Construction of Data Flow Networks for Tracking in Augmented Reality Applications

Daniel Pustka

Technische Universität München, Fakultät für Informatik
Boltzmannstraße 3, 85748 Garching bei München, Germany
Tel./Fax: +49 (0) 89 289 -17085 / -17059
E-Mail: daniel.pustka@in.tum.de

Abstract: Constructing Ubiquitous Tracking setups that can integrate new and unknown sensors dynamically at runtime is a difficult problem. One issue is that sensors in general are not synchronized, which may lead to registration errors when measurements from multiple sensors are combined. The approach presented in this paper solves the measurement simultaneity problem by creating data flow networks that include interpolation/prediction components from spatial relationship graphs descriptions based on a system of spatial relationship patterns.

Keywords: Augmented Reality, Tracking, Ubiquitous Tracking, Sensor Fusion, Synchronization

1 Motivation

Augmented Reality applications require high tracking accuracy, especially in the orientation of the user's head for classical HMD-based systems. Tracking systems also have to deal with fast motions, such as a rotation of the head, which may cause significant registrations errors when the latency of the tracking system is too high. Azuma [Azu97] computes a visible error of 60mm for a "typical" 100ms delay and "moderate" head rotation rate of 50 degrees per second for objects about an arm's length away. While today's tracking and rendering systems are faster, dynamic errors still are a major source of registration error.

This fact has been well recognized and has resulted in the design of tracking setups that include prediction and gyroscopes [AB94]. However, the software was always built with particular hardware setups in mind. In contrast to that, our group wants to build dynamic *Ubiquitous Tracking* setups, where new and unknown sensors can be integrated at runtime.

The particular problem addressed in this paper is that of combining measurements from unsynchronized sensors. When building loosely coupled distributed tracking setups, we cannot assume that measurements from all sensors are made simultaneously and have equal signal run-time from the sensor to the application. The violation of this *simultaneity assumption* [WB97] can cause sig-

nificant registration errors when measurements of multiple sensors are combined. The framework that is presented does not require a centralized instance where data from all sensors is aggregated, nor do we propose to use a single mathematical tool, such as a Kalman filter, in all situations.

1.1 Prerequisites

The approach presented in this document requires that sensors – or at least their drivers to the system – have two characteristics:

Timestamps All sensors must give exact timestamps of their measurements. It is usually not sufficient to timestamp measurements when they arrive in the computer over some interface, but timestamps must represent the time the measurement was made.

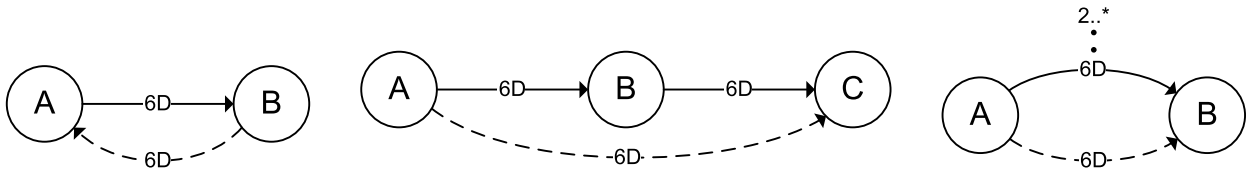
Error statistics All sensors have to provide accuracy information in form of a probability density, e.g. by a covariance matrix. For precise multi-sensor fusion, this probability density must be computed for every measurement. In the case of camera-based tracking, computing covariance matrices for single tracking targets is described in [BSP+06].

In addition to sensors, we assume that all data flow components that perform computations on sensor data handle timestamps correctly and perform error propagation to compute the accuracy of derived measurements based on the accuracy of the original measurements.

2 Spatial relationship graphs

In our approach, called *Ubiquitous Tracking*, a tracking setup is specified using a spatial relationship graph (SRG), which describes relevant coordinate frames and tracking devices. In this section, I will focus on just the aspects relevant for this paper. A more detailed explanation of spatial relationship graphs is found in [NWB+04] and a formal introduction to spatial relationship patterns is given in [PHBK06].

The nodes of a spatial relationship graph represent coordinate frames, e.g. that of a camera located at its camera centre, that of a CAD-model augmented onto some object or that of a tracker target. If the transformation between two coordinate frames is known or measured at runtime, this is indicated by a directed edge between those coordinate frames. Note that edges do not represent the measurements themselves, but indicate availability of measurements, i.e. they usually contain a reference to some software component, e.g. a driver, that provides the actual measurements at runtime. Edges in the SRG also have attributes specifying relevant properties of the measurement, such as the data type (2D, 3D, 6D, etc.) or whether the relationship is known to be static.



a) Inversion b) Concatenation c) Multi-Sensor Fusion

Figure 1: The basic spatial relationship patterns

In this paper, we will only look at 6D measurements, as these are the most common in current AR systems, and the necessary concepts can be explained nicely. However, the same concepts can be applied to other types of measurements.

2.1 Construction of Data Flow Networks

The goal of our approach is to take the abstract spatial relationship graph declaration and construct at runtime a data flow network consisting of tracking and transformation components that provide an application with estimates of those spatial relationships that the application requires. Generally, the transformations needed by an application are not directly measured, but instead can be *inferred* from existing measurements. When a new transformation is inferred, this adds a new edge to the graph which connects different nodes than the existing edges or has different attributes.

To describe which measurements can be inferred from the given SRG description, we use spatial relationship patterns [PHBK06]. Spatial relationship patterns are subgraphs of SRGs, which have two different kinds of edges: input edges that have to be present before a pattern can be applied, and output edges, which are added to the SRG afterwards. Input edges are denoted by solid lines and output edges are dashed.

The goal is to find a chain of pattern applications on the given SRG that allows us to infer the edge that corresponds to an application's request. Each pattern application corresponds to a component in a data flow network that performs the actual computation by taking the measurements of the input components and producing the inferred measurement of the output edge. Therefore, by finding the right chain of pattern applications, a data flow network can be constructed automatically at run time from a given SRG description.

Most tracking problems involving only 6D measurements can be solved using the three spatial relationship patterns, depicted in figure 1:

Inversion The inversion pattern represents the most basic transformation of tracking data: Consider that a transformation from coordinate frame A to coordinate frame B is described by a 4×4 matrix M. Then the inverse transformation, going from B to A, can be computed as M^{-1}. Similar

methods exist when the 6D transformation is described e.g. by a translation and a quaternion.

Concatenation The concatenation pattern exploits the transitivity of spatial relationships: If the transformations from A to B and from B to C are given as 4×4 matrices M and N, the transformation from A to C can be computed as the product MN.

Multiple sensor fusion When two or more edges are available between two nodes, the measurements can be fused by statistical combination, using the accuracy information that accompanies the measurements. This usually results in estimates of higher accuracy.

3 Push-Style and Pull-Style Communication

Most existing data flow architectures for tracking, e.g. [RS01, BBK+01], allow two basic communication protocols for transporting sensor measurements: Event-based push-style and pull-style. While it is clear, that the two are different from a technical point of view, it will be shown that this also enforces certain semantics onto components that implement one of the two methods.

Push-Style Interface Event-based transport of tracking data is most common in currently used systems [THS+01, BBK+01, RS01]. Every time a sensor makes a measurement, which usually happens in fixed intervals, the resulting measurement is put in a packet and sent to one or more receivers. Almost all sensors deliver data using the push-model. Even if it was possible to query a sensor at every time, the signal is usually sampled in fixed intervals and sent to an application.

For the purpose of this paper, the term "Push-Style Interface" is generalized even further and applies to all components that can only deliver information about particular points in time and do not store a history nor are able to look into the future. The term "Push" is used, because using this protocol, a receiver has no way of specifying what time it is interested in.

Pull-Style Interface When a Pull-Style interface is used, the application is typically provided with a method on some locally or remotely available object. Using this method, the measurement for some particular point in time can be queried. A bit more formally, a pull interface is a function $f : t \rightarrow m$, that maps continuous points in time to measurements.

For practical purposes, let us assume that it is not necessary to store the whole history nor to look far into the future, but that results are reasonably valid for a small interval around the current time, which is what an AR application usually needs to render an augmentation.

Of course, no physical sensor is able to provide a Pull-Style interface. Components that do, fall into two categories: Measurements of static relationships, derived by sensors, manual measurement or calibration, are valid for all points in time. If the tracked object is moving with respect

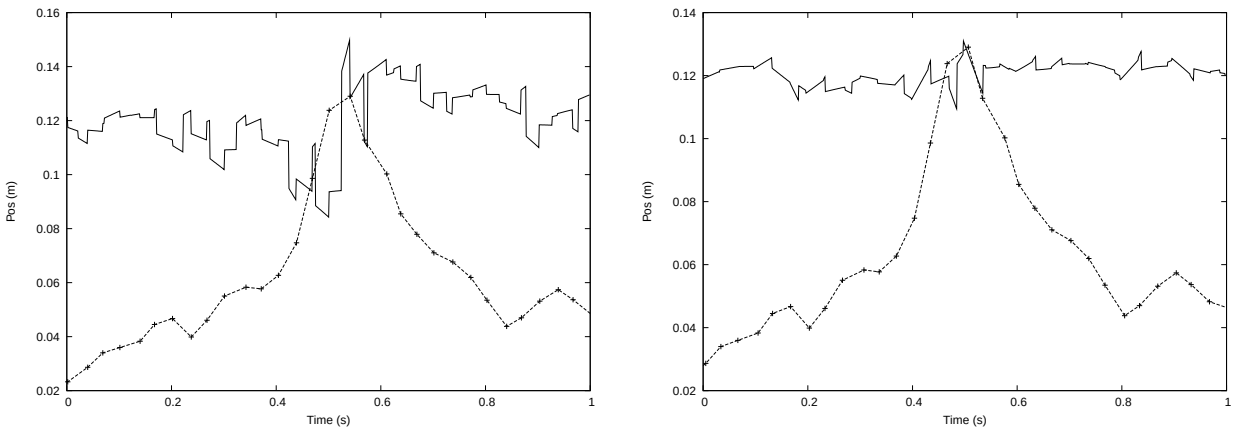


Figure 2: Violated simultaneity assumption in a concatenation: Measurements of one sensor are shifted by 30ms (left).

to the sensor, components that interpolate or extrapolate sensor data can be employed to provide a true Pull-Style interface. In that case, however, the accuracy information must be adjusted to reflect the uncertainty about the measured quantity using a motion model.

3.1 Computations involving multiple measurements

In Ubiquitous Tracking systems, measurements from multiple sensors frequently have to be combined in order to compute spatial relationships which cannot be sensed directly. An example is the concatenation pattern described above. In this case it is extremely important that the measurements to be combined are valid for the same point in time.

In figure 2, an example of unsynchronized measurements is given. A mobile camera, tracked by an external tracking system, is taking pictures of a marker, and the resulting pose is transformed into the coordinate frame of the external tracking system using the concatenation pattern. As the marker is not moving with respect to the tracking system, the resulting coordinates (solid line) should not change, even if the mobile camera (dashed lines) was moving. In the left image, the camera data had a delay of 30ms (about one frame at 30fps), and the result shows significant error at fast motions. In the right image, this delay was corrected using a kalman filter.

When combining multiple measurements, push and pull interfaces also have to be considered:

Pull+Pull The most simple case is a component combining data from multiple pull-style interfaces and providing the result as pull: When a request for data is received for a particular point in time, all inputs are queried for that time, the result is computed and returned to the caller.

Push+Push The events received on the different inputs of a data flow component usually have different timestamps and therefore in general cannot be combined.

Pull+Push When exactly one input of a component is using a push-style interface, the result of a combination can be returned again as push. When an event is received, the contained timestamp can be used to query the pull-interfaces for measurements at that moment. Then the result is pushed further down the data flow network. Other push-pull combinations again cannot be realized and some push inputs have to be converted to pull first.

3.2 Push-to-Pull Conversion

In order to realize combinations of multiple push sensors, conversion to a pull-style interface is necessary. Various strategies can be used to perform this conversion:

Buffering The simplest solution for performing the Push-to-Pull conversion is to always store the last measurement received from a sensor and return this to components further down the data flow network. However, this approach is only justified when the measured relationship is known to be static or moving slowly. Additionally, the accuracy taken from the stored measurement is likely to be invalid, especially if the buffered measurement is old and/or the tracked motion is fast.

Averaging If the measured relationship is known to be static, all received measurements can be averaged and the current result returned to the application. Compared to buffering, this has the advantage that measurement noise is reduced over time.

Simple Interpolation For faster motions, simple interpolation algorithms can be used to compute measurements valid for short time intervals around the last measurement, based on the last two or more measurements. Position can be inter-/extrapolated using first- or higher-order polynomials, and for rotations represented as quaternions, the SLERP [Kui02] algorithm is suitable.

Unfortunately, it is not easy to compute accuracy estimates for interpolated measurements, and the method is sensitive to noise, especially with increasing order of polynomials. Also, if the measurements have different accuracies, this cannot be regarded easily.

Kalman Filters In our system, an improved version of the setup described in [NWB+04], we use Kalman filters [Gel74] to convert push to pull interfaces. Despite its higher computational complexity, using a Kalman filter has many advantages: The filter can integrate measurements of varying accuracy and naturally provides an estimate for predicted values. Also, measurement noise can be reduced, however, that comes at the price of slower reaction to unexpected changes in direction.

It the tracked relationship is static, the filter can be used without a motion model, i.e. by leaving out the time update step, to provide a statistical averaging filter with correct accuracy estimates.

Another argument in favour of the Kalman filter is that it also can be used as the implementation basis of the multiple sensor fusion pattern. Of course, other statistical filters that employ a motion model may be used instead, such as particle filters.

3.3 Synchronized measurements

In many setups, there are tracking systems that can track multiple targets simultaneously. For example, an AR-Toolkit-like marker tracking system can detect multiple markers in one image, and even system with more than one camera can be synchronized in hardware. In such a case, it is correct to combine multiple measurements, as it is guaranteed that they are made at the same time, even if the sensors have a push-style interface.

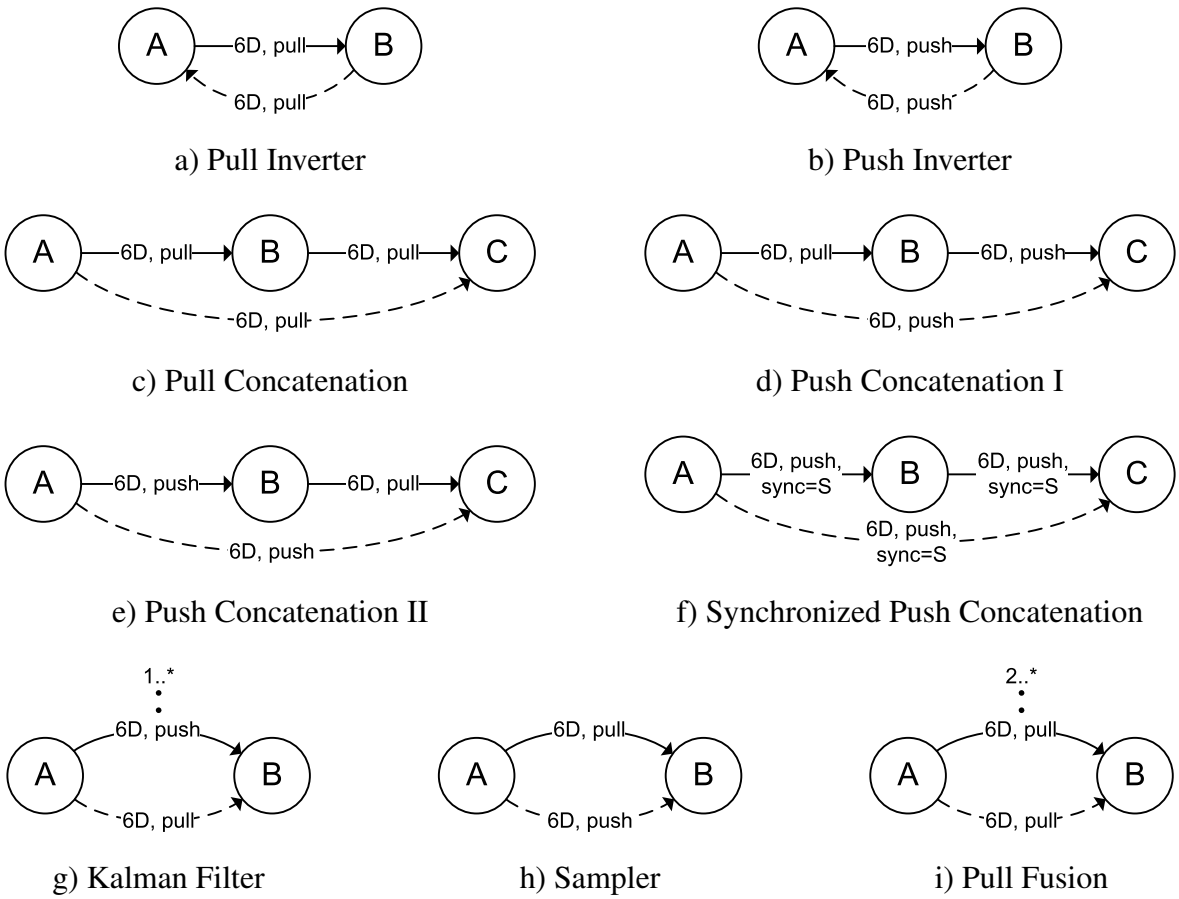
Compared to intermediate Kalman filters or other interpolators, computing the result directly from the raw measurements saves some computations and often gives higher accuracy, as no indirectly computed results are used. Therefore, the data flow generation software needs to take this into account. This is done by attributing every edge that represents a push component in the spatial relationship graph with the ID of the component that provided the original synchronization. For example, all measurements of markers made by a "Camera1" get a *SyncSource* attribute of "Camera1". Every time, a push component is inserted that has a push-style input, the *SyncSource* attribute is propagated to the output edge of the component.

Using this mechanism it is possible to decide whether two push-style measurements can directly be combined, even if other transformations have taken place in between. The data flow construction simply has to compare the SyncSource attributes of all push inputs, and, if they are the same, a *Synchronized Push* component can be inserted, where the SyncSource attribute is propagated from the input to the output edge.

4 Refined system of patterns

From the discussion in the previous section, it should be clear that a number of components is required to implement all scenarios that involve inversion, concatenation and multiple sensor fusion. As every component in the data flow graph can be mapped onto a spatial relationship pattern, this section presents a refined catalogue of patterns, where the initial patterns from section 2.1 are split into different patterns with different the interface types. To do that, an additional attribute is added to the edges of the spatial relationship graph to specify whether the interface is push- or pull-style. The graphical notation of the refined patterns is shown in figure 3.

Push Inverter The simplest component is the push-inverter, which inverts incoming measurements and pushes them further down the data flow graph. Of course, the timestamp is kept and the accuracy information is propagated to the result.



a) Pull Inverter b) Push Inverter

c) Pull Concatenation d) Push Concatenation I

e) Push Concatenation II f) Synchronized Push Concatenation

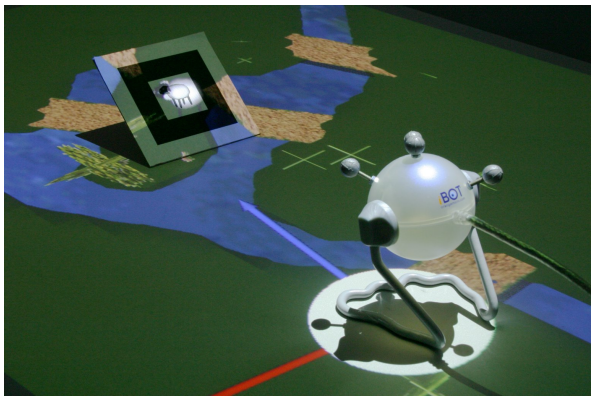
g) Kalman Filter h) Sampler i) Pull Fusion

Figure 3: The refined spatial relationship patterns regarding synchronization issues.

Pull Inverter and Pull Concatenation Similarly, the pull inverter, upon receiving a request to compute a result for a particular point in time, calls the input interface using the same timestamp, inverts the resulting measurement and sends it back to the caller. The same can be done in the pull concatenation, except that measurements from two inputs have to be requested.

Push Concatenation As explained in section 3.1 above, measurements in general can only be combined if at most one comes from a push interface. When a new event is received, the push concatenation component requests a measurement synchronous to the received one, multiplies the two and sends the result to the next component. Note that this pattern exists twice, with exchanged roles of the push and the pull input.

Synchronized Push Concatenation This component realizes the special case of concatenation of push communication, under the condition that both inputs have the same source of synchronization. The synchronized push concatenation simply waits until both inputs have sent measurements with the same timestamp, computes a result and pushes it on.



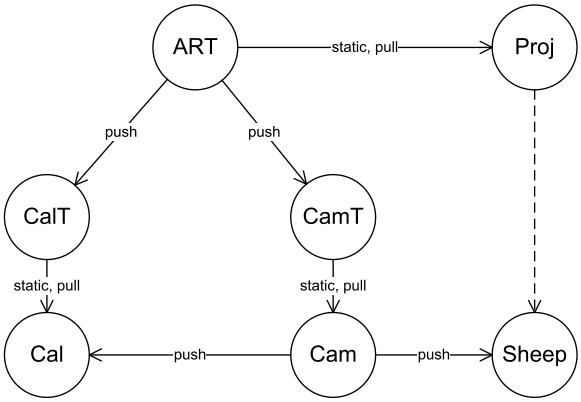


Figure 4: Picture and spatial relationship graph of a setup combining an ART DTrack system with marker-based tracking. All edges in the SRG represent 6D measurements.

Kalman Filter The Kalman filter is the swiss-army-knife component that serves different purposes. It is used to convert push- to pull-style communication by providing interpolation and extrapolation based on a motion model. When the result is known to be static, the motion model can be dropped and the filter performs statistically correct averaging over time. Additionally, when multiple push edges are available between two nodes in the spatial relationship graph, the Kalman filter can be used to statistically fuse the measurements.

Pull Fusion Multiple edges between two nodes in the spatial relationship graph can also be fused if they all have a pull-style interface. In this case, the fusion component also offers a pull interface. When a request for a measurement is received, all the inputs are queried (using the timestamp from the request), the result is computed by statistically combining all the measurements, based on their accuracy information, and returned to the caller.

Sampler When an application requests to receive measurement over a push-style interface, but intermediate results are computed as pull, it may be necessary to sample the computation at equally spaced points in time and send the results as events to the application. However, this should only be done at the end of a data flow network.

5 Example setup

To illustrate the data flow construction using the refined patterns presented in the previous section, an example setup using two different tracking systems is presented. The corresponding spatial relationship graph is given in figure 4. The setup consists of an infrared ART DTrack tracking system (*ART*), which tracks a target *CamT*, consisting of multiple retro-reflective balls, attached to a mobile camera *Cam*. For now, we will assume that the relationship between the camera target

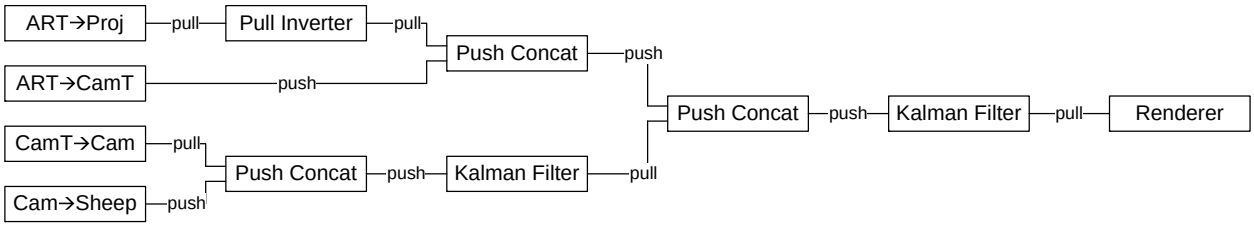


Figure 5: Data flow to project a virtual sheep onto a marker.

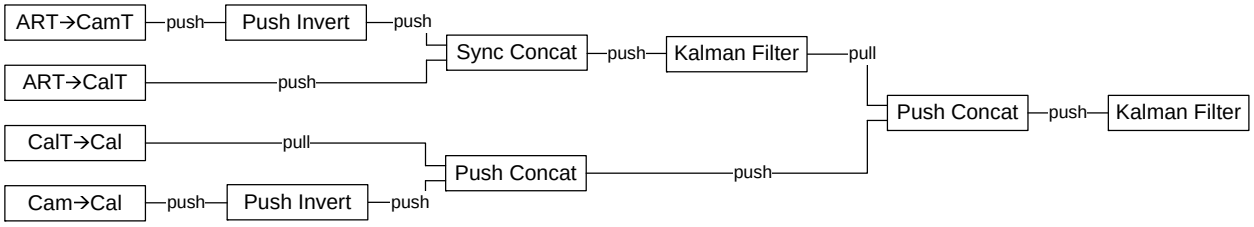


Figure 6: Data flow for computing the extrinsic camera parameters at runtime

and the camera, i.e. the extrinsic camera parameters, is known. Another DTrack-target *CalT* is attached to a visual marker *Cal* for calibration, and the transformation between the two is also known. The camera can track the visual markers *Cal* and *Sheep*. For visualization, the setup contains a projector *Proj*, that has been calibrated with respect to the coordinate frame of *ART*. The purpose of this small example system is to project a virtual sheep onto the marker *Sheep*, even if the transformation between the projector and the marker is not directly measured.

5.1 Dataflow

For rendering the virtual sheep onto the *Sheep* marker, the data flow depicted in figure 5 is constructed using only the inversion and concatenation patterns. Details about how patterns in the spatial relationship graph are detected and data flows are constructed, can be found in [PHBK06]. Conceptually, a directed path from the Projector to the Marker *Sheep* via *CamT* is constructed using the inversion pattern, and then the individual edges are combined using concatenation. Kalman filters are automatically inserted where direct concatenation is not possible. For simplicity, we assume that the calibration target is not visible, and therefore no sensor fusion is necessary.

In this example, the application has a pull interface, as it is rendering at maximum frame rate and needs predicted sensor data for the times when the image is rendered. This causes an additional Kalman filter to be added to the end of the data flow.

5.2 Autocalibration

In the data flow above, we have assumed that the extrinsic camera parameters, i.e. the *CamT→Cam* edge, are known. When the calibration target *Cal/CalT* is visible to both the camera and the ART tracker, the transformation *CamT→Cam* can be computed at run-time and used during tracking. The data flow from the previous section does not have to be modified except that the *CamT→Cam* component is exchanged by the data flow shown in figure 6.

The added data flow basically computes an alternative path from *CamT* to *Cam* via the *ART* and calibration target nodes. Note that in this case, as the computed transformation does not change, the Kalman filter component at the end does not use any motion model but only statistically combines all incoming measurements to provide higher accuracy.

6 Results and Conclusion

In this paper, a refined system of patterns was described for the construction of data flows for tracking in multi-sensor augmented reality setups. Starting from three simple patterns, these were expanded to take the aspects of loosely-coupled unsynchronized sensors into account, which is necessary to dynamically combine previously unknown sensors at runtime.

All the data flow components were implemented as services in our DWARF [BBK+01] middleware, and the setup described in section 5 was successfully built using an ART DTrack tracker and the AR Toolkit, connected to an iBOT firewire camera. The software system is similar to the one described in [NWB+04], extended by the concept of spatial relationship patterns [PHBK06].

Future Work For all but the most simple tracking problems, more than one data flow can be constructed to fulfil a request by an application. This results from the associativity of the concatenation when more than two edges need to be combined, from different orders of inversion and from the combination of two push sources, where it is not always clear which one to convert to a pull interface. It is still an open problem, which data flow to select in this case, but we will investigate this topic further, taking accuracy requirements of different applications into account.

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

References

[AB94] R. Azuma and G Bishop. Improving static and dynamic registration in an optical see-through hmd. In *Computer Graphics: Proceedings of SIGGRAPH 94*, 1994.

[Azu97] Ronald T. Azuma. A survey of augmented reality. *Presence: Teleoperators and Virtual Environments*, 6(4):355–385, 1997.

[BBK+01] Martin Bauer, Bernd Bruegge, Gudrun Klinker, Asa MacWilliams, Thomas Reicher, Stefan Riß, Christian Sandor, and Martin Wagner. Design of a componentbased augmented reality framework. In *International Symposium on Augmented Reality ISAR*, 2001.

[BSP+06] Martin Bauer, Michael Schlegel, Daniel Pustka, Nassir Navab, and Gudrun Klinker. Predicting and estimating accuracy of optical tracking systems. In *Proceedings of the 5th IEEE and ACM International Symposium on Mixed and Augmented Reality*, 2006. To appear.

[Gel74] Arthur Gelb. *Applied Optimal Estimation*. MIT Press, Cambridge, Massachusetts; London, 15th edition, 1974.

[Kui02] Jack B. Kuipers. *Quaternions and Rotation Sequences: A Primer with Applications to Orbits, Aerospace, and Virtual Reality*. Princeton University Press, 2002.

[NWB+04] Joseph Newman, Martin Wagner, Martin Bauer, Asa MacWilliams, Thomas Pintaric, Dagmer Beyer, Daniel Pustka, Franz Strasser, Dieter Schmalstieg, and Gudrun Klinker. Ubiquitous tracking for augmented reality. In *International Symposium on Mixed and Augmented Reality (ISMAR)*, Arlington, VA, USA, Nov. 2004.

[PHBK06] Daniel Pustka, Manuel Huber, Martin Bauer, and Gudrun Klinker. Spatial relationship patterns: Elements of reusable tracking and calibration systems. In *Proceedings of the 5th IEEE and ACM International Symposium on Mixed and Augmented Reality*, 2006. To appear.

[RS01] Gerhard Reitmayr and Dieter Schmalstieg. OpenTracker: An open software architecture for reconfigurable tracking based on XML. In *Proceedings of IEEE Virtual Reality*, pages 285–286, Yokohama, Japan, 2001.

[THS+01] Russell M. Taylor, ll, Thomas C. Hudson, Adam Seeger, Hans Weber, Jeffrey Juliano, and Aron T. Helser. Vrpn: a device-independent, network-transparent vr peripheral system. In *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 55–61. ACM Press, 2001.

[WB97] Greg Welch and Gary Bishop. SCAAT: Incremental tracking with incomplete information. *Computer Graphics*, 31(Annual Conference Series):333–344, 1997.