Distributed Tracking with Multiple Sensors for Augmented Reality

Martin Wagner
Technische Universität München, Fakultät für Informatik
Boltzmannstraße 3, 85748 Garching bei München, Germany
Tel./Fax: +49 (0) 89 289 -18230 / -18207
martin.wagner@in.tum.de

Abstract: Augmented Reality (AR) applications traditionally have been restricted to a small working volume, since current high-precision tracking technology is clearly limited in the range of operation. This paper presents ongoing work on creating a highly distributed and dynamic middleware for the aggregation of an arbitrary amount of location sensors.

It is designed to work in a peer to peer fashion to allow the integration of mobile setups into stationary ones or vice versa and the ad-hoc connection of two or more mobile setups at runtime. This allows AR to be used as an interfacing technology in Ubiquitous Computing environments.

The middleware is based on a formal model of attributed spatial relationship graphs that allow a uniform treatment of typical sensors used for AR. I present the distribution strategy and a generic distributed algorithm for selecting an optimal combination of available sensors, with optimality being defined differently by each application using the middleware. A discussion of intermediate results, observed problems and possible remedies concludes the paper.

Keywords: Augmented Reality, Distributed Computing, Ubiquitous Computing

1 Introduction and Related Work

Augmented Reality (AR) applications traditionally have been restricted to a small working volume, since current high-precision tracking technology is clearly limited in the range of operation. In recent years, more and more research groups have started to combine multiple tracking devices to extend the operation range of AR setups. However, these have typically been designed for a specific set of statically configured tracking devices. In this paper, I present my ongoing work towards creating a highly dynamic software architecture that allows the integration of new mobile or stationary trackers into existing setups at runtime. It is based on several previously published papers [BBK+01, NWB+04, NWP+03, WK03]. A much extended version will be part of my dissertation.

1.1 Merging Ubiquitous Computing and Augmented Reality

Augmented Reality focusses on enriching the user's experience of the real world by mixing in representations of virtual data. Most often, Head Mounted Displays (HMDs) are used to provide visual augmentations, however, other sensations such as sound can be used as well. Today's AR setups usually have a working volume that is restricted to the extent of a particular tracking technology. The massive amount of sensing and rendering technology involved have only recently allowed mobile AR to get usable [FMHW97, PT01, RS04]. Still, the tracking technology used in such setups is either a single one, most often DGPS, or a static predefined combination of a small number of trackers.

Mark Weiser envisions Ubiquitous Computing (UbiComp) [Wei93] as a "calm" technology where computing devices blend into the background, being available to users without them being aware of it. To let this vision come true, massive amounts of tiny sensors have to be deployed both in the environment and on the users themselves, all being networked and thus providing a maximum of contextual information to actually let the environment do what the user *wants* instead of what she or he explicitely *tells* the system by arcane commands given by a user interface as known today.

Using AR as an interfacing technology for UbiComp applications seems a natural fit – the user interface could be interwoven with the real world, thus letting Weiser's vision come true. However, several hard problems have to be solved to realize this idea, ranging from system architecture aspects to security and privacy concerns. In this paper, I focus on finding solutions to the subproblem of sensors used for location tracking by means of providing a suitable middleware.

Both AR and UbiComp applications rely heavily on positional information. However, the requirements differ significantly. Sensors in AR applications are scarce and usually deliver very precise position and orientation data with high update rates. This is necessary to let the user immerse into the augmented world. If data is inaccurate or delayed, the illusion of the virtual information being real gets lost. In contrast, UbiComp sensors are very numerous and sense many quantities. According to Beigl et al. [BKZD04], motion sensors are used in most UbiComp applications, but the typical ball or mercury switches or accelerometers can not be used directly to get spatial relations between objects that would be necessary to enable AR applications.

The goal of my work is to define a system that fulfills the realtime and accuracy requirements of AR applications in addition to the distribution and large-scale requirements of typical UbiComp applications. The system is based on a formal model described in section 2 that allows a uniform description of sensor networks. The implementation concept I propose in section 3 is highly distributed, thus allowing ad-hoc connections of multi-sensor setups at runtime. Some results of the actual implementation based on our *Distributed Wearable Augmented Reality Framework (DWARF)* [BBK+01] are described in section 4.

1.2 Related Work

The idea of combining multiple sensors for location tracking is not new as such. Several software frameworks exist, the most notable being OpenTracker [RS01] and VRPN [THS+01]. Both pro-

vide an abstraction layer between the sensors and the application and offer the facility to integrate many commercial tracking devices.

Hightower et al. [HBB02] propose a layered software model, the *Location Stack*, to structure the sensor fusion problem in UbiComp environments similar to the OSI network layer model. They apply this abstract model to the Easy Living project [BMK+00] which is in scope similar to my work on bringing AR to UbiComp environments. However, the sensors they actually deal with are clearly limited in accuracy, update rate and degrees of freedom. The architecture presented in this paper aims at overcoming these restrictions.

Hallaway and Hoellerer [HHF04, HHTF01] put much more focus on integrating sensors suitable for AR. They put much thought into correctly predicting the overall accuracy and use of this knowledge to adapt the resulting system's look and feel. The architecture they describe is rather static and does not allow the integration of new mobile sensing devices at runtime.

2 Formal Model: Ubiquitous Tracking

The formal model described in this section forms the base of the sensor fusion middleware. It has been specified in collaboration with Joseph Newman from TU Wien. We call this model and the efforts built on top of it *Ubiquitous Tracking (Ubitrack)*. A detailed description can be found in a previous publication [NWP+03] Here I present the key concepts.

The goal of the Ubitrack model is to provide, at any point in time, an optimal estimate of the spatial relationships between arbitrary objects. How optimality is defined, depends on the specific application interested in a particular relationship.

2.1 Spatial Relationship Graphs

As proposed by Brumitt et al. [BKMS00], we use a directed graph to model spatial relationships. Real or virtual objects are modeled as nodes in this so-called *SR Graph*, whilst spatial relationships are represented as edges. For the sake of clarity, we introduce three different graphs:

Real World Relationships This ideal model of the spatial arrangement in the world is the final goal we aim at. The graph is transitive, symmetric and reflexive, i.e., if we know the relationships between objects A and B and between B and C, we also know the relationship between A and C. All relationships stored in the graph's edges are without error. The spatial relationships get stored by an attribution scheme $\mathbf{W}$, mapping every pair XY of two nodes of the graph's node set N onto a function w_{XY} describing the spatial relationship between the objects X and Y over time. Thus, $\mathbf{W}$ can be written as follows:

$$\mathbf{W} : (\Omega = N \times N) \rightarrow w, \text{ where } w : D_t \rightarrow \mathbb{R}^{4\times4} \tag{1}$$

D_t is the source time domain, mapped by w onto the target spatial relationship domain. This definition matches the output of common tracking devices, yielding spatial relationships for different

points in time. An example of a simple real world relationship graph consisting of three nodes is shown in figure 1(a).

Measured Relationships Unfortunately, only an omniscient observer has all data of the real world SR graph. Most often, we only have a few measurements of relations between some objects. Therefore, we introduce a measurement graph containing all given knowledge from sensors in the system. Again, we define an attribution scheme, $\mathbf{P}$, and functions p_{XY}:

$$\mathbf{P} : (\Phi \subseteq N \times N) \rightarrow p, \text{ where } p : D_t \rightarrow \mathbb{R}^{4\times4} \times \mathcal{A} \tag{2}$$

Note that p is a multivalued function, yielding not only the spatial relationship between two objects at a given time, but also a set of *attributes* $\mathcal{A}$ describing the quality of the underlying measurements. Thus, descriptions of parameters such as the accuracy according to a specific error model, the monetary cost of a particular measurement, the update rate between subsequent measurements or the delay between the physical state and the availability in the system (lag time) can all be encapsulated in the measurement graph. Figure 1(b) shows the graph of a three object example with some measurements.

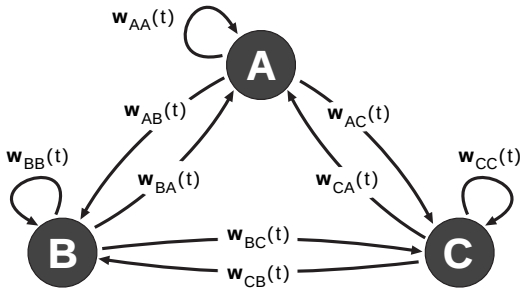
Inferred Relationships The measurement graph just introduced is, in general, neither symmetric nor transitive. To make any use of aggregated data of multiple sensors, thus allowing our system to overcome the usual restrictions of single sensor systems, we have to *infer* spatial relationships based on our knowledge of the world's physical properties. These inferences get modeled in a third graph, which is actually the only graph of practical relevance: the inference graph. Its definition is similar to the measurement graph's:

$$\mathbf{Q} : (\Psi \subseteq N \times N) \rightarrow q, \text{ where } q : D_t \rightarrow \mathbb{R}^{4\times4} \times \mathcal{A} \tag{3}$$

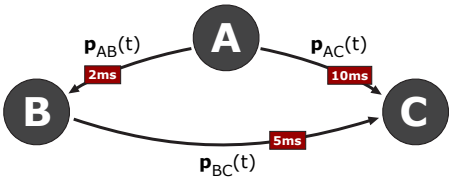
We can construct the inference graph by starting with a copy of the measurement graph. We subsequently use data stored in the graph in combination with knowledge of the world's physical properties in order to infer new spatial relationships. For example, if we know some relationship between objects A and B, the time they were taken and the accuracy of these measurements, we can use a movement model to configure a Kalman filter [WB01] that can provide an estimate of the spatial relationship between objects A and B at any point in time. We model this by adding a new edge AB and the corresponding function q^e_{AC} to the graph. q^e_{AB} gets derived from the function p_{AB} and the Kalman filter equations. An example constructed from the measurement example in the last paragraph with lag times as attributes and some new inferred edges q^e is shown in figure 1(c).

2.2 Finding Optimal Paths: The Evaluation Function

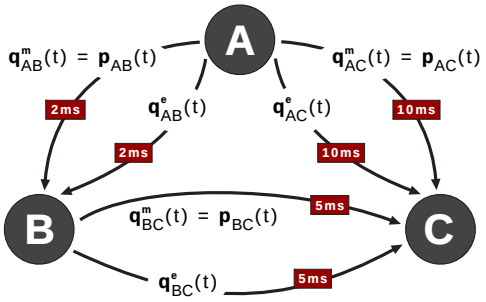
To serve multiple applications with varying requirements simultaneously, we introduce a so-called *evaluation function* that works on the attributes of the SR graph's edges. The evaluation function maps a path onto a real positive number, with the additional convention that a lower number signifies higher suitability for the application giving the specific function.



(a) Real world relationships



(b) Measured relationships



(c) Inferred relationships

Figure 1: Examples of spatial relationship graphs

This results in a generic algorithm for choosing an optimal inference between two arbitrary objects X and Y: First, we search all paths between X and Y, second, we compute the value of the evaluation function for every such path and select a path with the lowest value, i.e. the "best" path for the application corresponding to the current evaluation function. To make this approach possible, we have to provide algorithms for three things:

Transitivity of spatial relationships: Given the spatial relationship between object A and B and the relationship between B and C, compute the relationship between A and C. This is fairly simple, for the 4×4 homogeneous matrix representation of spatial state commonly used in AR, this is done by a simple matrix multiplication.

Transitivity of attribute sets: Given the attribute sets of edges AB and BC, compute the attribute set of the edge AC representing the spatial relationship derived in the previous step. This can be a much more complex task. In the case of attributes describing the error of measurements or inferences this leads to solving equations for error propagation, prerequisiting a careful choice of a suitable error model. It may be possible that the application requesting the relationship AC is not interested at all in a subset of the attributes. Then the members of this subset need not be computed explicitely, it suffices if they are somehow taken into account by the evaluation function.

Symmetry: Given a directed edge AB, compute the inverse edge BA. Again, for the spatial

relationship this is fairly simple, in case of the homogeneous matrix it leads to a simple inversion. For the attribute set, we can make essentially the same observations as with the transitivity of the attribute set – inverting edges is a complex task. If we want to reflect the numerical errors introduced by matrix inversion in the attribute subset describing the error of the inference, things get even more complex.

If these three algorithms are given for the chosen spatial state space and attribute set, we can use the formal model to successively compute spatial relations between all pairs of nodes which have a single or multiple paths as connection. Eventually, this leads to complete subgraphs of the inference graph for every connected component in the measurement graph. Note, however, that this will seldom occur in practice, as only transitivity paths that are needed by some applications will be set up. Once such a path is set up, it can be used to satisfy other applications' requests as well.

2.3 Pathwise and Edgewise Evaluation Functions

In general, the evaluation function is defined to be applied to complete paths. Thus, it would be necessary to first compute *all* paths between two nodes, and then apply the evaluation function to all of them. This *pathwise evaluation* can be an extremely costly task, as the number of paths is exponential in the number of nodes.

The problem gets much simpler if the evaluation function can be applied *edgewise*, i.e. if

$$e(p) = \sum_{i=1}^{n-1} e(N_i N_{i+1}) \tag{4}$$

with p being the path consisting of the nodes $N_1 N_2 \ldots N_n$. If the evaluation function is structured like that, we can assign every edge a weight based on its evaluation function's value and then use a standard shortest path algorithm to compute an optimal path from N_1 to N_n.

3 Distributed Implementation Concept

Up to now, we have a formal model that allows us to uniformly describe multi sensor setups. However, we have not yet discussed the issues arising when this model should be transformed into a working implementation. In this section, I present a concept how the Ubitrack model can get implemented in a fully distributed fashion.

The key requirements for every implementation, be it distributed or centralized, are threefold. First, a representation of the complete SR graph must be stored. Second, it must be possible to compute an optimal path between all nodes in a connected component. Optimality is defined by an evaluation function provided by the application interested in the spatial relationship between the objects represented by the nodes at the path's ends. Third, it must be possible to add or remove nodes and edges at runtime, thus changing the graph's topology, in order to allow sensors and/or locatables to be added or removed and to allow new inferences to be made at runtime, leading to new edges.

If an implementation fulfills all these requirements, it can serve as an abstraction layer between the sensors and the application. In addition, multiple applications can be served simultaneously, as each can provide its own evaluation function to the implementation.

3.1 A Peer to Peer Approach for Integrating Mobile Setups

Reconsidering the vision of ubiquitous computing, an implementation should allow the integration of a vast amount of sensors, both mobile and stationary. Given the scenario of an intelligent environment equipped with sensing and display technology and enabling its users to naturally interact with the infrastructure, a centralized approach seems reasonable. A central server could hold the complete representation of the SR graph, fulfilling all search requests and eventually telling each application which sensor data it should aggregate.

The situation gets different if we assume that the users bring their own equipment such as cameras on their cell phones or wearable sensors woven in clothes into the stationary environment. I propose a fully distributed peer to peer approach. Although it is much more difficult to design and implement, I think the following advantages make a clear point for going in this direction:

Allowing ad-hoc connections between mobile setups: If we treat every network node uniformly, we get ad-hoc connectivity between mobile setups for free. Thus, the computing infrastructure gets truly ubiquitous, even allowing two users meeting on the green field to use the sensors of their counterparts without any additional infrastructure.

Mobile setup is self-contained: If a user has a mobile setup with more than a single sensor, he can get sensor fusion results without the involvement of a central infrastructure. In addition, the user has full control over the data his sensors gather, he can use it without giving it to anyone else. This enables privacy and security mechanisms.

No single point of failure: In a centralized approach with a server in the stationary environment, a failure of the infrastructure renders all mobile and stationary sensing technology useless. This may lead to severe implications in case of an emergency, when sensors might be used to compute optimal escape routes. In the proposed distributed approach, parts of the overall sensor network can still be used, especially self-powered mobile setups.

3.2 Distribution Strategy

The primary goal of the choice of distribution scheme was to minimize the replication of data, thus reducing consistency problems. Remember that every edge in the SR graph represents some spatial relationship between two objects in the real or virtual world. Thus, some software component, possibly getting data from some hardware sensor, corresponds to every edge. In consequence, the edges are distributed among all nodes of the underlying computer network. An example with four network nodes is shown in figure 2.

The distribution scheme now works as follows: every network node constructs the subgraph of the SR graph consisting of all edges (and their adjoint SR graph nodes) representing software

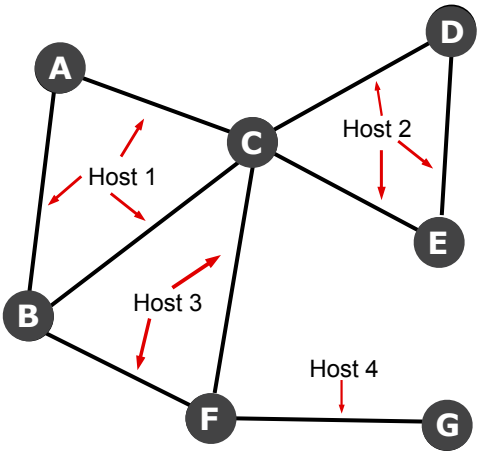


Figure 2: Logical SR graph. Every edge represents spatial data which is obtained by software and/or hardware components distributed among different hosts.

components running locally on the network node. In addition, every network node scans the network for other network nodes having edges with SR graph nodes that are also available locally. Once these are found, the SR graph nodes get duplicated and edges with zero weight are inserted between the SR graph node copies. Figure 3 shows this process for the example setup.

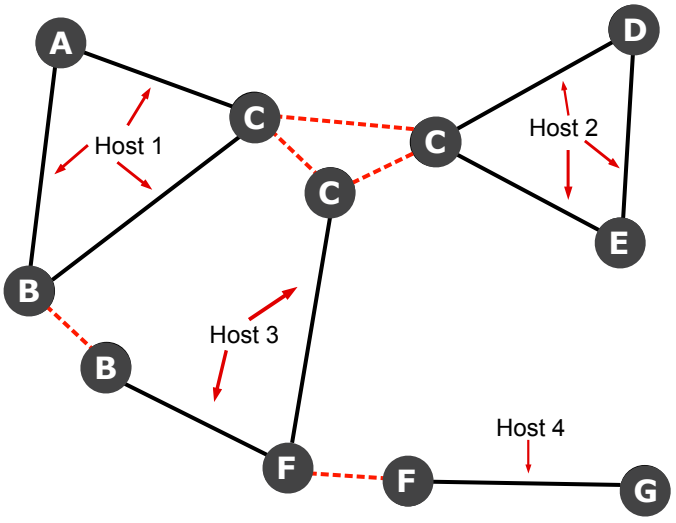


Figure 3: Distributed storage of SR graph. The dashed red edges have zero weight in a path search.

Finding adjoint SR graph nodes can be done with standard service discovery techniques, mostly based on directory services or broadcasts.

3.3 Searching Optimal Paths

As mentioned above, a pathwise evaluation function is computationally expensive. This is even more true for distributed setups, therefore from now on I assume that we are given an edgewise evaluation function. In consequence, we have to do a shortest path search in the distributed representation of the SR graph.

Two classes of shortest path algorithms exist: Dijkstra's algorithm [Dij59] works in a greedy fashion and iterates over the nodes, whereas the Bellman-Ford algorithm iterates over the path length from source to destination. The latter approach is suited much better to a distributed implementation based on message passing.

Lynch [Lyn96] presents an asynchronous variant of the Bellman-Ford algorithm, including an analysis of its time and message complexity. This algorithm has been the basis of the distributed path search described in the remainder of this section. It has been used as the primary routing algorithm in the ARPANET. Indeed, finding routes in computer networks is conceptually very similar to the problem of finding optimal paths in a SR graph.

Prerequisites For the algorithm to work, some assumptions must be made that in general do not harm the flexibility of the formal model discussed above. First, an edgewise evaluation function must be given. Second, every node in the SR graph must have a globally unique identifier. Usually this can be ensured by a concatenation of the network node's globally unique MAC address with a local identifier. In case of sensors detecting the spatial relationships between some unidentifiable objects, some bootstrapping mechanism has to be provided. This is ongoing work, for now we assume that every sensor and locatable can be identified uniquely. Third, it must be possible to detect the network node that hosts a specific SR graph node. This can be done by standard service discovery techniques which are beyond the scope of my work. In the current implementation I rely on MacWilliams' work on the DWARF middleware [Mac03]. Fourth, we assume the SR graph to be undirected, which can be assured by the symmetry algorithm required in section 2.2.

Distributed algorithm The distributed path search starts at the source node in the SR graph, thus at some network node hosting a copy of this object, which is found by service discovery. The application requesting an optimal path has to provide an evaluation function operating on the edges' attribute set. Conceptually, the code for this function has to be passed along the network nodes the path search proceeds, in practice, an identifier and some parameters of a predefined set of evaluation functions should suffice.

The path search gets assigned a unique *search ID*, and each SR graph node has to store the so far minimum distance to the source node of this specific search request. It gets initialized with a value of 0 for the source node and $+\infty$ for every other node, i.e. if a node gets a search ID for the first time, it assumes its current distance value for this ID to be $+\infty$.

The start node now passes a search request including the target node, a search ID, the evaluation function and its current minimum distance to the start node along all adjoint edges. Subsequently, every node getting such a message computes the evaluation function's value of the incoming edge. If the sum of the incoming minimum distance and this value is smaller than the currently stored minimum distance value, the local minimum distance gets updated and the node at the other end of the incoming edge gets stored as predecessor on the shortest path to the source node.

Messages with the new minimum distance value are passed to all adjoint edges except the one being used for the incoming message. The algorithm terminates if the target node is reached. It sends back acknowledgement messages including the new minimum distance to the source node.

If acknowledgement messages have been passed back as response to messages triggered by an updated minimum distance to a node via all edges, the node sends an acknowledgement message back to the node the search request came from. These messages can also be used to transfer the SR graph nodes the optimal path consists of.

Complexity Analysis The modifications made to the original algorithm as discussed by Lynch just increase the constant factors of the single steps' time and message complexity, thus the time complexity is still $O(n^{n+1}(l+d))$ and the message complexity is $O(n^n m)$, with n being the number of nodes and m being the number of edges in the SR graph, d being the time to pass a single message and l being the local processing time.

A synchronized variant of this algorithm brings down the time complexity to $O(n-1)$ and the message complexity to $O((n-1)m)$. However, the constant factor introduced by synchronizing the distributed processes may be prohibitively large for real world setups.

4 Current Status

Currently, a first implementation [Bey04] is running on a small scale. It is based on the DWARF middleware [BBK+01] and extends its functionality with the distributed path search just described. As such, it currently uses SLP [GPVD] without directory agents, thus restricting service location to a single broadcast subnet.

We have tested the path search on setups consisting of up to five network nodes and 50 SR graph nodes and achieved path search running times in the range of a second. We assume that the topology of the SR graph and the attributes describing the measurement and inference qualities do not change often compared to the spatial relationships of tracked objects. Therefore, we compute a path only once and consequently set up a *data flow component* that constantly aggregates the spatial data along the precomputed path. To handle infrequent changes in the graph's topology and the attributes, we periodically repeat the optimal path search for all applications still using a queried relationship. Thanks to the two-step communication setup, runtime behavior of our implementation has an efficiency comparable to that of static setups, being fast enough for AR applications, with a low probability of delivering suboptimal results due to the periodic recalculations.

5 Conclusions and Future Work

The formal Ubitrack model seems to be a viable solution to the multi sensor fusion problem arising when making AR applications mobile. It allows the modeling of almost arbitrary setups. The distributed implementation concept discussed in this paper allows a peer to peer solution, letting mobile setups participate seamlessly in Ubitrack setups. In addition, it allows every user to exert full control over all sensors he is carrying with him, a very important issue from a privacy and security viewpoint.

We are currently working on several areas necessary for large scale deployment of the Ubitrack ideas. First, the exponential running time of the path search makes it necessary to consider opti-

mization possibilities. Similar to the Internet, where routing protocols have exponential running time and work in practice nevertheless, we can incorporate knowledge about the real world to reduce the number of nodes that need to be considered for a specific search request. For example, we can assume most queries to be rather local, i.e. restricted to a single room. The number of nodes inside such a small entity is clearly limited. For large scale queries, it seems reasonable to aggregate several nodes into *supernodes*, thus introducing a hierarchy. In addition, we might use contextual information to further reduce the search space, by just taking into account sensors having data matching the current context.

Second, we consider how to identify so-called *anonymous* nodes, i.e. things that are observed by some tracker without any knowledge of their identity. This case is of particular use for natural feature based optical trackers.

Third, the validity of the Ubitrack formal model needs to be proven by showing how existing multi sensor setups can be modeled with it, and that an implementation of the formal model based on the distributed implementation concept is indeed as fast in its runtime behavior as static setups.

Acknowledgements

This work was supported by the Deutsche Forschungsgemeinschaft, project DySenNetz (KL1460/1-1). I thank Gudrun Klinker, Joseph Newman, Dieter Schmalstieg, Asa MacWilliams, Martin Bauer, Thomas Pintaric, Dagmar Beyer, Daniel Pustka and Franz Strasser for valuable discussions on the formal Ubitrack model and a joyful collaboration on trying to get things work.

References

[BBK+01] M. Bauer, B. Bruegge, G. Klinker, A. MacWilliams, T. Reicher, S. Riss, C. Sandor, and M. Wagner. Design of a component-based augmented reality framework. In *Proc. of 2nd IEEE and ACM International Symposium on Augmented Reality (ISAR 2001)*, New York, NY, USA, October 2001.

[Bey04] D. Beyer. Construction of decentralized data flow graphs in ubiquitous tracking environments. Master's thesis, Technische Universität München, 2004.

[BKMS00] B. Brumitt, J. Krumm, B. Meyers, and S. Shafer. Ubiquitous Computing and the Role of Geometry. *IEEE Personal Communications*, pages 41–43, October 2000.

[BKZD04] M. Beigl, A. Krohn, T. Zimmer, and C. Decker. Typical sensors needed in ubiquitous and pervasive computing. In *Proc. of First International Workshop on Networked Sensing Systems*, Tokyo, Japan, 2004.

[BMK+00] B. L. Brumitt, B. Meyers, J. Krumm, A. Kern, and S. Shafer. EasyLiving: Technologies for intelligent environments. In *Proc. of 2nd Intl. Symposium on Handheld and Ubiquitous Computing*, 2000.

[Dij59] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[FMHW97] S. Feiner, B. MacIntyre, T. Hoellerer, and A. Webster. A touring machine: Prototyping 3D mobile augmented reality systems for exploring the urban environment. In

Proc. of 1st International Symposium on Wearable Computers (ISWC), pages 74–81, Cambridge, MA, USA, October 1997.

[GPVD] E. Guttman, C. Perkins, J. Veizades, and M. Day. Service location protocol, version 2. `http://www.ietf.org/rfc/rfc2608.txt`.

[HBB02] J. Hightower, B. Brumitt, and G. Borriello. The location stack: A layered model for location in ubiquitous computing. In *Proc. of 4th IEEE Workshop on Mobile Computing Systems & Applications (WMCSA 2002)*, pages 22–28, Callicoon, NY, June 2002.

[HHF04] D. Hallaway, T. Hoellerer, and S. Feiner. Bridging the gaps: Hybrid tracking for adaptive mobile augmented reality. *Applied Artificial Intelligence, Special Edition on Artificial Intelligence in Mobile Systems*, 25(5), July 2004.

[HHTF01] T. Hoellerer, D. Hallaway, N. Tinna, and S. Feiner. Steps toward accommodating variable position tracking accuracy in a mobile augmented reality system. In *Proc. of 2nd International Workshop on Artificial Intelligence in Mobile Systems (AIMS '01)*, pages 31–37, 2001.

[Lyn96] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, Inc, 1996.

[Mac03] Asa MacWilliams. Self-extending systems for context-aware mobile computing. In *Doctoral Symposium at the International Conference on Software Engineering*, Portland, Oregon, USA, May 2003.

[NWB+04] 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 *Proc. of International Symposium on Mixed and Augmented Reality*, Arlington, VA, USA, Nov. 2004. To appear.

[NWP+03] J. Newman, M. Wagner, T. Pintaric, A. MacWilliams, M. Bauer, G. Klinker, and D. Schmalstieg. Fundamentals of ubiquitous tracking for augmented reality. Technical Report TR-188-2-2003-34, Vienna University of Technology, 2003.

[PT01] W. Piekarski and B. H. Thomas. Tinmith-evo5 a software architecture for supporting research into outdoor augmented reality environments. Technical report, Wearable Computer Laboratory, University of South Australia, December 2001.

[RS01] G. Reitmayr and D. Schmalstieg. OpenTracker: An open software architecture for reconfigurable tracking based on XML. In *Proc. of ACM Symposium on Virtual Reality Software & Technology (VRST)*, Banff, Alberta, Canada, 2001.

[RS04] G. Reitmayr and D. Schmalstieg. Collaborative augmented reality for outdoor navigation and information browsing. In *Proc. of Symposium on Location Based Services and TeleCartography*, Wien, Austria, 2004.

[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 & Technology (VRST)*, Banff, Alberta, Canada, 2001.

[WB01] G. Welch and G. Bishop. Course 8 – An introduction to the Kalman filter. In *SIGGRAPH 2001 Courses*, 2001. `http://www.cs.unc.edu/~tracker/ref/s2001/kalman/index.html`.

[Wei93] M. Weiser. Hot topics: Ubiquitous computing. *IEEE Computer*, October 1993.

[WK03] M. Wagner and G. Klinker. An architecture for distributed spatial configuration of context aware applications. In *Proc. of 2nd International Conference on Mobile and Ubiquitous Multimedia*, Norrköping, Sweden, 2003.