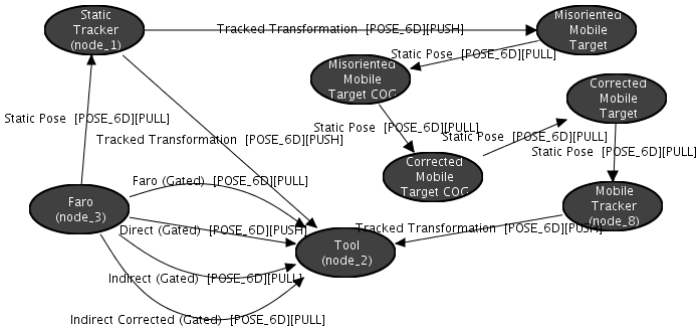


Fig. 12 SRG for comparison of accuracies resulting from different tracking approaches

At the moment, we are integrating a Faro arm[5] as a third tracking system, giving us reliable 3D point measurements as a reference. The corresponding SRG is depicted in Figure 12. It consists of approximately 100 patterns, whereas most of them are hidden. It contains two static transformations that have to be calibrated, one between the "Mobile Target" and the "Mobile Tracker" and another between the "Faro" node and the "Static Tracker" node. Both calibrations and the comparison itself were carried out completely in trackman, without support by additional tools, using just the methods described in 5.1 and 5.2. The plot shown in Figure 11 results from this comparison. Four edges exist between the "Faro" and "Tool" nodes, resulting in three deviations plotted relative to the reference measurement.

7 Conclusion

In 1.1, requirements on a tracking infrastructure have been stated. The "performance" requirement (R2) is clearly met by using the Ubitrack library. It provides state-of-the-art algorithms that can be linked and executed very efficiently. The other three requirements are somewhat interdependent. In a strict sense, "calibration" (R3) and "maintainability" (R4) are implications of the "separation of tracking from applications" (R1) requirement. The described graphical pattern modeling approach contributes a lot to the feasibility of these goals. Together with semi-automatic modeling and the meta pattern concept, the preparation of appropriate application data flows is simplified a lot. At the same time, less expert knowledge is needed, due to the high-level graphical description and exclusion of semantically wrong data flows.

[5] FARO Technologies Inc.

Altogether, this has the potential to lower the inhibition threshold for the acquisition and productive operation of a tracking infrastructure. However, to achieve real separation of tracking from applications, tools to aid in administrative tasks are also needed. We showed the principal feasibility of generic solutions for the essential calibration and error analysis tasks. Nevertheless, there is still room for improvements, e.g. by integrating consistency checks and outlier detection into calibration procedures, by providing a more realistic 3D visualization of spatial coherences and other more intuitive operational concepts.

Acknowledgements

This work was supported by the Bayerische Forschungsstiftung (project trackframe, AZ-653-05) and the PRESENCCIA Integrated Project funded under the European Sixth Framework Program, Future and Emerging Technologies (FET) (contract no. 27731).

References

1. J. Albus. 4D/RCS: a reference model architecture for intelligent unmanned ground vehicles. In *Proceedings of SPIE*, volume 4715, page 303. SPIE, 2002.
2. M. Bauer, M. Schlegel, D. Pustka, N. Navab, and G. Klinker. Predicting and estimating the accuracy of vision-based optical tracking systems. In *Proceedings of the IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, Santa Barbara, USA, October 2006.
3. B. Becker, M. Huber, and G. Klinker. Utilizing RFIDs for location aware computing. In *Proceedings of the 5th International Conference on Ubiquitous Intelligence and Computing (UIC'08)*, Oslo, Norway, 2008.
4. E. Coelho, B. MacIntyre, and S. Julier. OSGAR: A scene graph with uncertain transformations. In *International Symposium on Mixed and Augmented Reality (ISMAR 2004)*, pages 6–15, 2004.
5. M. Darms, P. Rybski, C. Urmson, C. Inc, and M. Auburn Hills. Classification and tracking of dynamic objects with multiple sensors for autonomous driving in urban environments. In *Intelligent Vehicles Symposium, 2008 IEEE*, pages 1197–1202, 2008.
6. E. D. Dickmanns. *Dynamic Vision for Perception and Control of Motion*. Springer Verlag, Berlin, 2007.
7. F. Echtler, M. Huber, D. Pustka, P. Keitler, and G. Klinker. Splitting the scene graph - using spatial relationship graphs instead of scene graphs in augmented reality. In *Proceedings of the 3rd International Conference on Computer Graphics Theory and Applications (GRAPP)*, Jan. 2008.
8. B. Horn, H. Hilden, and S. Negahdaripour. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A*, 4(4):629–642, 1987.
9. M. Huber, D. Pustka, P. Keitler, E. Florian, and G. Klinker. A system architecture for ubiquitous tracking environments. In *Proceedings of the 6th International Symposium on Mixed and Augmented Reality (ISMAR'07)*, November 2007.

10. P. Keitler, M. Schlegel, and G. Klinker. Indirect tracking for on-the-fly elimination of occlusion problems. Demonstration at IEEE International Symposium on Mixed and Augmented Reality (ISMAR'08), September 2008.
11. P. Keitler, M. Schlegel, and G. Klinker. Indirect tracking to reduce occlusion problems. In *Advances in Visual Computing, Fourth International Symposium, ISVC 2008 Las Vegas, USA, December 1-3*, volume 5359(2) of *Lecture Notes in Computer Science*, pages 224–235, Berlin, 2008. Springer.
12. J. Newman, A. Bornik, D. Pustka, F. Echtler, M. Huber, D. Schmalstieg, and G. Klinker. Tracking for distributed mixed reality environments. In *Workshop on Trands and Issues in Tracking for Virtual Environments at the IEEE Virtual Reality Conference (VR'07)*, March 2007.
13. J. Newman, M. Wagner, M. Bauer, A. MacWilliams, T. Pintaric, D. Beyer, D. Pustka, F. Strasser, D. Schmalstieg, and G. Klinker. Ubiquitous tracking for augmented reality. In *Proceedings of the IEEE International Symposium on Mixed and Augmented Reality (ISMAR'04)*, Arlington, USA, November 2004.
14. D. Pustka. Construction of data flow networks for augmented reality applications. In *Proceedings of the Dritter Workshop Virtuelle und Erweiterte Realität der GI-Fachgruppe VR/AR*, Koblenz, Germany, September 2006.
15. D. Pustka, M. Huber, M. Bauer, and G. Klinker. Spatial relationship patterns: Elements of reusable tracking and calibration systems. In *Proceedings of the IEEE International Symposium on Mixed and Augmented Reality (ISMAR'06)*, October 2006.
16. D. Pustka, M. Huber, F. Echtler, and P. Keitler. UTQL: The Ubiquitous Tracking Query Language v1.0. Technical Report TUM-I0718, Institut für Informatik, Technische Universität München, 2007.
17. D. Pustka and G. Klinker. Integrating gyroscopes into ubiquitous tracking environments. In *Proceedings of the IEEE International Symposium on Mixed and Augmented Reality (ISMAR'08)*, September 2008.
18. G. Reitmayr and D. Schmalstieg. An open software architecture for virtual reality interaction. In *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 47–54. ACM New York, NY, USA, 2001.
19. P. Strauss and R. Carey. An object-oriented 3D graphics toolkit. *Computer Graphics*, 26(2):341–349, July 1992. Siggraph '92.
20. 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 andtechnology*, pages 55–61. ACM Press, 2001.
21. S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. MIT press, Cambridge, Massachusetts, USA, 2005.
22. R. Tsai and R. Lenz. Real time versatile robotics hand/eye calibration using 3d machinevision. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 554–561, 1988.
23. G. Welch and E. Foxlin. Motion tracking: No silver bullet, but a respectable arsenal. *IEEE COMPUTER GRAPHICS AND APPLICATIONS*, pages 24–38, 2002.