Spatial Relationship Patterns: Elements of Reusable Tracking and Calibration Systems

Daniel Pustka* Manuel Huber Martin Bauer Gudrun Klinker

Technische Universität München, Fakultät für Informatik
Boltzmannstraße 3, Garching bei München, Germany

ABSTRACT

With tracking setups becoming increasingly complex, it gets more difficult to find suitable algorithms for tracking, calibration and sensor fusion. A large number of solutions exists in the literature for various combinations of sensors, however, no development methodology is available for systematic analysis of tracking setups.

When modeling a system as a spatial relationship graph (SRG), which describes coordinate systems and known transformations, all algorithms used for tracking and calibration correspond to certain patterns in the graph. This paper introduces a formal model for representing such spatial relationship patterns and presents a small catalog of patterns frequently used in augmented reality systems. We also describe an algorithm to identify patterns in SRGs at runtime for automatic construction of data flows networks for tracking and calibration.

Keywords: Tracking, Calibration, Ubiquitous Tracking, System Design

1 INTRODUCTION

Tracking for augmented reality requires the use of many different coordinate systems. Even in small setups we have camera coordinates, object coordinates, relative object coordinates and many more. As the trend goes to increasingly larger systems using two and more different tracking systems or even *Ubiquitous Tracking* installations [26] with many distributed sensors, this poses new challenges to engineers who have to set up and maintain such environments.

1.1 Motivation

In all these installations the same problems have to be solved: propagating subsequent coordinate transformations, combining different trackers for higher accuracy, calibration of unknown transformations and computing the display relative coordinates that are really needed to present the information.

For most tasks, a large number of algorithms can be found in the literature that describe solutions for various combinations of sensors. In most cases, the same problem is solved by many different algorithms which have different properties with respect to performance, accuracy or robustness. Additionally, the algorithms are in general not separated as stand-alone algorithms but embedded in either their specific application or a larger algorithm. Sometimes even different names for the same problem exist in different communities making it harder for someone not familiar with the problem to find the best solution.

*e-mail: daniel.pustka@in.tum.de

While the manual selection of algorithms is feasible in small static setups, we want to move toward larger distributed *Ubiquitous Tracking* systems that allow the introduction or removal of sensors and trackable objects at runtime. For example, a mobile setup, consisting of both cameras and trackable objects, can be brought into an instrumented environment for simultaneous inside-out and outside-in tracking. In such a scenario, the system infrastructure needs to react in a short time and compute a data flow that integrates the measurements from mobile and stationary sensors to satisfy the requirements of the application. In general, the tracking middleware responsible for this task has the same problem as the engineer in that is has to combine algorithms from a library to find a solution for the application.

1.2 Goals

Building augmented reality tracking setups requires up to now a lot of special knowledge and design experience. An experienced engineer who is familiar with a variety of possible solutions for common subproblems can easily apply these solutions to the overall setup without having to rediscover them.

In this paper we present new well-defined design structures that facilitate the reuse of a successful implementation for a new setup. By introducing a formalism for modeling complex setups, combined with a graphical signature of tracking algorithms, this leads to a systematic way of building augmented reality tracking setups by a clever combination of atomic parts.

Like software engineering has moved from trial-and-error to modeling and specification, and computer graphics has moved from programming every single primitive to declaring complex scenes, we want to make the step from reimplementing calibration and tracking algorithms to the automatic generation of the required data flow and algorithms from a declaration of the setup.

Having introduced a formal way of specifying both tracking setups and algorithms, we can go further and build systems that automatically combine known algorithms to support tracking and calibration by dynamically reconfiguring sensor environments.

1.3 Related Work

Scene graphs have been a well-known concept since the *SGI IRIS Inventor* in 1992 [11] and a variety of similar implementations exist. Attempts were made to extend the usage of scene graph APIs to tracking and calibration [25, 14] but showed to be not sufficient. The main difference is the strict tree-like structure of scene graph APIs which is not compatible with the general graphs needed for tracking setups [26]. The closest implementation to the concepts described in this paper is the *OpenTracker* framework [21] for the connection of various tracking hardware. However, in the current version this applies only to static setups and calibration is not supported.

Some approaches claim to generally solve the calibration problem [24] but they usually focus on one mathematical tool for calibration. In contrast to that we try to give a general description of a problem with all possible solutions from the literature. The

engineer searching for a solution of the problem can then choose the best fitting algorithms for his specific application. Section 4.2 shows an example where several solutions are possible.

Our first approach to generating data flows in *Ubiquitous Tracking* setups was based on a distributed path search in spatial relationship graphs using the Bellman-Ford algorithm [26]. It proved to be highly useful in both small and large tracking arrangements, however, usage is limited to sensors providing 6DOF tracking information. We see parts of this paper as a continuation of our previous work, which allows the integration of a broader range of sensors, such as cameras providing 2D measurements.

2 Specification of tracking setups

Before we go into details of some actual tracking patterns we first need to specify the input parameters. Every tracking setup can be described — and this is in fact usually done already [13] — as a graph where the nodes represent coordinate systems or orientation-free points on real or virtual objects and the edges represent transformations between the coordinate systems. We call this graph the *Spatial Relationship Graph (SRG)* [26].

The spatial relationship graph specifies all relevant properties of the tracking setup. Therefore the edges are not only 6D transformations but do have other attributes, most importantly the accuracy of the measurement, but also including any other property needed for the setup. For our system of patterns, we use the concept of the spatial relationship graph with the following assumption: The edges of the graph describe availability of measurements, not actual values.

While for many edges in the spatial relationship graph we have actual measurements of the transformation, in most applications we need the values of an edge that we can not directly measure but rather have to infer indirectly from the topology of the graph and the values of other edges. The spatial relationship patterns describe algorithms to solve subproblems on the way to computing the actual desired value.

2.1 Example Setup

The further concepts of the spatial relationship graph are explained with the example setup depicted in figure 1. The setup contains an HMD, to which an infrared camera is attached for inside-out tracking of five retro-reflective markers in a known and rigid arrangement. This marker defines the world coordinate system, in which the augmentation is to be displayed. For outside-in tracking, an commercial infrared camera system[1] is used, which tracks the world marker as well as another one attached to the HMD. In an approach similar to [18], the outside-in and inside-out trackers are to be combined to deliver improved accuracy and robustness in case one tracking system fails.

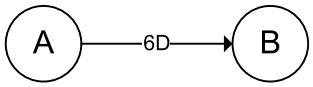
When the system is in operation, the tracking infrastructure needs to provide the transformation between the HMD image plane and the augmentation, in order to display a virtual object at a fixed location in the real world. In reality, the projection matrix of the HMD (the edge between the image plane and the camera) is statically loaded into the renderer, and only the transformation from the camera to the augmentation is tracked in real-time. In terms of the SR graph, the engineer setting up this system has to find a way to indirectly infer the edge connecting these two coordinate systems.

2.2 Edge Types

When specifying tracking setups as a spatial relationship graph, there are different kinds of edges which need to be distinguished:

[1] *DTrack* by Advanced Realtime Tracking GmbH (A. R. T.)

Data type Different algorithms need different input data types. Possible data types in common setups are 6D transformation, 3D translations, 2D translations or 3D→2D projective transformations. Other types of measurements, such as 1D distances or 3D orientations are also imaginable, but not used for the patterns presented in this paper. In the graphical description we write the data type of the measurement as a label to the edge, in this example a 6D edge from A to B.



In our example setup (fig. 1), the ART tracker is a closed system; therefore we model its measurements as 6D edges, although internally, the *2D-3D Triangulation* and *3D-3D Pose Estimation* patterns, explained later on, are used. The camera on the HMD, however, is home-made; therefore, we model both the projective parameters and the single 2D measurements as edges in the spatial relationship graph.

Static/Dynamic In a graph there can be edges that do not change their values since they are for example describing two objects that are rigidly connected to each other, or are assumed to stay constant during during the runtime of the application. We call these edges *static* in contrast to the *dynamic* edges which may change at any instant in time like for example tracked relationships.

In the graphical description we denote static edges with the keyword *static*. Note that edges that are not required to be static may still sometimes be static, for example if a tracked relationship does not change at runtime. As we will see in section 3.3, the information that relationships are static forms an important part of the system description, which allows for many optimizations.

In the example setup, both the ART system and the world target are fixed in the room, and the relationship between the two can be considered static. Other static relationships are a result of their rigid constructions, such as the retro-reflective balls F_{1-5} on the world target or the image plane inside the camera.

Measured/Inferred *Measured* relationships are measurements that are either known, given from a tracking system or from some other data flow component (e.g. a database), while *inferred* relationships are derived from other measurements by an algorithm. In the context of spatial relationship patterns, measured relationships form the underlying basic SRG while inferred measurements are derived via pattern applications, but, from an information-theoretic point of view, do not add new information to the graph.

2.3 Formalism

Formally speaking we can define a spatial relationship graph as a graph $G = (V, E)$ on the node set V of all real or virtual objects and with edge set $E \subseteq V \times V$ of directed spatial relationships. Note that we allow E to be a multiset, as we allow multiple edges between nodes. Over the edge set E we define a number of functions such as $type : E \mapsto \{6D, 3D, 3D{\rightarrow}2D, \ldots\}$, $static : E \mapsto \{\text{static}, \text{dynamic}\}$ and $inferred : E \mapsto \{\text{measured}, \text{inferred}\}$ to represent the attributes associated with each edge.

3 Patterns for Tracking and Calibration

To reach the outlined goals we propose a system of patterns for tracking and calibration. A spatial relationship pattern is used to identify parts of the overall spatial relationship graph for which a known algorithm exists. The corresponding data flow component executes the algorithm and provides the result again as an edge in the spatial relationship graph, where this can be recursively used to

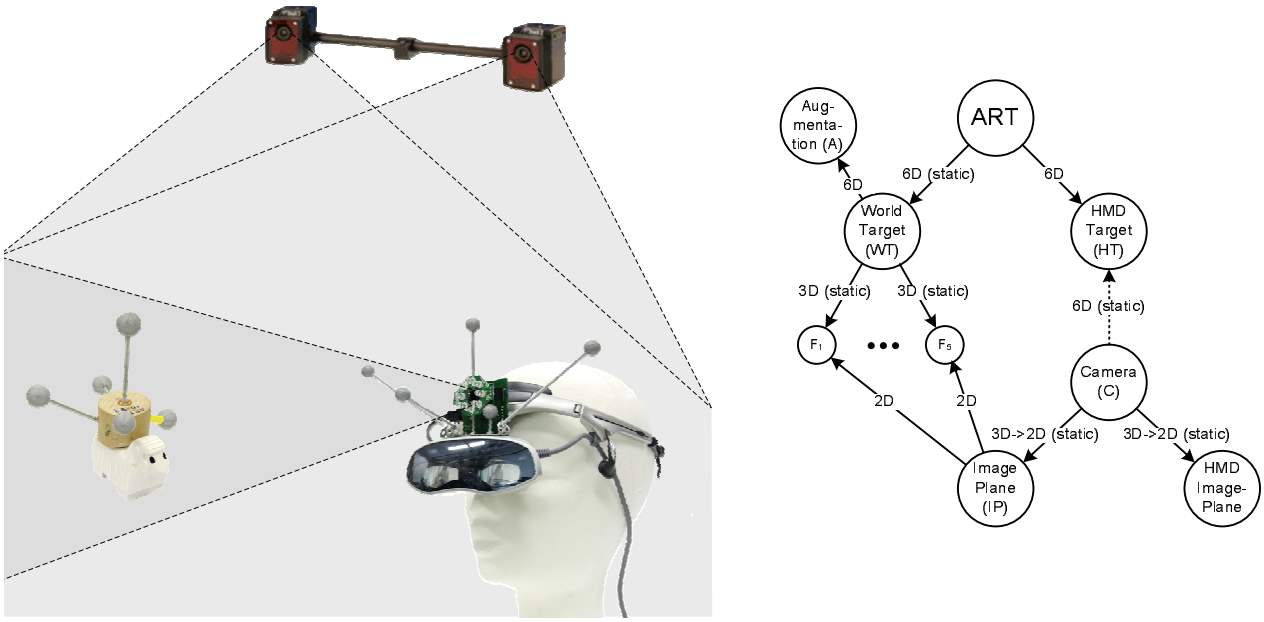


Figure 1: Example of a system combining inside-out and outside-in tracking

identify again solvable subproblems until a solution for all required transformations is found.

Such patterns define the 'signature' of an algorithms to solve a specific problem in a tracking setup. These algorithms have as input a set of measurements as defined by the problem and return a set of measurements that is part of the solution to the overall problem. But other than function signatures in programming languages, spatial relationship patterns do not merely define the type of arguments and return values, but also impose restrictions on the geometric relationship between them.

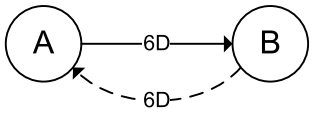
The graphical representation of spatial relationship patterns derives naturally from the one of a spatial relationship graph, because patterns are in fact sub-graphs of the SR graph, as we will explain later. However, there are two types of edges in a pattern: The first are the edges that are required in the spatial relationship graph before a pattern can be applied. They are drawn as ordinary lines. The other type are edges added by a pattern, drawn as dashed lines. These two types of edges define the inputs and outputs of an algorithm. We will give a more formal definition of spatial relationship patterns in section 3.2

3.1 Basic Patterns

Before introducing the formal notation in the next section, we start our overview with some basic patterns that are used in almost any tracking system. Although they are usually implemented implicitly, it is necessary to have them as separate components when trying to create a descriptive language for general tracking setups. Basic patterns construct the transitive reflexive closure of a spatial relationship graph. Using only the basic patterns, many runtime setups (after calibration) can be described already.

3.1.1 Inversion

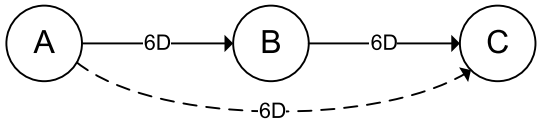
Most edges in the spatial relationship graph are directed edges, since 6DOF pose transformations have a direction as well. The *Inversion* pattern simply inverts this direction and gives the inverse transformation.



While this seems trivial for just the transformations, it gets more interesting when accuracy estimates are involved which need to get transformed as well. Note that the *inversion* pattern can also be applied to edges with 3D rotation or translation measurements.

3.1.2 Concatenation

The most common way of computing the transformation on an unknown edge is by concatenation of subsequent edges.

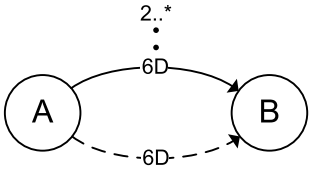


Again this can also be applied to other edge types, such as two 3D rotations or a matrix-vector multiplication of a 6D pose with a 3D vector.

3.1.3 Multi-tracker fusion

The spatial relationship graph is actually a multigraph: sometimes there exists more than one edge between two nodes; this is the case when there are redundant cycles in the setup that after applying several inversion or concatenation steps get transformed into a number of parallel edges providing estimates for the same transformation between two objects.

For perfect measurements these two estimates should be the same, but not in the presence of errors. Statistical fusion algorithms, such as the Kalman filter [1], can be applied to compute the combined estimate.



In our graphical representation we write $2\ldots*$ to mark a multiple edge with at least two instances. An in-depth explanation of the consequences of this concept will follow in the next section.

3.2 Formalism

Before we can continue with more complex examples, we need to introduce a formal definition of spatial relationship patterns: In the following we sometimes do not explicitly distinguish between a pattern and its graph representation. Nevertheless it should always be clear that each graph is motivated by some algorithm that could be instantiated in a data flow network.

Formal Definition We define a spatial relationship pattern P to consist of four parts:

- a graph $G_P = (V_P, E_P)$ describing the basic spatial relationship represented by the pattern,
- edge attributes $type, static$ which both describe additional restrictions imposed on the inputs of the algorithms, as well as describe properties of the output,
- and a set $\mathscr{C}_P$ of correspondences between certain parts of the input. This will be described below in detail.
- a set $\mathscr{A}_P$ of algorithms implementing the specific pattern.

Graph Part of the pattern P is the graph $G_P = (V_P, E_P)$ on the set of nodes V_P which are virtual objects or roles (such as camera, object or fiducal) which eventually have to be assigned to nodes in the spatial relationship graph. Between these nodes we define an edge set $E_P \subseteq V_P \times V_P$ of directed edges, which represent the relevant spatial relationships on which the algorithms operate. We further partition these edges as $E_P = I_P \dot\cup O_P$ into the set I_P of input edges and the set O_P of output edges. The input edges represent the data necessary as input to the algorithm in order to produce the output edges. When applying a pattern to an spatial relationship graph the input edges have to be present while the output edges are inferred by the pattern.

In our graphical representation we draw input edges I_P as normal and output edges O_P as dashed arrows.

Edge attributes The edges of a spatial relationship graph are attributed by functions $type : E_P \mapsto \{6\text{D}, 3\text{D}, 3\text{D}\rightarrow 2\text{D}, \ldots\}$ and $static : E_P \mapsto \{\text{static}, \text{dynamic}\}$ in the same way as they are in the spatial relationship graph. For input edges $e \in I_P$ these represent additional constraints that have to be fulfilled in order for the pattern to be applicable. For output edges $e \in O_P$ these attributes specify further properties of the inferred edges the pattern allows to be derived.

In our graphical representation edge attributes are represented as edge labels, just as for edges in the spatial relationship graph.

Application of a pattern on SRG Given a spatial relationship graph $G = (V, E)$ and a specific pattern P, an instance of this pattern is located in the SRG by an edge matching $\alpha : E_P \mapsto E \cup E^+$ with $E^+ \subseteq V \times V$ which assigns every input edge in I_P of the pattern to a suitable edge contained in the SRG and every output edge in O_P of the pattern to an edge in a set E^+ of derived edges which may or may not be already present in G.

We denote such an instance by $G[\alpha(P)]$, its node set by $V[\alpha(V_P)] \subseteq V$ and its edge set by $E[\alpha(E_P)] \subseteq E \cup E^+$.

For an instance $G[\alpha(P)]$ we can finally decide whether to add the output edges to the SRG or not. We call this procedure an application of the pattern instance $G[\alpha(P)]$ on G results in a new SR graph $G' = (V, E')$ with $E' = E \cup E^+$.

If we repeat this for every locatable instance of P we call this an application of P on G.

3.3 Correspondences

Many of the algorithms we are interested in take pairs of measurements in two different coordinate systems and compute the transformation between those. A good example is the 3D-3D pose estimation – also known as the *absolute orientation* problem – which takes corresponding 3D location measurements of a point in two coordinate systems. If measurement-pairs of at least three points are available, the rotation and translation between the coordinate systems can be computed. In practice, such an algorithm can be applied in two different situations:

Real-time Tracking When tracking a moving object, its position and orientation with respect to some tracker coordinate system must be computed for every point in time when a measurement is made. Therefore, the locations of at least three points on the object must be tracked simultaneously. One could, for example, attach three ultrasound emitters in a known, rigid configuration to the object, track each emitter independently and use the 3D-3D pose estimation algorithm to obtain the full 6D pose of the object.

Tracker Alignment In order to compute a rigid transformation between two tracking systems, it is sufficient to have only one point that can be tracked by both systems simultaneously over time. After obtaining at least three different (non-collinear) measurement pairs, in this case sequentially, the same 3D-3D pose estimation algorithm can be applied to compute the relationship between the two trackers.

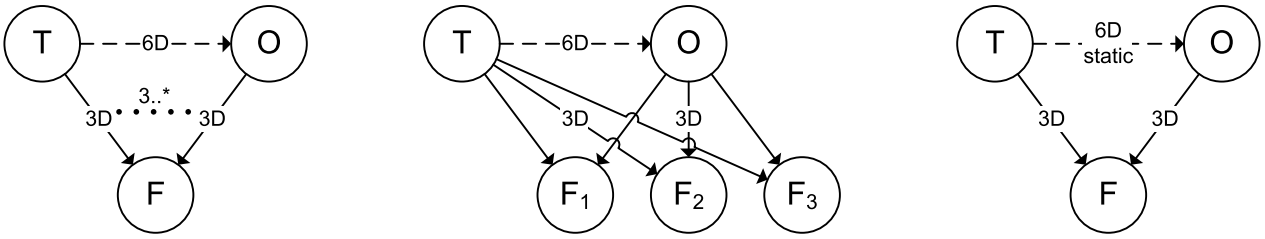
In the example above, the algorithm could be used for tracker alignment with only one pair of measurements at a time, because the estimated transformation was static. Having two different applications of the same algorithm is a general concept which appears with many patterns and thus warrants special treatment.

Formal Definition Every time an algorithm of pattern P needs to relate k different measurements in potentially different coordinate systems, we identify the corresponding edges $e_i \in I_P$ for $1 \leq i \leq k$ as a correspondence set C_P. Furthermore we associate with each correspondence set C_P a set of integers M_P of acceptable numbers of corresponding measurements needed by the algorithm. Most of the time this set will be characterized by some lower bound, but it is also thinkable that some algorithm only operates for example on multiples of four measurements. Finally we define the correspondence of a pattern as $\mathscr{C}_P = (C_P, M_P)$.

Note that these measurements in general have to be distinct from each other and have to supply significantly different data. Furthermore there may be additional restrictions which are not directly representable by a spatial relationship graph or pattern, such as non-coplanarity of measured points.

While patterns with two corresponding edges ($k = 2$) are the most common case, it is also possible to have one (e.g. the Multi-Tracker Fusion pattern, see above) or more than two edges in one correspondence. Also note that an easy generalization to more than one correspondence per pattern is possible.

Notation In our graphical notation, we denote the correspondence sets C_P by dotted lines connecting all edges $e_i \in C_P$. We further annotate these lines with restrictions on the number of acceptable measurements M_P. A label "'3...*"' for example signifies that at least three measurements are required with no upper limit or



a) Pattern notation b) Expansion in space c) Expansion in time

Figure 2: Expanding the 3D-3D pose estimation pattern in time and in space

$M_P = \{3,4,\ldots\}$ in this case. An example for this notation is given in figure 2 a).

In order for a subgraph of a spatial relationship graph to fulfill these requirements and thus to contain a pattern, it is necessary to find an expansion of the correspondence of the pattern, leading to an expanded version $\bar{P}$ of pattern P. For this there are two fundamental possibilities to expand: expansion in space and expansion in time.

Expansion in Space For a space expansion of a correspondence set C_P with restrictions M_P in a spatial relationship pattern P all edges $e_i \in C_P$ contained in the correspondence set are replaced in $\bar{P}$ by distinct instances $e_{i,1},\ldots,e_{i,l}$ of these edges for some acceptable multiplicity $l \in M_P$. This is accomplished by possibly duplicating nodes contained in a designated node set $F_P \subset V_P$. The different edges thus represent similar, but may not represent identical spatial relationships.

For example for the spatial expansion of the 3D-3D pose estimation example from above, it would be necessary to have at least three copies of the node representing the point which is measured in both coordinate systems. The space expansion of this pattern is depicted in figure 2 b).

Spatial expansions of a pattern calculate the output edges of the pattern for each time step in which the input edges are measured. Thus spatial expansion may have both static and dynamic output edges.

Expansion in Time The other possibility is to expand a correspondence (C_P, M_P) is in time. This way the edges $e_i \in C_P$ remain single edges in the expansion $\bar{P}$ but have to be measured at different times during the running time of the application. Furthermore all edges $f_j \in E_P \setminus C_P$ not contained in the correspondence in general have to be static, while the edges $e_i \in C_P$ must not be static and have to display m distinct values for an $m \in M_P$ during the running time of the system.

This is the case for example when calibrating a static relationship by sequentially placing a calibration tool at different points in space and making measurements each time. See figure 2 c) for an example of the time expanded 3D-3D pose estimation pattern. It should be noted that thus expansions in time are only applicable if static output edges are desired as $C_P \subseteq I_P$ and thus all output edges have to be static.

Application Note that a pattern P using correspondences is never directly applied to an SR graph, but rather in one of the two expanded forms. We denote the expanded form of a pattern P as $\bar{P}$.

In cases where both time and space expansion are possible, i.e. enough space-expanded edges are available and the other relationships are static, the time-expansion should be preferred, as the integration of many measurements over time usually results in higher accuracy.

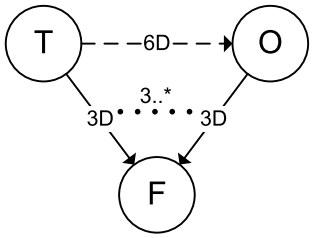
3.4 Advanced Patterns

This section gives a short catalog of frequently used spatial relationship patterns in current augmented reality systems. For every pattern, we give a short overview of how the space and time expanded patterns can be applied as well as references to algorithms that solve the problem.

This collection is by no means complete, but intended to further illustrate the concept of spatial relationship patterns. Building a larger collection is part of our future work.

3.4.1 3D-3D Pose Estimation

This pattern, also known as the *absolute orientation problem*, was already discussed in section 3.3 and it is listed here only for completeness.



Expansion in space Used for 6D-tracking in systems that can track rigid arrangements of multiple 3D points, such as some ultrasound systems.

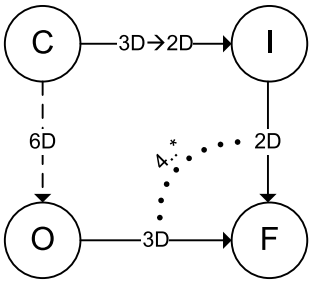
Expansion in time Tracker alignment by moving a single point through the overlapping tracking volumes of two trackers.

Algorithm references The classical algorithm is described by Horn [4]. Other solutions have been found by Walker [9], Umeyama [8] and Arun [3]. For unknown correspondences, a solution is proposed by Gold [15].

3.4.2 2D-3D pose estimation

The goal of *2D-3D pose estimation* is to determine the location and orientation of an object O with respect to a camera coordinate system C, given the projection matrix that maps 3D locations to 2D positions on the image plane I. Given the image coordinates of at least four features F, whose 3D position on the object is known, the transformation C→O can be computed.

For the sake of simplicity we do not distinguish between extrinsic and intrinsic camera parameters, however one could easily adapt the patterns to reflect this distinction if relevant for the particular implementation. For now, 3D→2D edges mean general projective transformations which can contain both intrinsic and extrinsic parameters.



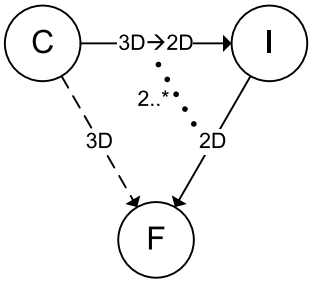
Expansion in space Most single-camera optical tracking systems determine the camera-to-object transformation by localizing multiple 2D feature points with known 3D positions in a single image. This includes marker-based approaches [23] such as the AR Toolkit, which detects the corners of a square.

Expansion in time In its time expanded version, a *2D-3D pose estimation* algorithm can be used to compute the extrinsic parameters of a calibrated camera with respect to some other tracking system. If a single point, tracked by both the camera and the 3D-tracker, is moved around, the position and orientation of the camera can be computed.

Algorithm references Due to the importance of this problem, a large number of algorithms exist that vary in accuracy, speed, robustness or the underlying mathematical structure. Some widely used approaches are described by DeMenthon [10], Lu [19] and Horaud [7]. If the correspondences between the 2D and 3D points are not known, the SoftPOSIT [20] algorithm can be used.

3.4.3 2D-3D Triangulation

This pattern describes the classical n-ocular stereo vision problem. When a single point F is seen by two or more cameras, its position in a common camera coordinate system C can be computed, given the projective transformations from C to the image planes I of the individual cameras.



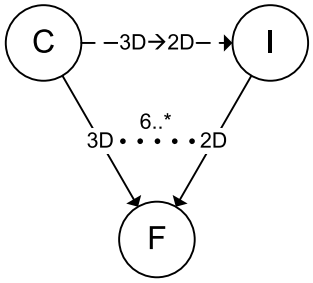
Expansion in space Classical n-ocular stereo, where a number of points are tracked by multiple cameras.

Expansion in time Instead of using multiple cameras, a tracked camera can move around and determine the world-coordinates of feature points seen in multiple images. In the computer vision community, this is known as the *structure from motion* approach. Note that in this pattern, the tracked camera position and orientation has to be part of the projection matrix.

Algorithm references The basic techniques of point reconstruction are well explained by Hartley and Zisserman [17] who also give further references.

3.4.4 2D-3D Projective Calibration

For augmented reality applications, the projective properties of cameras and display device have to be determined. In the simplest case this results in a 3×4 homogeneous matrix, which defines the transformation of 3D points from the camera coordinate system C onto the image plane I. Having eleven degrees of freedom, it is possible to compute the projection matrix from six corresponding measurements of a feature F whose position is known both in the camera image and in the external coordinate system.



Expansion in space In theory, a camera can be calibrated from a single image of a special calibration pattern. In practice this is rarely done, as this requires a lot of features which cannot lie in a single plane.

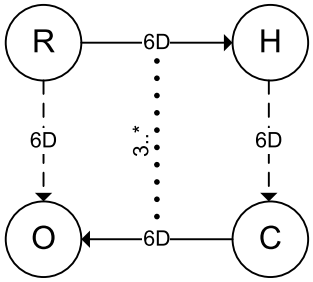
Expansion in time Most algorithms for camera calibration require multiple images of a planar calibration pattern, which contains special features that can be detected by the camera. Often internal and external camera parameters are computed separately and non-linear distortion parameters are added, but without loss of generality, these can be modeled by a single edge in the spatial relationship graph.

Another interesting example is the SPAAM algorithm for calibration of optical see-through HMDs where the user repeatedly has to align a point on the screen with a known point in the world.

Algorithm references The projection matrix can be estimated from a set of corresponding 2D-3D points using the *direct linear transform* (DLT) algorithm [17, 12], for camera calibration, Tsai [5] has proposed another method. Optical see-through head-mounted displays are a special case since the gathering of point correspondences requires user interaction [22] and therefore needs to fulfill some boundary conditions on usability.

3.4.5 Hand-eye calibration

The classical application of Hand-eye calibration comes from the field of robotics[6]. A robot R can, using internal sensors, determine the position of its hand H, on which a camera C is rigidly mounted. An object O, which is not moving, is lying on the table in front of the robot and can be tracked by the camera. If the robot moves to three different positions, resulting in two independent motion-pairs, the hand-eye calibration can compute both the pose of the camera on the robot's hand and the pose of the object relative to the robot's base coordinate system.



Expansion in space Unusual, as this would require multiple identical robots.

Expansion in time Besides its classical application in robotics, the hand-eye calibration is used for tracker alignment when the systems are based on different technologies, so there is no single target that can be tracked by both tracking systems. Instead, two rigidly connected targets are moved through the common tracking area. From corresponding motion pairs, the hand-eye calibration algorithm can compute both the transformations between the trackers and the two targets [24].

Algorithm references The classic approach and first formulation of the problem was given by Tsai and Lenz [6] who solve it by first decomposing the matrix in its translational and rotational parts and then solving first for the rotation and then for the translation. Other solutions following this general scheme were publisher later, but it was Daniilidis [16] who first did the estimation of the translation and rotation at the same time by using dual quaternions. In reality, most implementations only compute the H→C transformation, but then the R→O edge can be inferred trivially.

4 An Example Setup

In order to demonstrate the usefulness of the pattern approach, we now show how the combined inside-out outside-in tracking setup explained earlier (figure 1) works in terms of pattern applications. Careful analysis of the SRG (right half of fig. 1) will reveal three different methods of calibration.

4.1 Tracking

During normal operation of the system, one would compute a transformation between the camera C and the augmentation coordinate frame A in order to perform video-see-through augmented reality. The inside-out-outside-in sensor fusion setup gives increased accuracy and higher robustness in case one tracking system fails. In terms of patterns, this means the following computations are performed:

Applying the *2D-3D pose estimation* pattern to the HMD camera gives the transformation C→WT. For the outside-in tracking, the direction of the HT→ART edge needs to be changed using the *Inversion* pattern. Then, the *Concatenation* pattern is applied twice, resulting in a second C→WT edge. Combining the two edges (*Multi-Tracker Fusion* pattern) yields a third edge from C to WT, but with the desired properties of higher accuracy and robustness. To compute the transformation to the augmentation A, the *Concatenation* pattern is used again, finally giving the C→A edge.

4.2 Calibration

Before the system can be used, the inside-out and outside-in parts of the setup need to be calibrated to each other. The goal of this calibration is to compute the edge C→HT, which describes where on the HMD the ART target is positioned. Applying a systematic pattern analysis reveals three fundamentally different ways to perform that task.

Solution 1: Direct Computation Similarly to the tracking case, the *2D-3D pose estimation* is applied to yield the transformation C→WT. Using the *Inversion* pattern on ART→WT results in WT→ART. Finally, the *Concatenation* is applied twice, resulting in C→ART and C→HT, which is the relationship that is to be calibrated. The single steps of this solution are illustrated in figure 3.

If higher accuracy is needed, the time-expanded version of the *Multi-Tracker Fusion* pattern can be used to statistically combine many estimates of the C→HT edge over time.

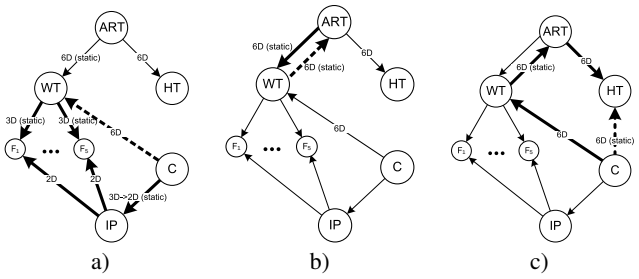


Figure 3: Calibrating the example setup from figure 1: direct computation a) 2D-3D Pose Estimation b) 6D Inversion c) 6D Concatenation

Solution 2: Hand-Eye Calibration This solution (fig. 4) also starts with the *2D-3D pose estimation* applied to the HMD camera, giving C→WT. In the next step, however, the actual measurements of the ART→WT edge are ignored, and just the fact that the relationship is static is exploited. Therefore, the time-expanded *Hand-Eye Calibration* pattern can be applied to the C→WT and ART→HT edges, which yields the desired transformation C→HT and also WT→ART, which is ignored.

Note that the calibration cannot be achieved in one step. The HMD needs to be moved around, since the *Hand-Eye Calibration* is expanded in time.

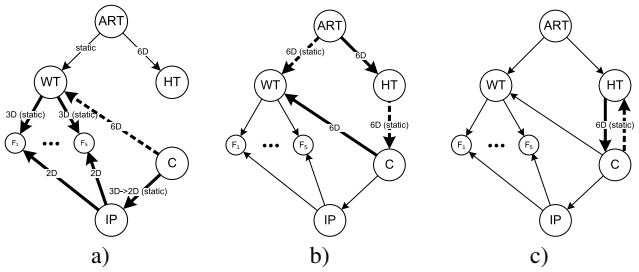


Figure 4: Calibrating the example setup from figure 1: hand-eye calibration a) 2D-3D Pose Estimation b) Hand-Eye Calibration c) 6D Inversion

Solution 3: 2D-3D Pose Estimation The third solution (fig. 5) is similar to the first one, but the patterns are applied in opposite order. First, *Inversion* and *Concatenation* are used to combine the ART→HT and the ART→WT edge resulting in an HT→WT edge. Then another *Concatenation* step is applied which joins the HT→WT edge with each of the feature locations WT→F_i, resulting in five new 3D edges HT→F_i that give the position of the marker points in the coordinate frame of the head target. In this case, the *Concatenation* effectively is a matrix-vector multiplication. Finally, a *2D-3D Pose Estimation* yields the desired edge C→HT. As in solution 1, the time-expanded *Multi-Tracker Fusion* could be used for higher accuracy.

Discussion Having shown three different solutions to the calibration problem, the question is raised, which solution is the best. At this point, we do not have an answer, but analyzing the different algorithms with respect to numerical stability and robustness is part of the ongoing research in our group. In the long term, general rules which are not bound to a specific setup will need to be found.

4.3 Auxiliary Patterns

In real tracking setups, simple transformations are frequently applied to measurements, such as splitting up a 6D pose into its trans-

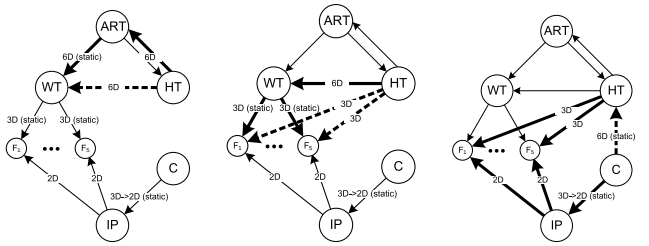


Figure 5: Calibrating the example setup from figure 1: 2D-3D pose estimation a) 6D Inversion and Concatenation b) 6D-3D Concatenation c) 2D-3D Pose Estimation

lation and rotation parts or decomposing a camera matrix into intrinsic and extrinsic parameters. Such transformations can also be described by simple patterns consisting of one input and one output edge.

These *auxiliary patterns*, however, are less useful to human system engineers than they are to automated systems inferring optimal data flows for tracking and calibration from a spatial relationship graph description of a setup. A description of such a system is given in the next section.

5 Automatic pattern detection in spatial relationship graphs

To actually put the spatial relationship graph and spatial relationship patterns to use in a changing or even dynamic tracker setup it is necessary to have some means to automatically process them in a way to apply suitable patterns. This enables us to implement many interesting applications.

Our approach to automatically manipulate spatial relationship graphs with patterns is split into two separate problems:

Single pattern detection Given a spatial relationship graph G and a pattern P, systematically apply the pattern to the graph as described above. This step first locates all instances $G[\alpha_i(\bar{P})]$ of a suitable expansion $\bar{P}$ of the selected pattern P as subgraphs of the SRG. This is related to the well known subgraph isomorphism problem. Formally, given graphs $G = (V_1, E_1)$ and $H = (V_2, E_2)$, the problem is: Does G contain a subgraph that is isomorphic to H. Unfortunately this problem is well known to be NP-complete[2] and thus is unlikely to have an efficient solution in the general case. On the other hand, in this special case there are some factors which work in our favor, which we will discuss later.

Control mechanism In order to reach a specific goal, for example find a calibration setup for a user selected edge, a control mechanism is necessary that determines a suitable sequence of patterns $(P_1, P_2, \ldots, P_n)$ to apply to the SRG in question in order to reach the desired goal. For this, the controller could exhibit some kind of reasoning about rules associated with the patterns or in the simplest case, it may be replaced by an explorative search over the tree formed by all possible pattern combinations.

5.1 Pattern Detection Algorithm

We now present an algorithm which implements the first step of this strategy. Given a graph G and a pattern P we determine the application of P on G. Note that instead of first choosing an expansion $\bar{P}$ and locating this expansion in G, we locate all unexpanded copies of P in G and unify these to expanded pattern instances. We will explain some further details below.

1. Find all occurrences $G[\alpha'_i(P)]$ of the unexpanded pattern P in the graph G
2. Find inclusion maximal sets of patterns that represent expansions of the pattern and respect all relevant constraints
3. If the pattern satisfies the correspondence requirements, we have located an instance $G[\alpha_i(\bar{P})]$ of an expansion $\bar{P}$ of P.
4. If the inferred edge contributes towards the overall strategy as determined by the controller, it is inserted into the graph

Unifying pattern instances After the first step, the algorithm has supplied us with a list $(G[\alpha'_1(P)], \ldots, G[\alpha'_k(P)])$ of all instances of the unexpanded pattern P. We need, in order to successfully detect suitable expansions of the pattern, collect compatible instances and unify them such that a resulting instance $G[\alpha_i(\bar{P})]$ contains as many edges as possible which contribute to the correspondence (C_P, M_P) of P. At the same time, all distinct occurrences of the pattern have to be preserved as individual instances.

An important further constraint is that for any expansion of a pattern, a set of edges contributes to the correspondence only if the information expressed by these edges is not already contained in the collection of the pattern so far. We for now, denote such a located collection by $G[\alpha_i(\bar{P})]$ and check if it actually represents an instance of an expansion of P.

If every edge $e_j \in C_P$ in the correspondence set of P is represented by m edges in the collected instance and $m \in M_P$ is an acceptable number of measurements, then we have found an instance $G[\alpha_i(\bar{P})]$ of a space expansion of P.

If on the other hand every edge $e_j \in C_P$ in the correspondence set of P is represented by a single edge in the collected instance and all edges in $E_P \setminus C_P$ are static as well as all edges in C_P are dynamic, we have found an instance $G[\alpha_i(\bar{P})]$ of a time expansion of P. In any other case, we reject the collected instance as an incorrect expansion of P. This step finally constructs a list of all found expanded instances of P in G.

When to add new edges After an instance of an expanded spatial relationship pattern $\bar{P}$ has successfully been located in the SRG G, we may proceed to insert the output edges O_P of the pattern after determining their new properties.

Here another problem arises, as the unconditional insertion of all derivable edges leads immediately to uninteresting results and perpetual insertion of similar edges. An obvious example for this is that an application of the inversion pattern produces a new edge which can of course be inverted again. But this obviously adds no new information to the spatial relationship graph. Furthermore, with each inversion the accuracy of the edge is reduced which is also undesirable.

To overcome this problem we additionally attribute each edge with a combination of information about the spatial relations (pairs of nodes in the SRG) from which the edge is derived as well as the combination of operations which are used. This is similar to the bookkeeping required to later extract information for the data flow generation for a particular edge.

An exception to this rule are inferred, static edges which may be averaged over time and thus indeed behave better than the set of edges from which they are derived (assuming not all edges contributing are static themselves).

Multiple Solutions If a sequence of pattern applications to an SRG derives multiple solutions for a queried edge, it is necessary to decide which solution is best for the given application. Therefore, each solution has to be evaluated against various criteria, such as accuracy, lag, update frequency or numerical stability of the algorithms used. In the design phase of a system, also the required resources, including the cost of the equipment used, may play a role.

It will be part of the future work in our group to investigate these factors and we will try to find general rules for evaluating different data flows to allow systems to automate this decision.

Data flow construction From the sequence of located pattern instances $(G[\alpha_1(P_1)],\ldots,G[\alpha_k(P_k)])$ as determined by the controller and the pattern detection algorithm which have to be applied to the SR graph G in order to compute a given transformation, a data flow graph can be created, which describes the connected components that performs the actual computations.

A data flow graph is a directed, acyclic graph with the tracking components as inputs and a single output that provides the desired spatial relationship and is instantiated as a data flow network at runtime. However, to save computational resources in larger setups, parts of data flow networks can be shared among different applications. An example of a data flow graph that performs the computations, explained in section 4.1, is shown in figure 6.

Constructing the data flow graph from the sequence of located pattern instances $(G[\alpha_1(P_1)],\ldots,G[\alpha_k(P_k)])$ is done in a rather straightforward fashion: All edges in the initial SR graph are associated with a tracking component or some other service (e.g. a database) that initially provides the unprocessed measurements. These components serve as the basic inputs to the data flow network. When an expanded instance of pattern P_i is applied in the SR graph, this creates a new component in the data flow graph. This component executes an algorithm from the set of associated algorithms of the pattern P_i and provides new outputs in the data flow graph, which are associated with the new edges $E[\alpha(O_{\bar{P}_i})]$ in the SR graph. The inputs of the component are connected to the components associated with the matched input edges $E[\alpha(I_{\bar{P}_i})]$. This is repeated for all pattern applications of the sequence till the desired measurement is available at the end of the data flow graph.

Performance We implemented a proof of concept design of all these concepts above using the high level language python. As expected the performance of this system is not yet good enough to be ready for every day use in real applications. The most computationally intensive parts are as expected finding all occurrences of a pattern in the SRG, as well as unifying these instances.

More clever bookkeeping and more intelligent data structures would be promising to significantly improve the performance in some of these aspects, but there are also fundamental problems stemming from the principal hardness of the problems involved as described above. Fortunately there are a few factors that work in our favor. First the size of a spatial relationship graph should in practice be relatively small so that subgraphs still can be found in reasonable time. Furthermore even if the graphs get larger, it should be possible to restrict the problem to a subgraph of the original spatial relationship graph. A second advantage is that the edges are attributed by type and by static/dynamic relationships. This essentially partitions the edge sets of the graph and leads to simple yet very helpful heuristics for subgraph search.

Also it is plausible that further heuristics may help reduce the size of the problem and thus speed up all parts of the computation. Further investigations of these possibilities are necessary and should be one of the next topics to research.

5.2 Applications

While our current implementation described above serves as a good proof-of-concept, we have yet to integrate the algorithm into real applications. The two scenarios that we are working on are a runtime engine for data flow creation in dynamic sensor networks and an interactive tracking environment design tool.

5.2.1 Data Flow Generation at Run-Time

In order to apply the pattern detection algorithm in a system for automatic data flow network generation in dynamic systems, a tracking middleware is necessary, which handles the registration of new trackers and applications that require tracking information. In this section we describe an approach based on our DWARF middleware. The system is an improvement of our earlier implementation of the Ubitrack concepts, which was adapted to suit the spatial relationship pattern concepts. We refer to [26] for an introduction to the concepts of DWARF and the *Ubitrack Middleware Agent* (UMA), which is responsible for monitoring application queries and the creation of data flow networks.

In DWARF, we implement spatial relationship patterns using a so-called *template service descriptions*, which have needs and abilities corresponding to the inputs and outputs of the algorithm, but where attributes and predicates are not bound to specific values. When a pattern is used in a data flow network, the corresponding *template service description* is copied by the UMA and the attributes and predicates are set to concrete values. Starting and connecting the actual processes is then done by the DWARF service manager.

5.2.2 Tracking Setup Design Tool

In small static setups, such as the one described in section 4, a tracking middleware is unnecessary and would introduce additional overhead. Still, computerized support is desirable when alternatives for computing a transformation can easily be overlooked.

We are working on implementing a semi-automatic design tool for tracking setups, which allows the user to interactively construct a spatial relationship graph with all relevant properties. Then, an edge in the graph can be selected and the program presents all possible data flows that can be used to track or calibrate the selected relation. If no suitable data flow is found, the tool can make additional suggestions such as asking whether it is possible to make a particular edge static during the time of calibration.

6 CONCLUSION AND FUTURE WORK

We have introduced the concept of spatial relationship patterns as a systematic method for finding suitable algorithms that solve the problems of tracking and calibration in complex sensor setups. As the pattern collection given in this paper focuses mostly on camera-based tracking-techniques, we want to expand this catalog to include algorithms used with other sensors such as gyroscopes, accelerometers or distance-based methods. We expect that this may require additional properties to be modeled in the SR graph, but the general formalism should be sufficiently strong to handle these cases. This collection will be available as a web page and provide more links to algorithms than possible within the scope of this paper.

Based on the spatial relationship pattern formalism, we have described an algorithm for automatic construction of data flow graphs in dynamic tracking environments. A system built upon it possibly has access to a larger repository of algorithms to choose from than previous approaches. Besides many small performance improvements, our future work will concentrate on creating a distributed version of the pattern detection to improve scalability in large environments. A promising idea is to use the Bellman-Ford algorithm, implemented in our previous work, to reduce the spatial relationship graph to a smaller subgraph to which the pattern detection is applied.

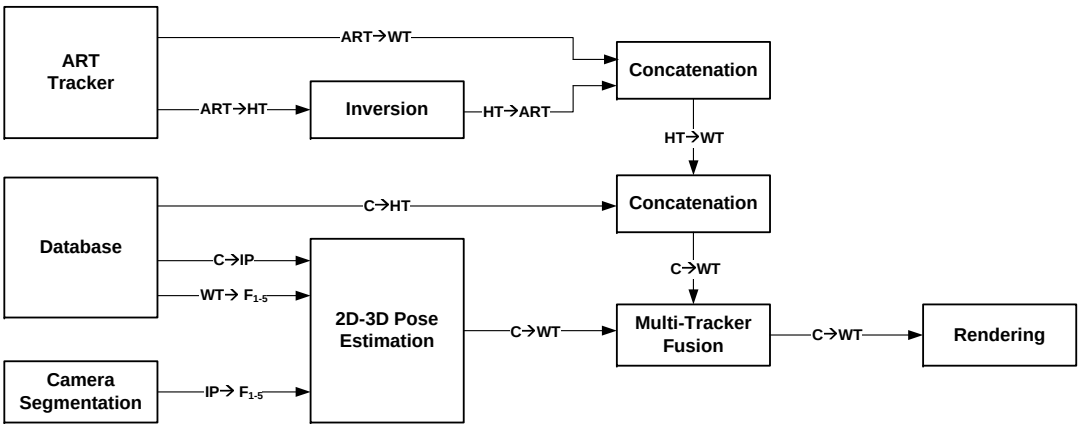


Figure 6: Data flow graph for implementing the example scenario from section 4.1.

ACKNOWLEDGMENTS

Part of this work was supported by the Deutsche Forschungsgemeinschaft (KL1460/2), the Bayerische Forschungsstiftung (project TrackFrame, AZ-653-05) and the European Commission (project PRESENCCIA, contract no. 27731).

REFERENCES

[1] GELB, A., *Applied Optimal Estimation* (MIT Press, Cambridge, Massachusetts; London, 1974), 15th ed.

[2] GAREY, M. R., JOHNSON, D. S., *Computers and Intractability; A Guide to the Theory of NP-Completeness* (W. H. Freeman & Co., 1979)

[3] ARUN, K., HUANG, T., BLOSTEIN, S., *Least-Squares Fitting of Two 3-D Point Sets*, in IEEE Transactions on Pattern Analysis and Machine Intelligence, 9(5) (1987) 698–700

[4] HORN, B., *Closed Form Solutions of Absolute Orientation Using Unit Quaternions*, in Journal of the Optical Society of America A, 4(4) (1987) 629–642

[5] TSAI, R., *A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses*, in IEEE Journal of Robotics and Automation

[6] TSAI, R., LENZ, R., *Real time versatile robotics hand/eye calibration using 3D machinevision*, in *IEEE International Conference on Robotics and Automation*, vol. 1 (1988) 554–561

[7] HORAUD, R., CONIO, B., LEBOULLEUX, O., LACOLLE, B., *An analytic solution for the perspective 4-point problem*, in Comput. Vision Graph. Image Process., 47(1) (1989) 33–44

[8] UMEYAMA, S., *Least-squares estimation of transformation parameters between two point patterns*, in Pattern Analysis and Machine Intelligence, IEEE Transactions on, 13(4) (1991) 376–380

[9] WALKER, M. W., SHAO, L., VOLZ, R. A., *Estimating 3-D location parameters using dual number quaternions*, in CVGIP: Image Underst., 54(3) (1991) 358–367

[10] DEMENTHON, D., DAVIS, L. S., *Model-Based Object Pose in 25 Lines of Code*, in *European Conference on Computer Vision* (1992) 335–343

[11] STRAUSS, P. S., CAREY, R., *An Object-Oriented 3D Graphics Toolkit*, in *Proceedings of SIGGRAPH 1992*, vol. 26, edited by CATMULL, E. E. (1992) 341–349

[12] ZHANG, Z., FAUGERAS, O., *3D Dynamic Scene Analysis* (Springer Verlag, Berlin Heidelberg New York, 1992)

[13] TUCERYAN, M., GREER, D. S., WHITAKER, R. T., BREEN, D. E., ROSE, E., AHLERS, K. H., CRAMPTON, C., *Calibration Requirements and Procedures for a Monitor-Based Augmented Reality System*, in IEEE Trans. Vis. and Comp. Graph., 1(3) (1995) 255–273

[14] SZALAVÁRI, Z., SCHMALSTIEG, D., FUHRMANN, A., GERVAUTZ, M., *Studierstube - An Environment for Collaboration in Augmented Reality*, in Journal of the Virtual Reality Society

[15] GOLD, S., RANGARAJAN, A., LU, C.-P., PAPPU, S., MJOLSNESS, E., *New algorithms for 2D and 3D point matching: pose estimation and correspondence.*, in Pattern Recognition, 31(8) (1998) 1019–1031

[16] DANIILIDIS, K., *Hand-eye calibration using dual quaternions*, in Journal of Robotics Research, 18 (1999) 286–298

[17] HARTLEY, R. I., ZISSERMAN, A., *Multiple View Geometry in Computer Vision* (Cambridge University Press, 2000)

[18] HOFF, W., VINCENT, T., *Analysis of Head Pose Accuracy in Augmented Reality*, in *IEEE Transactions on Visualization and Computer Graphics*, vol. 6(4), 319–334 (IEEE Computer Society, 2000)

[19] LU, C.-P., HAGER, G. D., MJOLSNESS, E., *Fast and Globally Convergent Pose Estimation from Video Images*, in IEEE Transactions on Pattern Analysis and Machine Intelligence, 22(6) (2000) 610–622

[20] DEMENTHON, D., DAVID, P., SAMET, H., *SoftPOSIT: An Algorithm for Registration of 3D Models to Noisy Perspective Images Combining Softassign and POSIT*, Tech. rep., University of Maryland, MD 2001

[21] REITMAYR, G., SCHMALSTIEG, D., *OpenTracker: An Open Software Architecture for Reconfigurable Tracking based on XML*, in *Proceedings of IEEE Virtual Reality* (Yokohama, Japan, 2001) 285–286

[22] TUCERYAN, M., GENC, Y., NAVAB, N., *Single-Point Active Alignment Method (SPAAM) for Optical See-Through HMD Calibration for Augmented Reality*, in Presence: Teleoperators and Virtual Environments, 11(3) (2002) 259–276

[23] ZHANG, X., FRONZ, S., NAVAB, N., *Visual Marker Detection and Decoding in AR Systems: A Comparative Study*, in *First IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR 2002)* (Darmstadt, Germany, 2002)

[24] BAILLOT, Y., JULIER, S., BROWN, D., LIVINGSTON, M. A., *A Tracker Alignment Framework for Augmented Reality*, in *Proceedings of 2nd International Symposium on Mixed and Augmented Reality (ISMAR)* (2003) 142–150

[25] COELHO, E. M., MACINTYRE, B., JULIER, S., *OSGAR: A Scenegraph with Uncertain Transformations*, in *Proc. IEEE International Symposium on Mixed and Augmented Reality (ISMAR'04)* (IEEE, Washington, DC, 2004) 6–15

[26] NEWMAN, J., WAGNER, M., BAUER, M., MACWILLIAMS, A., PINTARIC, T., BEYER, D., PUSTKA, D., STRASSER, F., SCHMALSTIEG, D., KLINKER, G., *Ubiquitous Tracking for Augmented Reality*, in *International Symposium on Mixed and Augmented Reality (ISMAR)* (Arlington, VA, USA, 2004)