Technische Universität München
Fakultät für Informatik

3rd Joint Advanced Summer School 2005
Course 3: Ubiquitous Tracking for Augmented Reality

Prof. Gudrun Klinker, Ph.D.
Martin Bauer
(Eds.)

April 2005

Contents

This document is a report on the results of the course "Ubiquitous Tracking for Augmented Reality" that was held during the 3rd Joint Advanced Students School (JASS05) in St. Petersburg, 30.3.-9.4.2005.

The main outline of the course was developed by Prof. G. Klinker together with M. Bauer, both from Technische Universität München; significant contribution came also from the russian side from Prof. Boris Kudryashov, State Univ. of Aerospace Instrumentation St. Petersburg and Prof. Ivan V. Andronov, St. Petersburg State University. As a guest we were also lucky to have Prof. Alexander Pastor, St. Petersburg State University with us for one day.

We would like to thank all of our students: Troels Frimor, Georgi Nachev, Basti Grembowietz, Katharina Pentenrieder, Kirill Yourkov, Alexander Chuikov, Christian Buckl, Alexey Minin, Irina Bobkova, Denis Bessonov, and Gordana Stojceska for their participation and the fruitful discussions before, during and after the course. Special thanks to Alexey and Irina for showing us around in SPb helping us to decypher the menus and to order food and beer.

Thanks also to the Directors of the 3rd Joint Advanced Students School, Prof. Dr. Yu. Matiyasevich and Prof. Dr. E. W. Mayr together with Dr. V. Ganzha for organizing the course, and the sponsoring companies and institutions, the Bavarian Ministry of Economics, Siemens AG and Infineon, for providing the necessary funding.

Garching, May 2005
Martin Bauer, Prof. Gudrun Klinker

1 Introduction and Concepts of Ubiquitous Tracking

Troels Frimor, Technische Universität München

In providing the possibility for wide area Augmented Reality applications, combining sensor information into complicated tracking setups becomes more and more an issue. With Ubiquitous Tracking a formal framework is introduced which addressees the problems with describing these setups. The framework defines three different kinds of spatial relationship graphs to create an abstract view of the relations between objects and sensors. This gives such a general and flexible way to describe spatial relations that it can be used for all kinds of setups also from existing applications.

1.1 Introduction

First some related topics will shortly be explained to give a better understanding of what *Ubiquitous Tracking* is and why we need a framework to describe our tracking setups. Then the formalism behind *Ubiquitous Tracking* will be described with accompanying examples to show how it is used. These introductions are meant to give a common basis for discussions of tracking for *Augmented Reality* and it associated ptroblems.

1.2 Related Concepts

To better grasp some of the aspects of *Ubiquitous Tracking* some of the related concepts will be introduced shortly here with special focus on what the connection to our topic is.

1.2.1 Augmented Reality

Augmented reality (*AR*) is an emerging technology in which the real and virtual world are combined so that the user will be able to sense the real world with additional virtual information superimposed. This can be done in many ways but is classically done via a head-mounted-display. Several different definitions of *AR* exist, where the most commonly used is defined by Azuma with the following characteristics:

- Combines real and virtual
- Interactive in real time
- Registered in 3D

This definition is created so that simple applications like the overlapping of virtual and real in movies, does not fall under the *AR* category, since they are not interactive in real time.

The usability of *AR* is broadened out on many fields like medical, maintenance, production and games. An example of the latter is the SHEEP[1] game, which is designed to explore some aspects of *AR*, like user interfaces.

In the game several virtual sheep run around in an virtual landscape which is projected onto an table. Another sheep, a real physical model, has infrared markers attached to it, thus allowing the system to track it and register it with the virtual environment. The spatial relationships with the other virtual sheep can then be determined, and the virtual sheeps will try to follow the model as it is moved around on the table. Laptops and PDAs can be used in connection with the game as hand-held see-though displays. On fig.2.8 this setup can be seen where the laptops are tracked with infrared markers.

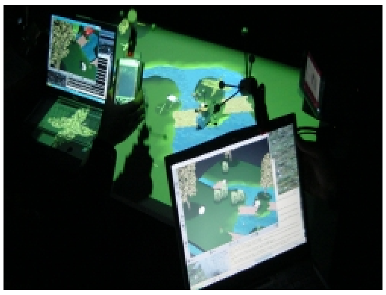


Figure 1.1: SHEEP [1]

One of the biggest fields of research regarding *AR* is in the medical world. On fig. 1.2 an example application, Medarpa (MEDical Augmented Reality for PAtients[?]), can be seen which deals with assistance for medical surgery and especially keyhole surgery. This is based on the idea of making the surgeon able to see through and inside the patient so he would only need to make small holes on the outside to enter the instruments, thus minimizing harm on other than the actual area that need surgery.
The system seen, uses a mounted see-through screen which is registered with both patient and instruments, thus the surgeon will even be able to see the instruments inside of the patient.

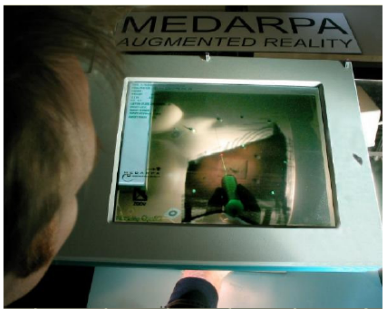


Figure 1.2: MEDARPA [?]

Both of the above mentioned examples are based on small local areas with just a few tracking mechanisms. If wanting to create a full scale *AR* application, like an augmented hospital, tracking the patient and even the instruments around the entire hospital would be necessary, also in the hallways, elevators and so on. Throughout the hospital many different tracking mechanisms would be used though, and combining them all would become a very big task. With the framework described later in this paper, subparts of the system could be defined and reused in different locations and even in other hospitals.

1.2.2 Tracking Devices

Many different ways of tracking are possible and the development of tracking devices keeps on adding new possibilities. A typical tracking mechanism for *AR* applications uses a couple

[1]Shared Environment Entertainment Pasture.

of infrared cameras and small infrared-reflective balls attached to the objects of interest. This setup has several drawbacks[2] though and most often other methods have to be considered.
Of many different tracking devices, the following are the most widely used:

- Cameras attached to the object of interest (Inside-out) or observing the object (Outside-in)
- Gyroscopes attached to the object to measure changes in orientation
- Accelerometers attached to measure changes in linear acceleration
- Devices measuring magnetic fields where a central unit produces a strong magnetic field and smaller devices on the objects of interest then measure this field.

In general the tracking methods are classified according to their different characteristics, which also tells something about their advantages and limitations. The characteristics include:

- *Physical medium* the sensors use. Is the devices mechanically connected, using inertia, using light, ...
- *Measured values* of the sensors, this can be 2D pixel value, time of flight (sound), angle between joints, ...
- *Accuracy* of the sensors. Can the noise be descibed easily (Gaussian?).
- *Update rate*: of new measurements.

To really take advantages of the tracking devices they have to be combined into hybrid systems. That is: combining the different sensors to take advantage of their different properties and/or measurements for more precise results. Another issue is how well the individual tracking mechanisms fit in different environments. For example whether it is possible to make it sterile in clinical environments or use magnetism where there are magnetic materials in the background.
Hybrid systems quickly become necessary when more than a simple desktop *AR* application is needed but for wide area systems they can be very difficult to describe, maintain and reproduce. The *Ubiquitous Tracking* framework described later deals with these issues, having hybrid systems as a natural part of the system description.

1.2.3 Ubiquitous Computing

The world of computing can be divided into three generations where the first generation was the *mainframe computing*, then the *personal computing* and now the *ubiquitous computing era* is emerging.

Ubiquitous Computing is the intention that computers should be integrated into our everyday lives in an invisible manor, so we do not notice that we constantly use them. Mark Weiser, the father of *Ubiquitous Computing*, has described this as:

[2]Drawbacks include the requirement of line-of-sight and that small balls have to be attached to all tracked objects.

Inspired by the social scientists, philosophers, and anthropologists at PARC, we have been trying to take a radical look at what computing and networking ought to be like. We believe that people live through their practices and tacit knowledge so that the most powerful things are those that are effectively invisible in use. This is a challenge that affects all of computer science. Our preliminary approach: Activate the world. Provide hundreds of wireless computing devices per person per office, of all scales (from 1" displays to wall sized). This has required new work in operating systems, user interfaces, networks, wireless, displays, and many other areas. We call our work "ubiquitous computing". This is different from PDA's, dynabooks, or information at your fingertips. It is invisible, everywhere computing that does not live on a personal device of any sort, but is in the woodwork everywhere.[3]

This idea of computing *"everywhere at anytime"* when transfered to tracking leads to the ideas behind *Ubiquitous Tracking.*

That we are heading towards a world where *Ubiquitous Computing* is everywhere can be seen on the type of computers and computing devices sold. Where the personal computers are falling in sales, smaller and/or more integrated computing devices such as PDAs, mobile phones, intelligent whiteboards and other wireless enabled devices are becoming more used. This leads to more *Mobile Computing* and the *Ubiquitous Computing era* following in the aftermath.

1.3 Ubiquitous Tracking

As mentioned earlier, when building large scale *AR* applications the complexity of the tracking setup can quickly become overwhelming. This is caused by the limitations and the diversity in attributes of the different tracking mechanisms. Especially the limitations of coverage area means that many sensors have to be connected and form a large network of sensor data which might even have to be extensible or otherwise changing dynamically.

Under the assumption that we have all of these tracking devices everywhere, we need a way of communicating our ideas, describing our sensor setups and the spatial relationships between objects. The *Ubiquitous Tracking* framework introduces a formal way of modeling such large networks. This is done in such a broad way that it even allows existing setups to be described. If looking at arhitecture layers, *Ubiquitous Tracking* will provide a new layer below the application layer, so appalications can be build independently of the sensors and use the *Ubiquitous Tracking* framework directly.

1.3.1 Framework

The framework is based on spatial relationship graphs which provides the systems view of the real world. For this, three different kinds of relationship graphs will be introduced.
The spatial relations are assumed to be represented via a 4×4 homogeneous matrix, like it is common in computer graphics. This is not a requirement for the framework but it makes it easier to visualize what is going on during the presentation of the framework. We also do not have to go into details of how the individual sensors work or what they measure. This knowledge is of cause assumed to be available for the system, but it is unnecessary here.

[3]Mark Weiser, 1988, http://www.ubiq.com/weiser/.

The goal is to provide query mechanism, which provides an optimal estimate of the spatial relationship between two abitrary objects at any point in time, and where *optimal* is defined by the application via an error function.

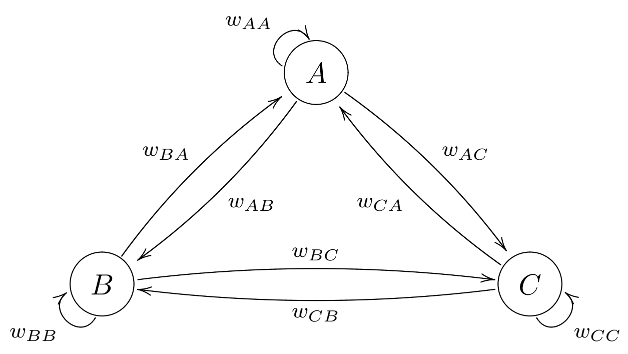
1.3.2 Spatial Relationship Graphs

The spatial relationship graphs used in the framework of *Ubiquitous Tracking* all have the nodes representing sensors and tracked objects, where spatial relationships then are represented as the directed edges in the graph.

Each edge has a function that describes the relationship by returning the 4×4 homogeneous matrix representing the spatial relationship associated with the edge. Thus if one wants to find the the relation $R^{4\times4}_{A\to C}$ between two nodes A and C separated by node B, one just has to multiply the matrices returned from the associated function $R^{4\times4}_{A\to C} = R^{4\times4}_{A\to B} \times R^{4\times4}_{B\to C}$.

Three different graphs are used in the description of a setup.

Real: A spatial relationship graph representing how an omniscient viewer would see all the objects of interest, i.e. this graph is fully connected and symmetric with the precise relations at all times.
An example setup with 3 objects could then look like:



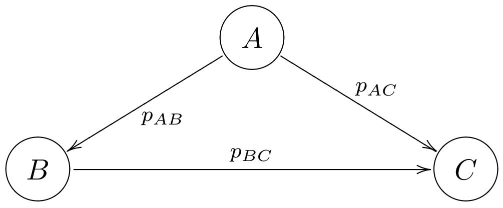
The weight functions w are then defined for all points in time D_t, and we can define a binary relation $\Omega = N \times N$ which maps pairs of objects to their weight functions:

$$W : (\Omega = N \times N) \to w, \text{ where } w : D_t \to R^{4\times4}$$

This graph is of cause impossible to obtain but it is what we would like to try to approximate, so we could query at any point in time for any relation.

Measured: A graph with all measurements as edges in the graph. This is the graph that is directly available to the system but it has major flaws compared to W - the real relationships. First and foremost are the relationships only defined at discrete points in time, i.e. every time a sensor makes a measurement, secondly is it corrupted by noise, and thirdly are only some relationships known at all.
We now associate an *attribute set* A with each weight to allow the system to compare different paths in the graph. Different attributes will be discussed later.

Same setup as before could as a measured graph look like:



$$\begin{tikzcd} & A \arrow[dl, "p_{AB}"'] \arrow[dr, "p_{AC}"] & \\ B \arrow[rr, "p_{BC}"] & & C \end{tikzcd}$$

With the measured relationship graphs a new binary relation Φ is then defined on the objects:

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

and p only is defined at discrete points in time but also returns the attributes A for the measurement.

$$p_{XY}(t) = \begin{cases} (H_1, A_1) & \text{if } t = t_1 \\ (H_2, A_2) & \text{if } t = t_2 \\ \vdots & \\ (H_n, A_n) & \text{if } t = t_n \\ undefined & \text{otherwise} \end{cases}$$

Inferred: This is a super graph of the measured relationship graph with added relations from the information the system has inferred, i.e. has calculated from the measurements or from external information. For the inferred relationships a binary relation Ψ is defined:

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

which should try to approximate the real relations Ω. q is then a set of relations between two objects since different kinds of relations are available.

As earlier mentioned are the relations in the measurement graph only available at discrete points in time. As an example this can be avoided by inferring a relation that interpolates the time of the sensor measurements. For two objects X and Y where measurements have been made at $\{t_1 \ldots t_n\}$, an interpolation could look like

$$q^i_{XY}(t) = \begin{cases} (H_1, A_1) \text{ if } & t_1 \leq t \leq t_1 + \frac{t_2 - t_1}{2} \\ (H_2, A_2) \text{ if } & t_1 + \frac{t_2 - t_1}{2} < t \leq t_2 \\ \vdots & \\ (H_n, A_n) \text{ if } & t_{n-1} + \frac{t_n - t_{n-1}}{2} < t \leq t_n \\ undefined \text{ otherwise} & \end{cases}$$

which would be added as an edge between the two nodes X and Y. This could also be an extrapolation q^e_{XY} which would be defined at all times, even times smaller than t_1 and larger than t_n. If we continue the example from before, the inferred graph would then look like (with the measurements marked as q^m_{XY}):

$$\begin{tikzcd}
& A \arrow[dl, bend right, "q^i_{AB}"'] \arrow[dl, "q^m_{AB}"] \arrow[dr, bend left, "q^m_{AC}"] \arrow[dr, "q^i_{AC}"'] & \\
B \arrow[rr, bend left, "q^m_{BC}"] \arrow[rr, bend right, "q^i_{BC}"'] & & C
\end{tikzcd}$$

Many different attributes can be used with the relations, for example:

- The *latency* (s) of measurements, i.e. the time from the actual measurement to it is available.
- The *update frequency* (Hz) of how often new measurements are available.
- The *confidence value* ([0;1]) which is the probability of it being the correct feature that is detected.
- The *pose accuracy* can be a very important for some applications.
- The *time to live* (s) is the time a relationship is likely to be valid.
- The *cost of sensors* can be very useful in a application simulation before anything is actually setup. It can be used to judge which sensors, with the lowest costs, are necessary in different parts of the applications coverage area.

Optimal paths

An application can query for a path in the inferred spatial relationship graph. To get a path that matches certain criteria, an error function is introduced:

$$e : A \to \mathbb{R}$$

which takes the attributes A from the associated function q_{XY} and returns a *weight* for this relation. For each different error function this effectively adds a weight to each edge in the graph, and in an implementation the different shortest path algorithms, such as Dijkstra's, can be used to find the optimal paths.

In some cases the definition has to be extended since not all attributes can be evaluated individually and the entire path has to be considered instead. In these cases one can define the error function as:

$$e : A^* \to \mathbb{R}$$

Now the normal shortest path algorithms can not be used though and the evaluation of paths has to follow the pattern:

Algorithm Given two nodes X and Y, an error function e and query time t. The way to find the optimal path is as follows:

1. Find all paths from X to Y that are defined at t
2. Evaluate e on collected attributes A^* for every path
3. Calculate the spatial relationship between X and Y by multiplying all matrices along the path r

If the inferred relationship is a new relation then it is added to the spatial relationships Q, so the graph is populated with all the used spatial relationships and they do not have to be calculated twice.

1.3.3 Optimizations

So far we have described the straight forward formalism used to describe tracking setups with the essential being the set attribution Q, the inferred relations q^f_{XY} and the error function e:

$$\begin{gathered}Q : (\Psi \subseteq N \times N) \rightarrow q, \text{ where } q : D_t \rightarrow R^{4\times4} \times A \\ d^f_{XY} = f((H_1, A_1), (H_2, A_2), \ldots, t) \\ e : A^* \rightarrow \mathbb{R}\end{gathered}$$

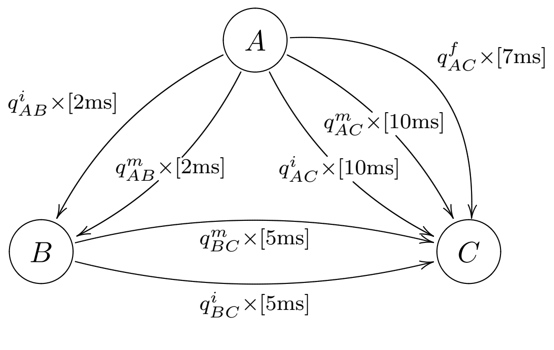
These would be straight forward to implement and integrate into existing systems, but to improve performance one should consider to implement the *data flows graphs* and *super nodes* mentioned next.

Data flow Graphs

Each time an optimal path is found it should be saved, so the path does not have to be recalculated and so it straight away can take part of other more complex paths.

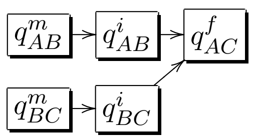
With data flow graphs this is done by creating a new graph where the edges from the new path in the spatial relationship graph are nodes. For a new inferred relation the new graph would then have the shape of a tree where data flows from the leaves to the root, the root being the inferred relation.

We can use the example from earlier by adding a latency to the attributes and then query for the path from A to C with the lowest latency as shown in fig. 1.3 where q^f_{AC} is added to the spatial relationship graph as the path from A to C with the lowest latency. With this



A
B
C
$q^f_{AC} \times [7\text{ms}]$
$q^i_{AB} \times [2\text{ms}]$
$q^m_{AC} \times [10\text{ms}]$
$q^m_{AB} \times [2\text{ms}]$
$q^i_{AC} \times [10\text{ms}]$
$q^m_{BC} \times [5\text{ms}]$
$q^i_{BC} \times [5\text{ms}]$

(a) Spatial relationship graph



(b) Data flow graph

Figure 1.3: Example of a data flow graph

data flow graph saved, every time the application queries for the path, it is found right away and the spatial relationship AC is just recalculated with the data at given time t.

The advantages of using a data flow graph is dependent on how often structural changes in the relationship graph occur since such changes would mean that the data flow graph had to be recomputed. Furthermore must it be recomputed if an attribute of one of the components changes, since the error function, used to calculate the path, is dependent on the attributes.

Super Nodes

Another way to optimize the graph searches for optimal path is to group nodes that have some special relation to each other. This can be done if some nodes are statically linked with each other, or if two clusters of nodes are so far apart (compared to internal distance), that if the relation between one node A in cluster S_1 with another node B in cluster S_2 is queried then the relation between A and S_2 is enough.

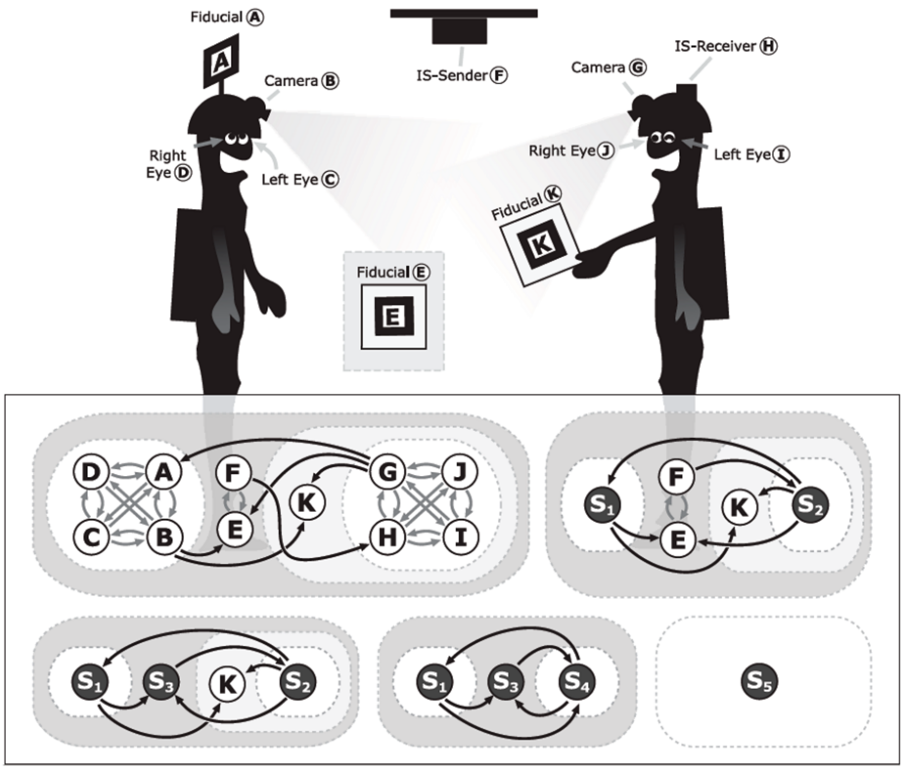


Figure 1.4: Super nodes courtesy of http://ar.in.tum.de/

Fig. 1.4 shows an example of how super nodes are created. In the first sub-picture is the entire spatial relationship graph shown and in the second it is then reduced where the static linked nodes are combined to S_1 and S_2. In the sub-picture is the IS-Sender F and the fiducial E also linked together to S_3, since the fiducial is not moving compared to the IS-Sender. If moving even further away one can also combine the *soft links* which combines the loosely connected nodes like the right person holding the fiducial K and combined to S_4. Finally if moved far enough away like in another building, then would the internal relations most likely not matter, and they could all be grouped into one node S_5.

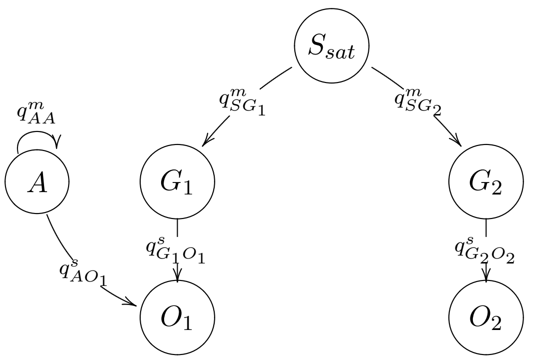
1.3.4 Example

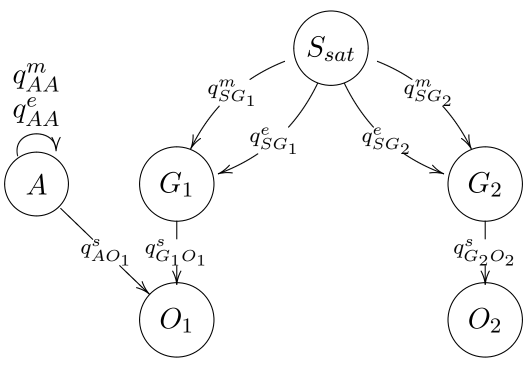
This section is based on an example of use of the framework, to show how an tracking setup could be described, and to shown a bit of the flexibility of the framework.

Imagine wanting to relate an accelerometer A in a car O_1 with an GPS measurement of the cars position to have an more frequent update of the position (50Hz whereas GPS can provide 1Hz) compared to another stationary object O_2.

What we then query about is: O_1 *spatial relation to* O_2 *at time* $T+\Delta t$ *via a Kalman filter.* Note that this is a small time step in the future.

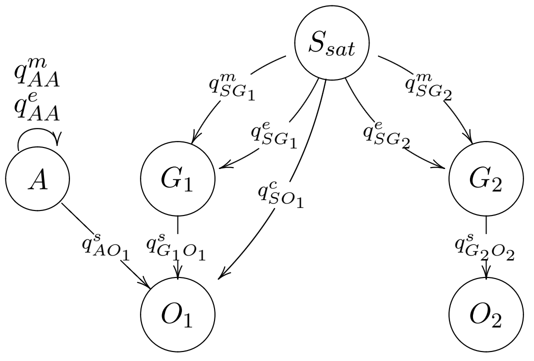
O_1 is the moving car, O_2 is a stationary object. Both have a GPS object G_1 and G_2 respectively and the car also contains an accelerometer A. Note that the accelerometer only measures relatively to its own former position.



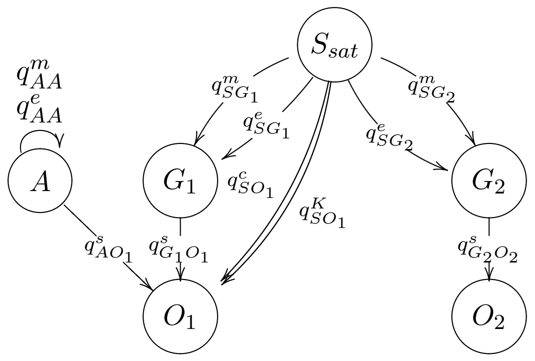


First of all, an extrapolation is added at each measurement to get results at any requested time (the measurements q^m are only defined at the discrete points in time where a measurement is done.)
This gives us the new edges q^e running next to all measurements.

Then is the GPS combined with the accelerometer to get an higher update rate of the cars position. By every update from the GPS, would the position of the accelerometer be reset and drift in the accelerometer is avoided.
The new edge q^c is then still a spatial relationship between the origin of the GPS and the moving car.



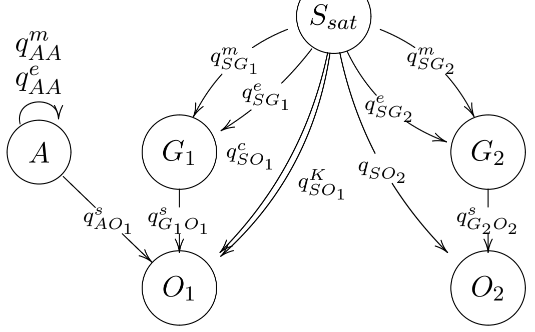
$$\begin{tikzcd}
\boxed{q^m_{AA}} \arrow[r] & \boxed{q^e_{AA}} \arrow[r] & \boxed{q^s_{AO_1}} \arrow[r] & \boxed{q^c_{SO_1}} \\
\boxed{q^m_{SG_1}} \arrow[r] & \boxed{q^e_{SG1}} \arrow[r] & \boxed{q^s_{G_1O_1}} \arrow[ur] &
\end{tikzcd}$$

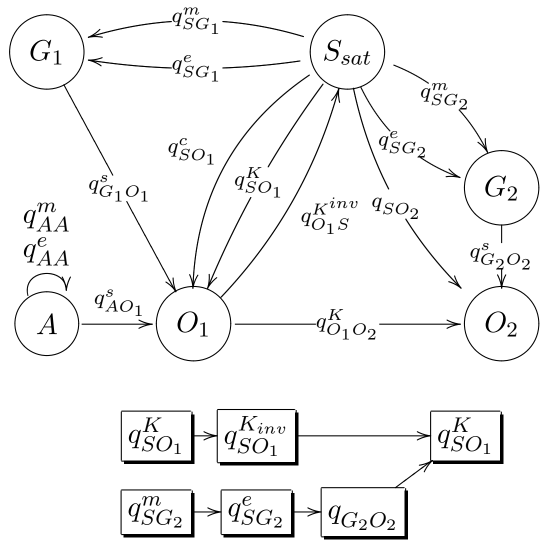


$$\begin{tikzcd}
\boxed{q^m_{AA}} \arrow[r] & \boxed{q^e_{AA}} \arrow[r] & \boxed{q^s_{AO_1}} \arrow[r] & \boxed{q^c_{SO_1}} \arrow[r] & \boxed{q^K_{SO_1}} \\
\boxed{q^m_{SG_1}} \arrow[r] & \boxed{q^e_{SG_1}} \arrow[r] & \boxed{q^s_{G_1O_1}} \arrow[ur] & &
\end{tikzcd}$$

The edge q^c can then be used in a Kalman filter to make small predictions about where the car is going to be.
This new edge is also a spatial relationship between the origin the GPS use and the car.

A relationship edge q_{SO_2} is also added to give a direct relationship between the origin of the GPS and the stationary object O_2.





The Kalman filter edge is then inverted $q^{K_{inv}}_{SO_1}$ and finally a direct relationship between O_1 and O_2 can be made. Like all other relations, once the new path is build, it does not have to be recomputed as long as no major changes in the graph occurs, and thus the application has freed resources to other computations, like faster sensor updates.

1.4 Conclusion

This paper first introduced some related concepts of *Ubiquitous Tracking* which helped explain why we need it to describe our tracking setups. Following that, a more formal introduction was made but with several examples to illustrate the frameworks use.
The framework itself is based on spatial relationship graphs, with objects as nodes and attributes plus a representation of the spatial relationship associated with each edge. It is meant to create a seperation between *AR* applications and the sensors, and defines a mathematical framework to describe complex tracker setups.

Bibliography

[1] A. MacWilliams, C. Sandor, M. Wagner, M. Bauer,G. Klinker, B. Bruegge, *Herding Sheep: Live System Development for Distributed Augmented Reality*, (2004).

[2] R. Azuma, *A Survey of Augmented Reality, pp. 355-385*, Presence: Teleoperators and Virtual Environments, vol. 6, no. 4, (1997).

[3] G. Klinker, T. Reicher, B. Bruegge, *Distributed User Tracking Concepts for Augmented Reality Applications.*

[4] J. Rolland, L. Davis, Y. Baillot, *Fundamentals of Wearable Computers and Augmented Reality*, Lawrence Erlbaum Associates, 2001, ch. A survey of tracking technology for virtual environments.

[5] G. K. M. Wagner, *An Architecture for Distributed Spatial Configuration of Context Aware Applications.*

[6] U. Hansmann, L. Merk, M. S. Nicklous, T. Stober, *Pervasive Computing - The Mobile World*, Springer Professional Computing, Springer-Verlag Berlin Heidelberg New York, IBM, Germany, second ed., 2003.

[7] M. Wagner, A. MacWilliams, M. Bauer, G. Klinker, J. Newman, T. Pintaric, and D. Schmalstieg, *Fundamentals of Ubiquitous Tracking*, in Second International Conference on Pervasive Computing, Hot Spots section, Vienna, Austria, 2004.

[8] M. Weiser, *Ubiquitous Computing.* http://www.ubiq.com/weiser/.

2 Existing Software and Systems

Georgi Nachev, Technische Universität München

In this work some existing systems for Ubiquitous Tracking are presented: OpenTracker, which implements a static dataflow model for streams of sensors readings; DWARF - a framework for component-based peer-to-peer systems; VRPN - a static network-transparent abstraction between applications and pre-defined trackers and Trackd - a commercial system from VRCO Inc., which abstracts the tracking from the other applications and offers a common API.

2.1 Introduction

Augmented Reality applications are highly dependent on accurate and precise tracking data. Since current tracking technologies do not always provide such information everywhere in real-time, application developers have to combine several trackers to minimize negative properties of one tracker by another. The result are sensor networks. They can be used to inform applications about the current position and orientation of objects.

Currently the most AR applications use their own customized solution of this problem. Typically, these solutions are hardly reusable in other systems. Because there are no standard interfaces between these technologies, the development of large-scale sensor networks is inhibited. In this paper, I will look at some existing architectures and systems, which configure ubiquitous tracking environments that are composed of several sensor networks.

Some current systems have a modular architecture that allows to apply different types of tracking devices easily and simultaneously. This is tipically the approach in commercial VR systems, which offer support for many popular tracking systems and input devices, but limit in this way their extensibility and configuration options.

Research systems are usually limited to their particular research domain and are not for commercial use, but they could offer features that are not provided by other systems.

Needed are tools that allow a high degree of customization, but also easy to use and to extend. They must allow mixing and matching of different features, as well as simple creating and maintainance of possibly complex tracker configurations.

2.2 Important goals and requirements

There are many important goals and requirements, which have to be satisfied by the middleware-systems:

- Device abstraction - to provide a fixed interface to the application for different devices and tracking systems, and to offer simple services for relaying the data over the network between several hosts.
- Support for distributed simulation (simultaneous users, or to better exploit available hardware). Decoupled simulation is used in almost any VR software. It can be implemented either by multithreading or symmetric multiprocessing on the host, or by set of hosts working as a cluster.
- Describe and configure complex dependencies and interactions between devices and subsystems.
- Model AR applications as distributed systems which are formed spontaneously by mobile and stationary components.
- The system has the possibility of runtime reconfiguration and recalibration.
- Ease of use - no additional code is required to adapt the system to the needs and requirements of the user.
- No single point of failure - in case of failure of the infrastructure parts of the overall sensor network can still be used.

Nowadays there is no available system, that provides all these features together, but the developers have recognized the importance of these requirements and partly implemented them into the applications described below.

2.3 Existing Systems

In this section the systems are presented in detail: OpenTracker - research system, part of the Studierstube project [5] DWARF - AR-framework based on distributed services with dynamic setup [1]; VRPN[9] and trackdVRPN[7] - comercially used systems for Virtual Reality.

2.3.1 OpenTracker

OpenTracker is architecture that provides a framework for the different tasks involved in tracking input devices and processing multi-modal input data in virtual environments and augmented reality applications [4]. It eases the development and maintenance of hardware setups in a more flexible manner, what is achieved by using an object-oriented design based on XML, taking full advantage of this technology by allowing to use standard XML tools for development, configuration and documentation. The OpenTracker engine is based on a data flow concept for multi-modal events. Transparent network access allows easy development of decoupled simulation models. Finally, the application developer's interface features both a time-based and an event-based model, that can be used simultaneously, to serve a large range of applications.

OpenTracker has the following important characteristics:

- An object-oriented approach to an extensive set of sensor access, filtering, fusion, and state transformation operations.

- Behavior specification by constructing graphs of tracking objects from user defined tracker configuration files.
- Distributed and decoupled simulation by network transfer of event at any point in the graph structure.

OpenTracker encourages exploratory construction of complex tracking setups, what can be useful for end users, who want to fully exploit their hardware without any custom programming, as well as for developers, who can easily build test environments. The modular approach gives instant access to wide range of tracking related functionality for any application.

Typically in the most VR and AR applications tracking data passes through a series of steps - it is generated by tracking hardware, read by device drivers, transformed to fit the requirements of the application and sent over the network to other hosts. Different setups and applications may require different subsets and combinations of the steps described, but the individual steps are common among a wide range of applications.

The main concept behind OpenTracker is to break up the whole data manipulation into these individual steps and build a data flow network of the transformations. Each transformation is represented by a node in a data flow graph. They are connected by directed edges to describe the direction of flow. The origin of the edge is the child node and the endpoint is the parent node. A port is a distinguished connection point for an edge. Each node has one or more input ports and a single output port. The output port of one node is connected to any of the input ports of another node - see Figure 2.1.

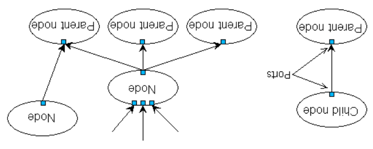


Figure 2.1: Data flow concept in OpenTracker

After receiving a new data event via one of its inputs the node computes a new update for itself and sends the new data event out via its output port. Because computations have typically more than one parameter, multiple input ports are needed. An input port can also be connected to several output ports. This enables the connection of several children nodes to the same input port of a node, but the parent node only distinguishes the input ports, not the actual children. On the other side, an output port can also be connected to many input ports of different nodes. However the child node cannot selectively send events to one parent, but only equally to all.

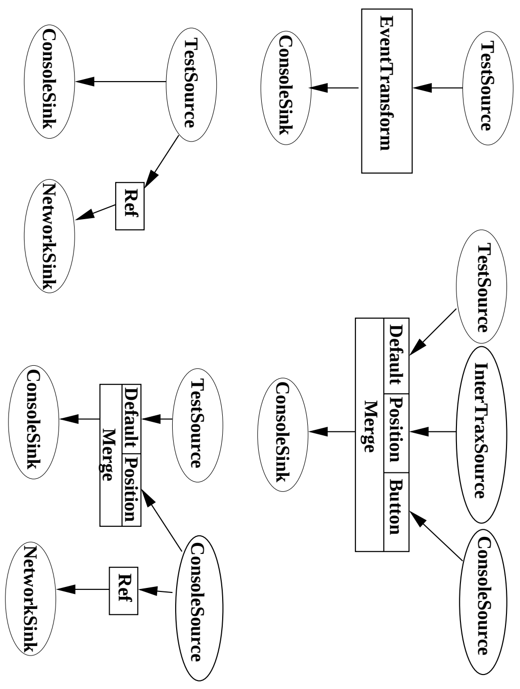


Figure 2.2: Examples of data flow graphs, built with OpenTracker

Another important concept is, that different edge types are distinguished. They are typed by typing the ports of the nodes they connect. Only two ports of the same type can be connected and this type is equal to the type of the edge. There are three edge types: event, event queue and time dependent. Event type is implemented by event passing; by the event queue is possible to query the number of stored events and retrieve them by index, and the time dependent interface can be queried by specifying a point in time, for which the appropriate data is returned.

OpenTracker also differentiates between three types of nodes - source nodes, which receive their data from external sources; filter nodes, that are intermediate nodes and modify the values received from other nodes; and sink nodes, which propagate their data to external ouputs (see Figure 2.2). Most source nodes encapsulate a device driver, that directly accesses a particular tracking device. Others emulate a tracker via the keyboard, access network data or simply respond with constant values. The filter nodes receive data from one or more child nodes and compute their own state based on the collected data. The computation may be geometric transformation of the children's values, prediction in order to compensate measuring lag, merge of different parts of data values, conversion of one data type into another, etc. The sink nodes are similar to source nodes but distribute data rather than receive it. They include output to network multicast groups, debugging output to a user interface or threadsafe shared memory.

Because the intent of OpenTracker is to provide an auxiliary library that is to be integrated into VR and AR applications, the software is kept very lightweight and customizable. The library is implemented in C++ and is build around a small set of core classes, a parser that builds the runtime structure from a configuration file and the main loop driving the event model. Any other functionality is implemented by a set of module classes that can be easily extended or modified. There is no fixed interface to the integrating application in order to maximize flexibility. Application programmers have to use one of the supplied nodes, or supply their own module implementing sink nodes as interfaces to their application.

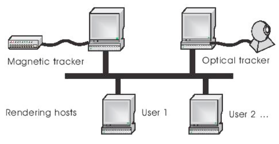
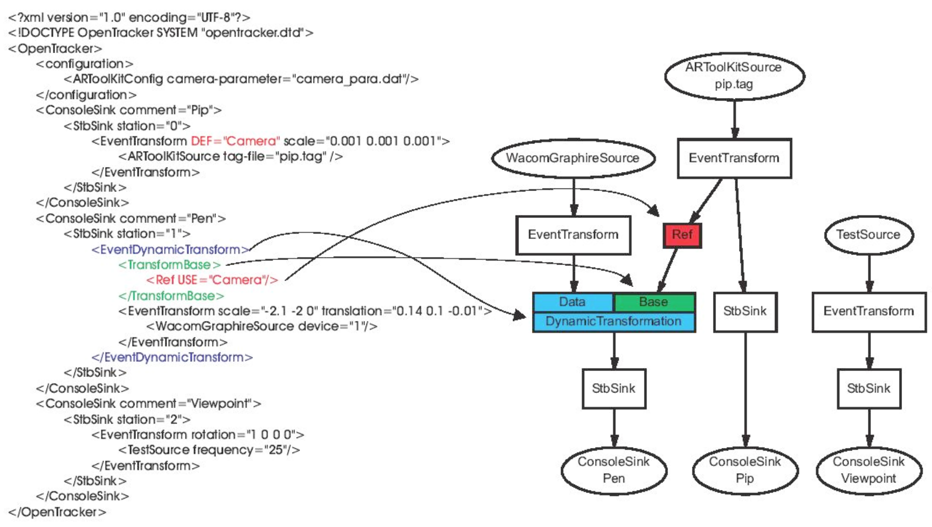


Figure 2.3: Distributed tracking data over the network

One of the important goals of the OpenTracker software is to share tracker data over the network. There are several reasons why it is desirable to do this. Network support makes it easy to span multiple operating systems, in particular if a specific tracking device or service is only available at one particular host. To achieve some degree of load balancing multi-processing based on inexpensive PCs becomes possible with little configuration effort. And last but not least, using tracker data at multiple hosts, distributed virtual environments(Figure 2.3) are achieved.



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE OpenTracker SYSTEM "opentracker.dtd">
<OpenTracker>
    <configuration>
        <ARToolKitConfig camera-parameter="camera_para.dat"/>
    </configuration>
    <ConsoleSink comment="Pip">
        <StbSink station="0">
            <EventTransform DEF="Camera" scale="0.001 0.001 0.001">
                <ARToolKitSource tag-file="pip.tag" />
            </EventTransform>
        </StbSink>
    </ConsoleSink>
    <ConsoleSink comment="Pen">
        <StbSink station="1">
            <EventDynamicTransform>
                <TransformBase>
                    <Ref USE="Camera"/>
                </TransformBase>
                <EventTransform scale="-2.1 -2 0" translation="0.14 0.1 -0.01">
                    <WacomGraphireSource device="1"/>
                </EventTransform>
            </EventDynamicTransform>
        </StbSink>
    </ConsoleSink>
    <ConsoleSink comment="Viewpoint">
        <StbSink station="2">
            <EventTransform rotation="1 0 0 0">
                <TestSource frequency="25"/>
            </EventTransform>
        </StbSink>
    </ConsoleSink>
</OpenTracker>
```

Figure 2.4: Example configuration file and the resulting graph.

OpenTracker allows multiple senders and receivers of tracker data to communicate asynchronously through the network using IP multicasting. Because each of the senders and receivers can operate independently, this approach implements effectively decoupled simulation. It is even possible for a single host to operate as a sender and receiver at the same time.

The data flow graph in OpenTracker is defined from user by tracker configuration files. These configuration files are written in markup language defined in XML, that consists of elements, annotated by name-value pairs called attributes and structural model of the possible ways these elements may be nested - Figure 2.4.

OpenTracker maps elements to nodes and attributes to members of these nodes. Then XML parser builds a tree of elements representing the given configuration file, and so for each element a new node can be created. The corresponding members are set according to the parsed string values of the attributes. The parent - child relationship of the data flow is directly mapped onto the parent - child relationship of the XML elements.

An example made by Studierstube Augmented Reality Project(Figure 2.5), where OpenTracker is integrated, represents an experimental pen-and-pad interface, where a vision tracking approach(ARToolkit) for the pad and a magnetic tracker(Ascension Flock of Birds) for

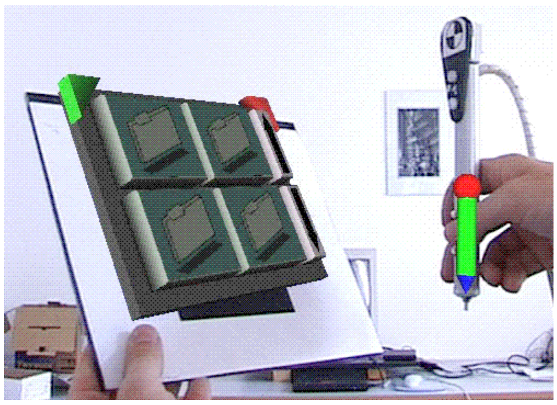


Figure 2.5: Experimental pen-and-pad, where vision and magnetic tracking data are combined

the pen are combined. Two separate servers for video and magnetic tracking send their measurement over the network to a rendering host, where the combined data is picked up by an OpenTracker component.

2.3.2 DWARF

DWARF stands for Distributed Wearable Augmented Reality Framework and represents a framework for component-based peer-to-peer systems. Here AR applications are modelled as distributed systems which are formed spontaneously by mobile and stationary components [2]. Such a system can reconfigure itself at runtime by exchanging components and changing component configurations. It represents a network of collaborating distributed services, which expose their requirements, called Needs, and their offers, so called Abilities - Figure 2.6. In fact there is no central management-component, but a service manager on each network node. They control their local services, maintain descriptions of them and set up connections over the network to other services. The service description is written in XML and contains its Abilities, Needs and communication protocols, called connectors.

Both the Abbilities and the Needs have a set of attributes describing the quality of service parameters of the service, that are offered respectively expected. Thus, it is possible for the service manager to select abilities, that can provide a sufficient quality of service to satisfy a given need, or to ensure at runtime, that the desired quality of service is still provided.

This concept leads to several advantages:

- Platform independence and heterogeneity: The hosts, where DWARF services were deployed may use different operating systems. As well there is a flexibility in the choice of the programming language for building a particular service. Usually this choice is made according to the third-party library needed to build the service.

- Modularity and Distribution: All hardware details are abstracted by services, which become highly reusable and flexible.

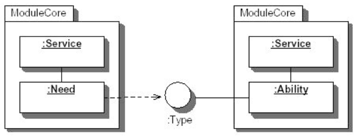


Figure 2.6: Services are connected by Needs and Abilities

- Easily integration of third-party hard- and software: There are several components available from other research groups, which can be reused. They can have special requirements like a fixed runtime environment, but using appropriate wrappers around these components, ease of use can be achieved.
- Remote Monitoring: For monitoring and debugging purposes a so called DWARF Interactive Visualization Environment is developed. This tool represents a graphical view of the network of interconnected services that dynamically changes if the system configuration changes, see Figure 2.7. Developers can trace arbitary event streams in the system.

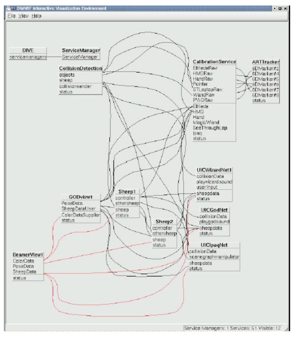


Figure 2.7: Interactive Visualization Environment

The DWARF Services are grouped in subsystems, which are responsible for specific tasks. Some important ones are:

- The Presentation Subsystem provides an Viewer - 3D-OpenGL-Renderer where models can be loaded and registered with Needs for position data. It supports various file formats such as OpenInventor(.iv), VRML1 and VRML2(.wrl), and three different render modes for mono- and stereovision. The Viewer is based on the OpenInventor implementation Coin by Systems in Motion.
- In the Tracking Subsistem any tracking hardware is encapsulated in appropriate services. For example there is a service, that provides tracking data from the ART Dtrack system. Furthermore this subsystem provides a calibration service for tuning purposes.
- The Application Subsystem is the place where the application developer can put its own services.
- In the Input Subsystem all services concerned with user interaction and input devices have to be implemented.

An example project made by Augmented Reality Research Group using DWARF is SHEEP(Figure 2.8) - The Shared Environment Entertainment Pasture. In this demo a multiplayer sheep-herding game is used to explore the possibilities of multimodal, multiuser interaction with wearable computing in an intelligent environment.

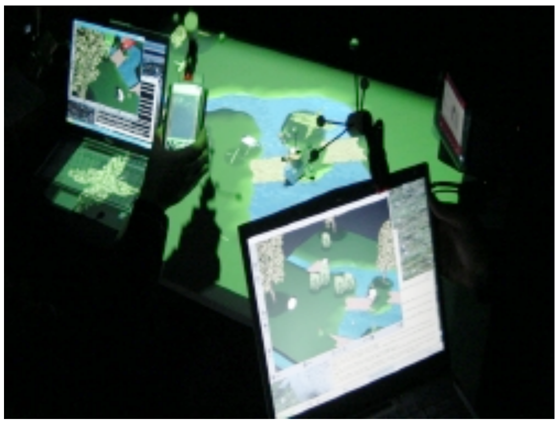


Figure 2.8: SHEEP-demo

The game is centered around a table with a beamerprojected pastoral landscape. Players can use different intuitive interaction technologies (beamer, screen, HMD, touchscreen, speech, gestures) offered by the mobile and stationary computers. Building on the DWARF framework, the system uses peer-to-peer, dynamically cooperating services to integrate different mobile devices (including spectators laptops) into the game.

Ubiquitous Tracking in DWARF

In order to dynamically adapt tracking middleware systems to changes in sensor infrastucture, an approach is proposed, in which diverse and widespread heterogeneous tracking sensors are automatically discovered [3]. Ubiquitous Tracking or Ubitrack obtains an optimal estimation of arbitrary geometric relationships and their accuracy at any time, in response to an application's query of environmental state for a given definition of optimality. The spatial information is modelled by a Spatial-Relationship (SR) graph. The nodes of this graph are all identifiable objects in the (real or virtual) world, the edges represent the spatial information of two node objects.

When the relationship of one object to another is searched, all edges from one to the other are traversed and all information on these paths is evaluated. This framework introduces an abstraction layer between the Sensor and the Application Layer - Ubitrack Layer. It aggregates tracking data, inferres knowledge of spatial relationships, and builds runtime data flow graphs from the abstract spatial relationship graph. Then a query API incorporating the desired spatial relationship and the specification of the evaluation function is provided to applications.

In order that Ubitrack can be integrated into DWARF, an Ubitrack Middleware Agent(UMA) is introduced, which is started on each host. It is connected to the local ServiceManager and receives in this manner information about all local objects and the measurements between them. All possible paths between arbitrary local objects can be computed and stored together with the corresponding set of attributes. After this step, all local objects are grouped in a supernode and the paths are cached. If a new object appears in the tracking range of a local tracker, it is stored as an anonymous object until it can be identified. Attempting to identify this object the local UMA tries to connect to other UMAs which are in close range and requests information about their local objects in order to find similarities. If the anonymous object can be identified, a communication channel to its UMA is set up. The new object is now stored in both UMAs. If the object leaves the local tracking range, it is deleted from the local UMA as well as the communication channel.

2.3.3 VRPN

The Virtual Reality Peripheral Network(VRPN) provides a device independent and network-transparent interface between applications and several physical devices and systems (e.g. Tracking) used in a VR-system [6]. It offers moreover:

- Multiple simultaneous connections to devices
- Automatic reconnection to failed remote servers
- Time stamps for all messages
- Clock synchronization between client and server on different machines
- Storage and replay of interactive sessions

Usually in the laboratories multiple graphics display stations require access to several VR-peripherals. To run interface cables from each device to each host is often inconvenient. Different devices, which perform the same function, may have different interfaces. Some of them require special connections or work only on certain operating system. VRPN is developed to address all these issues, using local device servers, which communicate with graphics engines through the network.

Talking about VRPN, we have to define first the canonical device types, which specify a consistent interface and semantics. Particular device is of one or more canonical device types and implements its or their interface(s). Some device types are: Tracker, Button, Analog, Dial, Force Device. Then we have to map the capabilities of this device onto its interface. VRPN uses the following features to do this:

- Factoring devices based on their functions: When a device implements more than one function, the VRPN driver will export interfaces for multiple device types. Thus, no client-side code change is required to move an application to different sets of I/O devices.
- Mapping devices to connections: Although there are exported multiple interfaces for a single device, they all are internally mapped to the same network connection to maintain communication efficiency.
- Enabling devices to export multiple interfaces: The same physical device may act as different device types at different times. Both interfaces are exported under different names. A special case is the layered device - a higher-lever behavior is built on top of an existing device. Application code can afterwards attach to either or both devices - Figure 2.9.

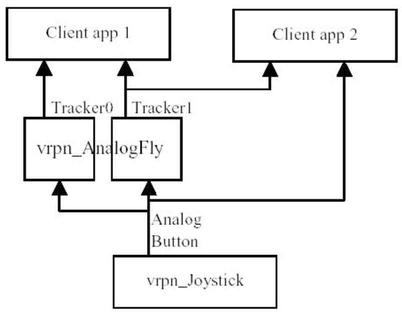


Figure 2.9: Example layered device.

Each device driver in VRPN may have a server- and client-side, which communicates over a connection object. No dependence on particular port on the server and on the client is required for this object. Although it is complicated, it runs only at connection startup: Each server opens a well-known UDP-Port for connection requests from clients. A client opens any available TCP port and sends a UDP request to the server asking it to connect to that TCP port. The client waits for response from server and returns control to the application if it is not received. Once the reliable TCP chanel is established, it is used to establish a separate unreliable UDP chanel between the hosts, also for version checking and clock synchronization. In case of dropped connection, a message is sent to any interested objects indicating the drop. A client will attempt to re-establish the link using the same algorithm. This enables the application to be robust and very useful in cases of long start up times. Even though VRPN is designed to separate client and server over network connections, it is also possible to run in separate process on the same machine, or within the same process.

The possibility of VRPN to store and replay a session is realized by a log file mechanism, by which all messages passed over a connection are saved. This can be used e.g. to record user motion during human-factors studies. The logging is done at either the client side or the server side. Then an application can connects to a stored file and reads from its devices, or replays the original session at its normal rate without any extra code.

One of the most important question using VRPN is, if the performance of the system is comparable with this of a locally-connected device using device-specific driver. Against the time costs VRPN measures well. For some configurations the time to read a message using a remote VRPN server can be significantly less than that of a locally-connected device. The timing information and latency-reducing optimizations are responsible for this. Network latency tests between SGI and Linux box within a switched Ethernet showed average one-way times of 3,3ms for application-level VRPN messages. This includes all overheads from the operating systems network layers. Slightly lower times was measured from Linux client to a Windows 98 server, and an average of 1,7ms between SGI client and Windows 98 server.

2.3.4 Trackd

Trackd is a comercial device software from VRCO Inc. for VR applications in the immersive display industry. It is a small deamon application that takes information from a variety of tracking and input devices and makes that information available for other applications to use. The software is used by all VRCO products and many applications form leading software companies and is available with most commercially installed systems. Its networking capabilities and support for a variety of operating systems allow different graphics machines to share information from tracking and input devices [8].

Applications using the Trackd do not need to know what type of hardware device is being used, because the applications receive data from trackers and input devices through a generic Application Programmers Interface, called the trackdAPI. This generic interface allows applications to support a range of hardware without having to modify source code, or write specific hardware drivers. It also allows immersive display owners to change, replace or upgrade components without having to replace their software, or losing functionality.

The Trackd package is a combination of two applications - Trackd Server and Trackd Daemon, and various modules, which are compiled as shared object libraries to support various devices. Each device has its own module.

Trackd Server opens up a connection and "talks to" tracker or controller devices. It sends the devices' data through the network or serial port to a Trackd Daemon, or to another Trackd Server. Trackd Daemon collects data from any connected devices via the serial port, or reads data coming across a connector from the Trackd Server. It is also responsible for storing device data in shared memory. Once in shared memory, the data can be accessed by applications using the trackdAPI. To handle device data, the Trackd Daemon and Trackd Server requires configuration files. Each tracker or input controller must be configured in one of these files.

Both, the Trackd Daemon and Trackd Server can be configured to connect to any of the supported trackers or controllers. But only the Trackd Daemon is able to write the data into shared memory for use by the user-application. The Trackd Server is only necessary when an input device (tracker or controller) can not be directly connected to the system running the user-application software.

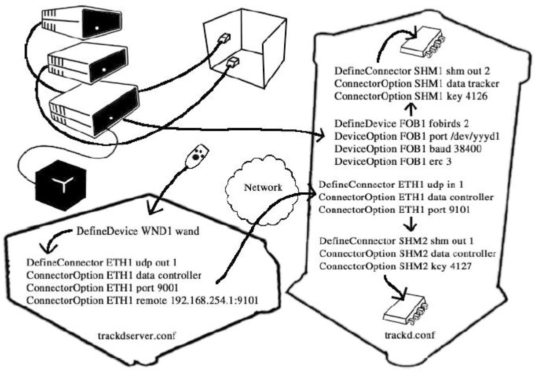
Trackd uses two terms to describe objects that do the actual work within Trackd Deamon and Trackd Server - device and connector. A device is an object that communicates with a physical input device. Otherwise a connector represents an object that is responsible for transferring, shipping and storing data. There are three types of connectors in Trackd - shared memory, serial cable and network(UDP). Connectors and devices are defined in the configuration files. Each connector and device has a unique set of variables that can be configured for it. These variables describe how and where a connector is to transfer data, and how and where a device is to acquire data.

Furthermore, connectors have two "flavours", one for tracker data and one for controller data. A given connector can receive or transmit data for either a tracker or a controller, but not both. So if a Trackd Daemon were configured for two devices, one a tracker and the other a controller, two shared memory output connectors would also have to be defined. One connector to output tracker data to shared memory, and one to output controller data to shared memory.

Within a Trackd Daemon and a Trackd Server data is transferred from device objects to outgoing connector objects automatically. There is nothing required in the configuration file to inform the Trackd Daemon or Trackd Server to transfer the data that is read in by a device object to an output connector object. If an output connector is defined, any device data of the same type (controller or tracker) will automatically be passed to the connector. Likewise input connectors automatically pass their data to output connectors of the same data type.

The Trackd Server can ship data to the Trackd Daemon using one of two connector types: over a network using UDP or over a serial cable using RS232 ports. The Trackd Daemon can receive data from a Trackd Server in the same way, but it has one more connector: the shared memory connector. It is the mechanism used by the Trackd Daemon to write data to shared memory for use of the VR application.

The device object describes a physical piece of hardware, and the Trackd software knows the protocol for talking with that hardware device. Each devices that the Trackd supports has its own shared object library written for it. There are many device types, one for each tracker and controller that the Trackd package supports.



```
DefineDevice WND1 wand

DefineConnector ETH1 udp out 1
ConnectorOption ETH1 data controller
ConnectorOption ETH1 port 9001
ConnectorOption ETH1 remote 192.168.254.1:9101

trackdserver.conf
```

```
DefineConnector SHM1 shm out 2
ConnectorOption SHM1 data tracker
ConnectorOption SHM1 key 4126

DefineDevice FOB1 fobirds 2
DeviceOption FOB1 port /dev/yyyd1
DeviceOption FOB1 baud 38400
DeviceOption FOB1 erc 3

DefineConnector ETH1 udp in 1
ConnectorOption ETH1 data controller
ConnectorOption ETH1 port 9101

DefineConnector SHM2 shm out 1
ConnectorOption SHM2 data controller
ConnectorOption SHM2 key 4127

trackd.conf
```

Figure 2.10: Controller attached to remote PC and tracker attached to VR display system.

The most interesting Trackd scenario is when one of the devices is plugged into the local machine, and another is plugged into an auxiliary machine with Trackd Deamon running on the local machine, see Figure 2.10.

Connecting devices directly to the VR display system may eliminate a negligible amount of lag in the transfer of data across the network. But this is usually a judgment call by the user, which may depend on the limited number of available serial ports on a given system, as well as cabling requirements for connecting the device.

2.4 Conclusion

Sensor networks minimize negative properties of one tracker by another. Therefore middleware-systems are developed, which generate ubiquitous tracking environments, consisting of these networks. They have modular architecture, define standard interfaces, allow to easily and simultaneously apply different types of tracking devices. Moreover they are easy to use and to extend, yet allow high degree of customization. Some Similar systems have shown in this work. Neither of them satisfy all requirements listed in chapter 2, because all have a different main focus.

Bibliography

[1] *Augmented Reality Research Group*, 2005.

[2] B. Bruegge and G. Klinker, *DWARF - Distributed Wearable Augmented Reality Framework*, November 2004.

[3] 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 International Symposium on Mixed and Augmented Reality (ISMAR) 2004, Arlington, VA, USA, 2004.

[4] G. Reitmayr and D. Schmalstieg, *An Open Software Architecture for Virtual Reality Interaction*, in Proceedings of ACM Symposium on Virtual Reality Software and Technology (VRST 2001), Banff, Canada, 2001.

[5] *Studierstube Augmented Reality Project*, 2005.

[6] R. M. Taylor II, T. C. Hudson, A. Seeger, H. Weber, J. Juliano, and A. T. Helser, *VRPN: A Device-Independent, Network-Transparent VR Peripheral System*, in Proceedings of ACM Symposium on Virtual Reality Software and Technology (VRST 2001), Banff, Canada, 2001.

[7] *Trackd Overwiew*, 2005.

[8] *Trackd User's Guide*, 2005.

[9] *Virtual Reality Peripheral Network*, 2005.

3 Algorithms for Tracker Alignment

Basti Grembowietz, Technische Universität München

Given two sensors rigidly connected to each other, tracker alignment is used to determine the transformation between those two. Once correctly calibrated, the measurements made by the sensors can be related each other. A short scenario: A camera can be tracked by attaching a fiducal to it that is tracked by another device; knowledge of the pose of the camera can be used together with the images obtained by the camera to generate three dimensional models of the scene perceived.

This paper presents techniques for obtaining the transformation between two coordinate systems obtained from trackers, starting from the classical way from Tsai, a nonlinear two stepped algorithm, and ending at a linear one stepped algorithm from Daniilidis which makes use of dual quaternions.

3.1 Introduction

This section provides an overview about tracker alignment defining what it is, how it can be achieved and how it is related to hand-eye calibration. After that, some applications will be shown.

3.1.1 What is Tracker Alignment?

Tracker alignment is needed when more than one tracking system are combined, i.e. trackers of different tracking systems rigidly connected (see figure 3.1), or if more than one fixed receiver for the same tracking system is used.

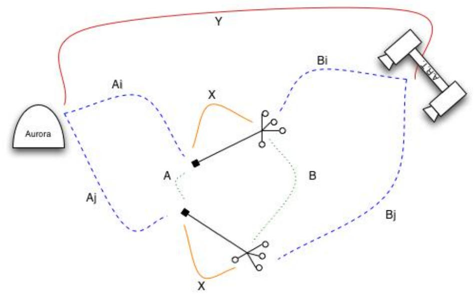


Figure 3.1: alignment of two sensors

It is equivalent to hand-eye calibration which in robotics is used to align a camera ("eye") fixed on a robot gripper ("hand") to the gripper.

Sample applications of what can be done with calibrated trackers will be shown in the next section.

3.1.2 What are Aligned Trackers used for?

As pointed out before, the main advantage of using several different trackers is the combination of their strengths and the compensation of their weaknesses. Furthermore the same methods used for tracker alignment can be used to align a camera mounted on a robot with the robot itself.

The next sections will first explain the general advantage of combining trackers, after that a few examples will be shown that combine a camera and a robot arm.

Combining Different Sensors

Two or more rigidly connected sensors that use different tracking methods can be combined using tracker alignment – in figure 3.1, there is an electro-magnetic sensor (Aurora[1]) used as well as optical tracking in the infrared spectrum (ARTrack[2]). The A.R.T tracking has an update rate of 60 Hz while Aurora only provides 45 Hz at max. On the other side, A.R.T depends on the line-of-view, Aurora does not. When both are combined a robust hybrid tracker with high tracking rate emerges that still works if the line-of-sight is not given.

When optical tracking is used with inertial tracking the much higher update rate of the gyroscopes can be used to obtain pose information if the tracker moves really fast. The greatest weakness of the gyroscopes, their drift, can be eliminated using the feedback of the optical tracking.

Combining several trackers is the key to high accuracy tracking and robustness because it allows adding the trackers' individual strengths and minimizing their weaknesses.

Robotics Aided Model Generation

Figure 3.2 shows a man semi-automatically digitizing the shape of a car. A laser scanner mounted on top of a mechanical linkage is moved around the car, and because the pose of the tip of the robot arm ("hand") can be related to the pose of the laser scanner ("eye"), the three dimensional values obtained by the laser scanner moving along can be used to generate a model of the car.

This scenario can be extended to fully automatically 3D model generation when this arm is exchanged against an robot arm that uses motors in order to move itself. This new scenario can be used to inspect the object which should be modeled from every angle, thus taking care of problems of occlusion, depth focus, resolution, field of view etc.

Loden-Frey, a tailor in Munich, Germany, uses an automatic 3d scanner (see figure 3.3) to minimize the time the customer needs to stay there for taking measurements needed for custom-made clothes.

[1]see http://www.ndigital.com/aurora.php
[2]see http://www.ar-tracking.com/

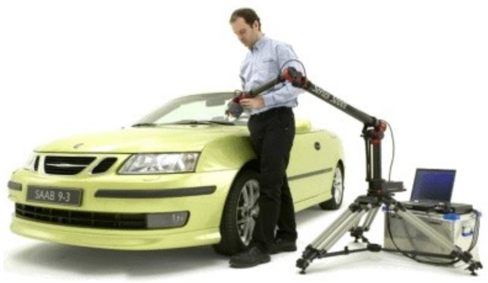


Figure 3.2: semi-automatic scanning of model

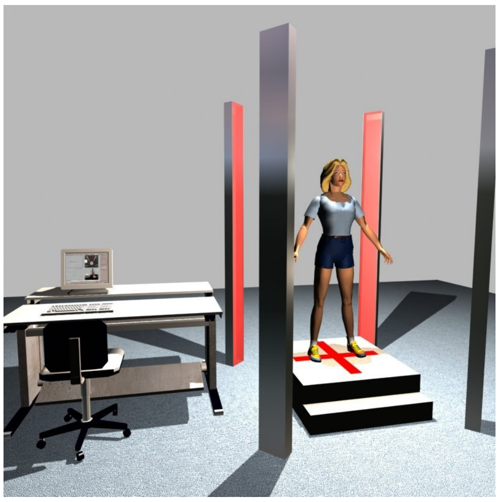


Figure 3.3: full-automatic scanning of human body

Automated Part Acquisition or Assembly

Guided by cameras, robotic arms can be used to grasp and assemble parts. The video camera provides live feedback for the gripper so that the moves of the gripper can be adjusted depending on the images.

When correctly calibrated, a robot arm can also be used for high precision welding – correcting possible interferences of the assembly line (see figure 3.4).

3.2 How to perform Tracker Alignment

The last section explained what tracker alignment is and what can be done once sensors are properly aligned. This section presents different ways how tracker alignment can be achieved.

3.2.1 Forward Engineering

When a new hybrid tracking device is designed the engineer can modify the design in such a way that the tracking devices of with the hybrid tracker consists can be mounted physically

Figure 3.4: welding using robot arms

only in one way – if this is assured, the trackers are aligned a priori; the engineer can easily use his CAD-tool to measure the transformation between the two trackers.

Although this is the ideal solution since no calibration by the user is needed at all, the number of precalibrated hybrid trackers available is extremely low. Another disadvantage is that the hybrid tracker is static, not allowing any modifications.

3.2.2 Manually Registering Points

Another way to align trackers is shown in figure 3.5. There is an A.R.T system (marked green) and a camera (marked blue) which have to be aligned, i.e. the unknown transformation X tracker-to-camera is to be computed.

In the approach described here, the user is asked to use the A.R.T pointer (marked orange) to point on several well-defined spots in a fixed order on a fiducal. Transformation X can be computed since A_C and A_P are measured from the A.R.T system, the geometries of both the fiducal and the A.R.T pointer are known and as the point correspondences provided by the user can be related to the camera pose C measured by the camera itself.

An advantage of this approach is that trackers can be build dynamically in contrast to the previous method. A disadvantage is that the calibration takes much time and heavily depends on how accurate the user pointed at the predefined spots on the fiducal.

3.2.3 Automatic Alignment from Movements

In contrast to the previous method that asks the user to specify point correspondences, the following method yields the more accurate results because it just asks the user[3] to move the trackers until enough samples are evaluated.

[3]In case a robot arm is used, the movements can be predefined wisely. See 3.4.3 for details.

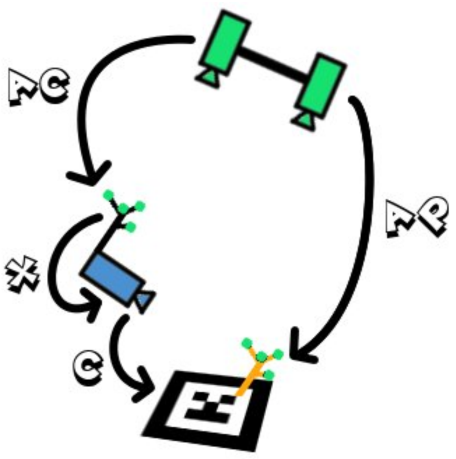


Figure 3.5: tracker alignment by manual registration

How this exactly works is subject of the rest of this paper.

3.3 Setup

In the next section two techniques for automatic tracker alignment will be shown and compared to each other; both approaches use the setup illustrated in this section.

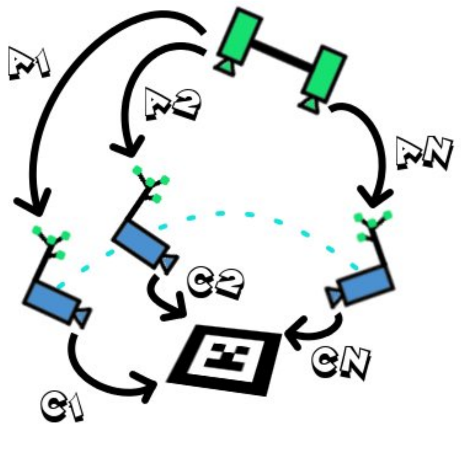


Figure 3.6: measurements taken

Figure 3.6 is a schematic depiction of the setup we use for tracker alignment[4].

[4]Note that this setup is not specific for A.R.T tracking and regular optical tracking, the sensors can be exchanged arbitrarily.

We use a fiducal for measuring pose information with the camera[5] – these measurements are called C_i, their coordinate frame origins at the focal point of the camera. The A.R.T system measures the transformations A_i which are in a different coordinate frame than C_i.

Mathematically speaking we obtain the homogeneous[6] transformation 4×4 matrices A_i from the A.R.T sensor and C_i from 3d pose estimation of the camera:

$$A_i = \begin{bmatrix} R_{A_i} & \vec{t}_{A_i} \\ \vec{0}^T & 1 \end{bmatrix}$$

$$C_i = \begin{bmatrix} R_{C_i} & \vec{t}_{C_i} \\ \vec{0}^T & 1 \end{bmatrix}$$

where R_x is a 3×3 rotation matrix[7], $\vec{t}_x$ is a 3×1 translational vector and $\vec{0} = (0,0,0)^T$.

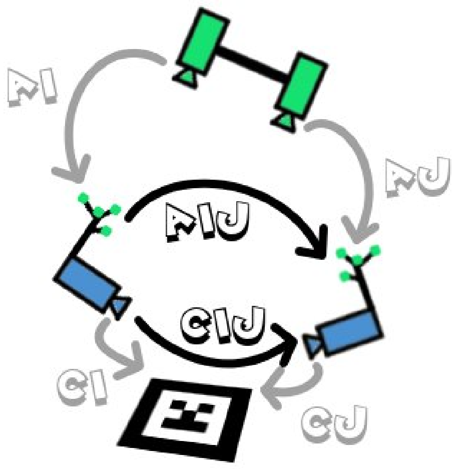


Figure 3.7: transformations computed

Next we select[8] movement pairs out of the measurements and compute the relative movements A_{ij} and C_{ij} for both sensors as shown in figure 3.7.

The interstationary transformations A_{ij} and C_{ij} are computed as follows:

$$A_{ij} = \begin{bmatrix} R_{A_{ij}} & \vec{t}_{A_{ij}} \\ \vec{0}^T & 1 \end{bmatrix} = A_j A_i^{-1}$$

$$C_{ij} = \begin{bmatrix} R_{C_{ij}} & \vec{t}_{C_{ij}} \\ \vec{0}^T & 1 \end{bmatrix} = C_j^{-1} C_i$$

We can use the transformations A_{ij} and C_{ij} to compute the transformation X (shown in figure 3.8) that aligns both trackers:

$$X = \begin{bmatrix} R_X & \vec{t}_X \\ \vec{0}^T & 1 \end{bmatrix}$$

[5]for camera calibration see [5]
[6]for details see 3.6.2
[7]see 3.6.1 for details
[8]for good criterions how movement pairs should be selected see 3.4.3

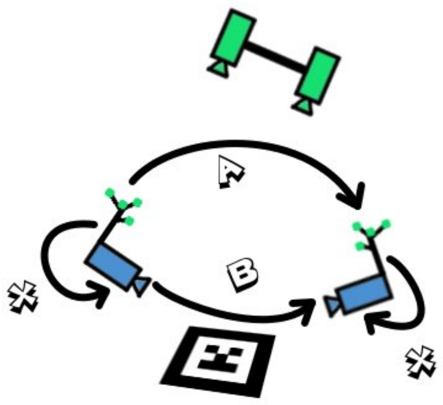


Figure 3.8: AX = XB visualized

Mathematically stated, we search a solution to the following equation:

$$A_{ij}X = XC_{ij}$$

This problem is known under the name $AX = XB$. We rename A_{ij} to A and C_{ij} to B and obtain the fundamental equation:

$$AX = XB \tag{3.1}$$

3.4 Solving AX=XB

This section will present several methods for solving the hand-eye calibration problem. At first, we will present the classical way presented by Tsai [4] which solves for rotation first and translation second. Then a simultaneous approach by Daniilidis [1] for solving for translation and rotation at the same time using dual quaternions will be shown.

3.4.1 Classical Solution

In 1989, Tsai and Lenz [4] used two phases to obtain transformation X. For this split the fundamental equation $AX = XB$ what will be shown in the next section, after that the two phases will be presented.

In the first phase the rotation of X is calculated using Rodrigues representations for the rotation matrix[9]; in the second phase the translation of X is computed and thus X is completely defined.

[9]for Rodrigues formula see 3.6.4

Splitting $AX = XB$

The fundamental equation for tracker alignment

$$AX = XB$$

is written in matrix form

$$\begin{bmatrix} R_A & \vec{t}_A \\ \vec{O}^T & 1 \end{bmatrix} \begin{bmatrix} R_X & \vec{t}_X \\ \vec{O}^T & 1 \end{bmatrix} = \begin{bmatrix} R_X & \vec{t}_X \\ \vec{O}^T & 1 \end{bmatrix} \begin{bmatrix} R_B & \vec{t}_B \\ \vec{O}^T & 1 \end{bmatrix}$$

which is decomposed into two linear equations:

$$R_A R_X = R_X R_B \tag{3.2}$$
$$R_A \vec{t}_X + \vec{t}_A = R_X \vec{t}_B + \vec{t}_X \tag{3.3}$$

Note that the rotational part R_X is separated from the translational part $\vec{t}_s$ in (3.2); Tsai and Lenz use this to compute R_X first, after that $\vec{t}_x$ can be evaluated from (3.3).

Phase 1 - Computing R_X

This phase consists of two steps in order to compute R_X respectively its Rodrigues representation $\vec{r}_x$.

Tsai and Lenz use a modified version of Rodrigues formula to a rotation around the unit vector $\vec{n}$ with angle θ

$$\vec{r} = 2 \sin \frac{\theta}{2} \vec{n}$$

How a rotation matrix R can be obtained from $\vec{r}$ is shown in 3.6.4, the other way round in 3.6.3. In the first phase, $\vec{r}_x$ will be computed instead of R_X directly. To be more precise, $\vec{r'}_x = \lambda \vec{r}_x$ is computed first, second $\vec{r}_x$. $\vec{r'}_x$ is defined as

$$\vec{r'}_x = \frac{1}{2\cos\left(\frac{\theta_{R_X}}{2}\right)} \vec{r}_x = \frac{1}{\sqrt{4 - |\vec{r}_x|^2}} \vec{r}_x$$

Step 1: Compute $\vec{r'}_x$ For all selected transformations (A_{ij}, C_{ij}) a system of linear equations is established

$$(\vec{r}_{A_{ij}} + \vec{r}_{C_{ij}}) \times \vec{r'}_x = \vec{r}_{C_{ij}} - \vec{r}_{A_{ij}} \tag{3.4}$$

with rotation $\vec{r'}_x$ as unknown; using a skew-symmetric matrix[10] instead of the cross-product

[10]see 3.6.5 for $[a]_x$

$$[\vec{r}_{A_{ij}} + \vec{r}_{C_{ij}}]_x \vec{r'}_x = \vec{r}_{C_{ij}} - \vec{r}_{A_{ij}} \tag{3.5}$$

this can be solved uniquely by linear least squares. As the skew-symmetric matrix is always singular, at least two pairs of transformations have to be used.

The proof that (3.5) respectively (3.4) are correct is not shown here due to its length, it can be looked up in [4].

Step 2: Compute $\vec{r}_x$ From $\vec{r'}_x$ we easily obtain $\vec{r}_x$ using the following computation:

$$\vec{r}_x = \frac{2\vec{r'}_x}{\sqrt{1 + |\vec{r'}_x|}}$$

As mentioned before, the rotation $\vec{r}_x$ in Rodrigues representatino can be converted easily to matrix rotation R_X – take a look at 3.6.4 for details.

Phase 2 - Computing $\vec{t}_x$

Once R_X is known, we can easily obtain the translation $\vec{t}_x$ from the following equation derived from (3.3):

$$(R_{A_{ij}} - I)\vec{t}_x = R_X \vec{t}_{Cij} - \vec{t}_{A_{ij}}$$

Again, we use at least two pairs of transformations and solve the resulting system of linear equations by using linear least squares.

3.4.2 Modern Way

In 1998, Daniilidis [1] proposed an approach solving both translation and rotation at the same time, thus having much better error characteristics than Tsai and Lenz's approach. For this "modern" way solving $AX = XB$ he used dual quaternions which are shortly introduced here[11], starting with complex numbers.

Complex Numbers

Quaternions are a non-commutative extension of complex numbers. As our particular interest is rotation in $\mathbb{R}^3$ using quaternions, rotation in $\mathbb{R}^2$ using complex numbers will be described here. For a more general introduction to complex numbers see 3.6.6.

A complex number $c = c_1 + c_2 i = (c_1, c_2)^T$ can also be represented as follows:

$$c = |c|(\cos\theta + \sin\theta i) = |c|(\cos\theta, \sin\theta)^T \tag{3.6}$$

where θ is the angle between the complex number interpreted as vector in the complex plane C and the axis of real numbers (see figure 3.9). θ is also called the argument of a complex number.

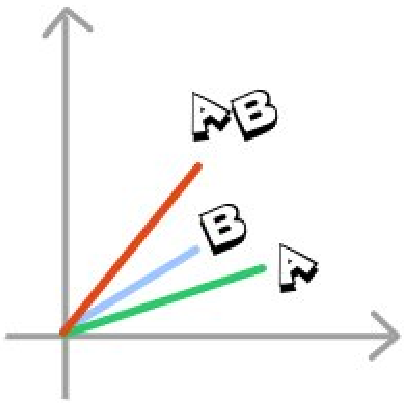


Figure 3.9: rotation in $\mathbb{R}^2$ using complex numbers

Now to the promised rotation in 2d: When multiplying two general complex numbers a and b, their absolute values $|a|$ and $|b|$ are multiplied and their arguments θ_a and θ_b are added. This means a multiplication geometrically is a rotation and a scaling. When multiplying with a unit complex number (i.e. having an absolute value $|c| = 1$) a rotation only is performed.

An example is given in figure 3.9, where a marked green is a vector in $\mathbb{R}^2$ with an angle to the x/real-axis of $\theta_a = 20$. b marked blue is a unit complex number with $\theta_b = 32$ generated using (3.6). The resulting red vector ab is calculated treating a as complex number and multiplying with b. The size of a is preserved because $|b| = 1$, the arguments are added so that $\theta_{ab} = \theta_a + \theta_b = 52$ – a rotation using complex numbers was done.

Quaternions

Quaternions [2] are, as already said, an extension of the complex numbers – in contrast to the complex numbers that algebraicly form a field, the quaternions are non-commutative and hence form a division algebra.

Sir William Rowan Hamilton discovered the quaterions on 1834-10-16 when walking over Brougham Bridge in Dublin, Ireland (see 3.7). A quaternion consists similar to complex numbers of a real and an imaginary part – the imaginary part is now a vector instead of a single scalar. A short introduction to quaternions in general is in 3.6.9, here we focus on rotation only.

For every rotation about an normalized axis $\vec{n}$ with an angle θ a corresponding unit quaternion q can be constructed the following way:

$$q = \begin{pmatrix} \cos\frac{\theta}{2} \\ \sin\frac{\theta}{2}\vec{n} \end{pmatrix}$$

A vector $\vec{x} \in \mathbb{R}^3$ can be rotated by a quaternion q, resulting in $\vec{y} \in \mathbb{R}^3$ using

$$\begin{pmatrix} 0 \\ \vec{y} \end{pmatrix} = q \begin{pmatrix} 0 \\ \vec{x} \end{pmatrix} \bar{q}$$

[11] for longer introduction take a look at [1] and his references about this topic

Superiority of Quaternion in Rotation

In the following, some advantages of quaternions as representations of rotations in comparison to an rotation matrix and Euler angles are described.

size A quaternion only stores 4 values, an orthogonal matrix stores 9.

complexity It is much easier to obtain the rotational angle θ and the axis $\vec{r}$ from a quaternion or construct a quaternion from given θ and $\vec{r}$ than in the rotation matrix of Euler angles representation. Also combining rotations, i.e. multiplying rotation representations, takes less steps when using quaternions instead of rotation matrices.

interpolation When "smooth" rotations are needed e.g. in computer games, this can be much more easily applied with interpolation between quaternions than the other representations. This is called Spherical Linear Interpolation, or SLERP for short.

gimbal lock A gimbal is a device to measure the rotation of an object in 3d using Euler angles. A gimbal lock is when two of the three gimbals cooincide, for details see 3.6.8. This does not occur using quaternions.

robustness If an rotation matrix is, due to numerical instablilties, "off a little", it is not ortogonal anymore, therefore not a representation of a rotation anymore. A quaternion that is "off a little" still represents a valid rotation, furthermore it can be easily renormalized by dividing by its norm. With rotation matrices this is much more complex.

Dual Numbers

The second ingredient to dual quaternions are dual numbers. They were invented by Clifford in 1873 [1], an introduction can be found in 3.6.10.

Dual numbers can be used (in dual vectors) to representate lines in $\mathbb{R}^3$ known as Plücker coordinates[12]:

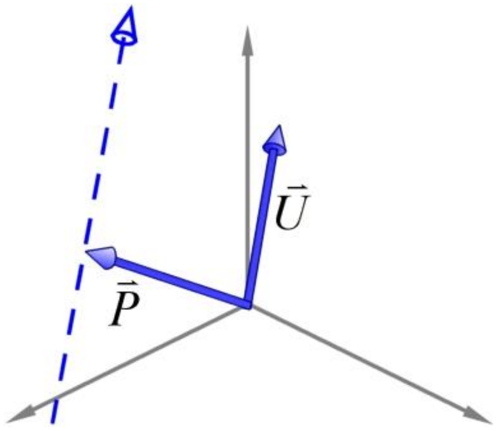


Figure 3.10: Representation of a line in $\mathbb{R}^3$ with moment $\vec{P}$ and direction $\vec{U}$

[12]Information about Plücker coordinates is obtained from a nice article by Lionel Brits available at http://www.flipcode.com/articles/pluecker_part01.shtml; Plücker coordinates are a specialization of Grassmann coordinates.

Given a vector $\vec{p}$ representing a point in $\mathbb{R}^3$ and a vector $\vec{u}$ representating the direction of a line in $\mathbb{R}^3$ (see figure 3.10), i.e. a directed line, the Plücker coordinate corresponding to that line is defined as $L = (\vec{p}, \vec{u} \times \vec{p})^T = (\vec{u}, \vec{v})^T$ where $\vec{u}$ is called line direction and $\vec{v} = \vec{u} \times \vec{p}$ is called the line moment.

Plücker coordinates can also be generated from two arbitrary points $\vec{p}$ and $\vec{q}$ on the line:

$$L = \begin{pmatrix} \vec{p} - \vec{q} \\ \vec{p} \times \vec{q} \end{pmatrix} = \left(\begin{pmatrix} p_1 - q_1 \\ p_2 - q_2 \\ p_3 - q_3 \end{pmatrix}, \begin{pmatrix} p_2 q_3 - p_3 q_2 \\ p_3 q_1 - p_1 q_3 \\ p_1 q_2 - p_2 q_1 \end{pmatrix} \right)^T$$

Some applications achieved with Plücker coordinates are noted in 3.6.11.

Dual Quaternions

Dual quaterions are defined very similar to quaternions – the only difference is that instead of real numbers, dual numbers are used.

We write a dual quaternion $\check{q} = (\check{s}, \check{\vec{q}})^T$ where $\check{s}$ is a dual number and $\check{\vec{q}} = (\check{q_1}, \check{q_2}, \check{q_3})^T$ is a dual vector.

Dual quaternions can also be written as $\check{q} = q + q\prime\epsilon = (q, q\prime)^T$ where q and $q\prime$ are quaternions.

Daniilidis [1] showed that like quaterions can perform rotations with the expression $x_{rot} = qx\bar{q}$, dual quaternions can perform general motions of lines with the expression $\check{c}_{rot} = \check{q}\check{c}\bar{\check{q}}$. In the following section, this will be discussed briefly.

Line Transformations with Unit Dual Quaternions

As we are already familiar with Plücker coordinates from section 3.4.2, we define a line $l = (\vec{l}, \vec{m})^T$. We introduce the special constraint of that the line direction $\vec{l}$ must be a unit vector, i.e. $|\vec{l}| = 1$. Together with the constraint that $\vec{l}$ is perpendicular to $\vec{m}$, i.e. their dot product $\vec{l}^T\vec{m} = 0$ this guarantees that an arbitrary line in space has four degrees of freedom.

Daniilidis shows in [1] that for an arbitrary line l_a transformed with $(R, \vec{t})$ into a line l_b, there exists a unit dual quaternion $\check{q}$ such that $\check{l_b} = \check{q}\check{l_a}\bar{\check{q}}$. The proof of this will be sketched, starting from vector notation, then to quaternions and finally ending at dual quaternions.

vector A transformation $(R, \vec{t})$ applied to line $l_b = (\vec{l_b}, \vec{m}_b)^T$ resulting in l_a can be achieved using the following equations:

$$\vec{l_a} = R\vec{l_b}$$
$$\vec{m}_a = R\vec{m}_b + \vec{t} \times R\vec{l_b}$$

Now we change to quaternion notation, implying that a vector $\vec{x}$ is now represented as quaternion $x = (0, \vec{x})^T$.

quaternion R is represented as a unit quaternion q as shown in 3.4.2. After converting vectors to quaternions and using the identity $(0, \vec{t} \times \vec{q}) = \frac{1}{2}(q\bar{t} + tq)$ we obtain

$$l_a = ql_b\bar{q}$$
$$m_a = qm_b\bar{q} + \frac{1}{2}(ql_b\bar{q}\bar{t} + tql_b\bar{q})$$

dual quaternion Defining $q' = \frac{1}{2}tq$ and $\check{q} = q + q'\epsilon$ and after rearranging the lines to Plücker coordinates, we have

$$l_a + m_a\epsilon = (q + q'\epsilon)(l_b + m_b\epsilon)(\bar{q} + \bar{q}'\epsilon)$$

After also rewriting the lines l_a and l_b as dual quaternions, we finally get a formula quite similar to rotation using quaternions:

$$\check{l_a} = \check{q}\check{l_b}\bar{\check{q}}$$

$\check{q}$ is a unit dual quaternion since $|\check{q}| = 1$.

Note: If $\check{q}$ is a solution, $-\check{q}$ is a solution as well. We enforce an unique solution by the constraint that the non-dual scalar part has to be positive.

Summarizing the relations above, $\check{q} = (q, q')^T$ is assembled by

$$q \text{ from } R \text{ (see 3.6.9)}$$
$$q' = \frac{1}{2}tq$$

Screws and Dual Quaternions

The scalar and vector part of a unit dual quaternion are related to screws. This will be quickly shown here.

According to Chasles' theorem [1], every rigid transformation can be represented as a rotation about an axis not through the origin and a translation along that axis (pitch). Furthermore, the angle of rotation θ is the same both in the rigid transformation and the screw.

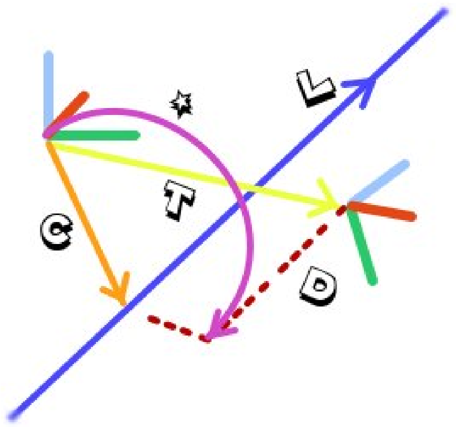


Figure 3.11: geometry of a screw

Figure 3.11 shows a typical screw; l is the rotation axis, d is the so called pitch, $*$ is the angle of the rotation. Furthermore, the translation $\vec{t}$ of the original transformation is shown.

Daniilidis [1] shows that the parameters d and the rotation axis described by $(\vec{l}, \vec{m})^T$ defining the screw can be obtained as follows:

direction $\vec{l}$ The direction of the rotation axis $\vec{l}$ of the screw is parallel to the original rotation axis, i.e. the direction is the same.

pitch d The pitch d of the screw is the projection of the translation on the rotation axis:

$$d = \vec{t}^T \vec{l}$$

moment $\vec{m}$ Introducing the auxiliary point c, the moment of the rotation axis can be computed:

$$\begin{aligned} \vec{m} &= \vec{c} \times \vec{l} \\ &= \frac{1}{2}(\vec{t} \times \vec{l} + \vec{l} \times (\vec{t} \times \vec{l}) \cot \frac{\theta}{2}) \end{aligned}$$

Note that the moment $\vec{m}$ and hence the screw axis is not defined if θ is either 0 or 180.

The next step towards tracker alignment with dual quaternions is the computation of the unit dual quaternion $\check{q}$ representing the screw. This can be done as follows[13]:

$$\begin{aligned} \check{q} &= \begin{pmatrix} \cos \frac{\theta}{2} \\ \sin \frac{\theta}{2} \vec{l} \end{pmatrix} + \begin{pmatrix} -\frac{d}{2} \sin \frac{\theta}{2} \\ \sin \frac{\theta}{2} \vec{m} + \frac{d}{2} \cos \frac{\theta}{2} \vec{l} \end{pmatrix} \epsilon \\ &= \begin{pmatrix} \cos \left(\frac{\theta + d\epsilon}{2}\right) \\ \sin \left(\frac{\theta + d\epsilon}{2}\right)(\vec{l} + \vec{m}\epsilon) \end{pmatrix} \\ &= \begin{pmatrix} \cos \frac{\check{\theta}}{2} \\ \check{\vec{l}} \sin \frac{\check{\theta}}{2} \end{pmatrix} \end{aligned}$$

where $\check{\theta} = \theta + d\epsilon$ and $\check{\vec{l}} = \vec{l} + \vec{m}\epsilon$. This formula again looks quite similar to the equation of pure rotation for quaterions.

Using Dual Quaternions for Tracker Alignment

The fundamental equation $AX = XB$ written using dual quaternions is

$$\check{a}\check{q} = \check{q}\check{b}$$

or equivalently but more similar to usual quaterion notation

$$\check{a} = \check{q}\check{b}\bar{\check{q}} \tag{3.7}$$

Note that (3.7) is the most compact equation for tracker alignment because the homogeneous transformations in (3.1) consist of 12 components each in contrast to the 8 components needed for each dual quaternion.

According to the Screw Congruence Theorem[14] the pitch and the angle of a screw remain invariant under coordinate transformations. As this means that the scalar parts of $\check{a}$ and $\check{b}$ are equal only the vector parts are used for computing $\check{q}$:

$$\sin \frac{\check{\theta}_a}{2} \begin{pmatrix} 0 \\ \check{\vec{a}} \end{pmatrix} = \check{q} \begin{pmatrix} 0 \\ \sin \frac{\check{\theta}_b}{2} \check{\vec{b}} \end{pmatrix} \bar{\check{q}} = \sin \frac{\check{\theta}_b}{2} \check{q} \begin{pmatrix} 0 \\ \check{\vec{b}} \end{pmatrix} \bar{\check{q}}$$

[13]for a derivation of the conversion screw to dual quaternion see [1]
[14]elegantly proven with dual quaternions in [1]

If the angles θ_a and θ_b are neither 0 nor 360, this can be simplified to

$$\begin{pmatrix} 0 \\ \check{\vec{a}} \end{pmatrix} = \check{q} \begin{pmatrix} 0 \\ \check{\vec{b}} \end{pmatrix} \bar{\check{q}}$$

which is just the motion of the screw axes.

Solving $\check{a} = \check{q}\check{b}\bar{\check{q}}$

As shown in the last paragraph, we do not have to compute the scalar part of the dual quaternions. Thus we re-define $\check{a} = (0, \check{\vec{a}})$ and $\check{b} = (0, \check{\vec{b}})$.

We split the fundamental equation (3.7) into a dual and a non-dual part as follows:

$$\begin{aligned} a &= qb\bar{q} \\ a' &= qb\bar{q'} + qb'\bar{q} + q'b\bar{q} \end{aligned}$$

After multiplying on the right with q and using the identity $\bar{q}q' + \bar{q'}q = 0$, this can be written as

$$\begin{aligned} aq - qb &= 0 \\ (a'q - qb') + (aq' - q'b) &= 0 \end{aligned}$$

Disregarding the scalar parts, this can be written in matrix form:

$$S \begin{pmatrix} q \\ q' \end{pmatrix} = 0$$

where S is the 6×8 matrix

$$\begin{pmatrix} \vec{a} - \vec{b} & [\vec{a} + \vec{b}]_x & 0_{3\times 1} & 0_{3\times 3} \\ \vec{a'} - \vec{b'} & [\vec{a'} + \vec{b'}]_x & \vec{a} - \vec{b} & [\vec{a} + \vec{b}]_x \end{pmatrix}$$

and $\vec{x}$ is the vector part of the corresponding quaternion x.

Next we construct the $6n \times 8$ matrix T from n motion pairs

$$T = \begin{pmatrix} S_1 \\ S_2 \\ \vdots \\ S_n \end{pmatrix}$$

and compute the Singular Value Decomposition (SVD) $T = USV^T$; if T is of rank 6, i.e. not all screw axes are mutually parallel, the two last right singular vectors $\vec{v_7}$ and $\vec{v_8}$ span the nullspace of T, hence:

$$\begin{aligned} \begin{pmatrix} q \\ q' \end{pmatrix} &= \lambda_1 \vec{v_7} + \lambda_2 \vec{v_8} \\ &= \lambda_1 \begin{pmatrix} \vec{x_1} \\ \vec{y_1} \end{pmatrix} + \lambda_2 \begin{pmatrix} \vec{x_2} \\ \vec{y_2} \end{pmatrix} \end{aligned} \quad (3.8)$$

where the 8×1 vectors $\vec{v_7}$ and $\vec{v_8}$ are each split into two 4×1 vectors $(\vec{x_1}, \vec{y_1})^T$ respectively $(\vec{x_2}, \vec{y_2})^T$.

Using the two constraints we have on the the quaternions

$$q^T q = 1 \text{ and } q^T q' = 0$$

we obtain the following two quadratic equations in λ_1 and λ_2:

$$\lambda_1^2 \vec{x_1}^T \vec{x_1} + 2\lambda_1\lambda_2 \vec{x_1}^T \vec{x_2} + \lambda_2^2 \vec{x_2}^T \vec{x_2} = 1 \tag{3.9}$$

$$\lambda_1^2 \vec{x_1}^T \vec{y_1} + \lambda_1\lambda_2(\vec{x_1}^T \vec{y_2} + \vec{x_2}^T \vec{y_1}) + \lambda_2^2 \vec{x_2}^T \vec{y_2} = 0 \tag{3.10}$$

Because λ_1 and λ_2 never both vanish, we assume that $\vec{x_1}^T \vec{y_1} \neq 0$ so that $\lambda_2 \neq 0$. Furthermore we introduce a new variable $s = \lambda_1/\lambda_2$ and rewrite (3.9) to obtain

$$\lambda_2^2(s^2 \vec{x_1}^T \vec{x_1} + 2s \vec{x_1}^T \vec{x_2} + \vec{x_2}^T \vec{x_2}) = 1 \tag{3.11}$$

which delievers us two solutions for s. For each of these solutions we compute $s^2 \vec{x_1}^T \vec{x_1} + 2s \vec{x_1}^T \vec{x_2} + \vec{x_2}^T \vec{x_2}$ and choose the s that returns the largest value. Using (3.11) to obtain λ_2 and from this directly $\lambda_1 = s\lambda_2$, the resulting transformation for tracker alignment is computed via (3.8).

The rotation R and translation $\vec{t}$ of the resulting dual quaternion $\check{q}$ can be easily extracted: the non-dual part q corresponds to R, $\vec{t}$ can be computed as $\vec{t} = 2q'\bar{q}$.

3.4.3 Improving Accuracy

Tsai and Lenz give several recommendations in [4] to improve accuracy, most of these are specific to hand-eye-calibration, i.e. tracker alignment using a robot and a camera. Their applicable recommendations together with those of Schmidt et al [3] are shortly presented here.

Selecting Movement Pairs

If movement pairs are obtained simply by their temporal order, there usually is no big difference between the movement pairs. For tracker alignment, two movements with different rotation axes have to be used anyhow, otherwise there will be no reasonable result.

Schmidt et al [3] propose as optimality criterion for the selection of movements the absolute value of the scalar product between the two rotation axes of the sensor that inherently yields better accuracy:

$$s_{ij,kl} = |a_{ij}^T a_{kl}| \tag{3.12}$$

where a_{ij} is the rotation axis of the movement from station i to j. The sum of all $s_{ij,kl}$ should be minimal.

If the rotation axes of two movements would be orthonormal but the rotation itself is very little (only a small angle used), this movement pair is ill-suited for tracker alignment as well. Because of this, a threshold angle θ is used for preselecting movement pairs; those selected, i.e. those with an angle α fulfilling $\theta \leq \alpha \leq (\pi - \theta)$, are examined using (3.12).

Eliminating Outliers

The more movement pairs used, the more accurate the result will be. So far the theory. Using real measurements, a single outlier, i.e. a wrong measurement / result of great magnitude, can have a devastating effect to the accuracy of the whole tracker alignment.

For solving this problem, a RANSAC approach like the following can be used.

1. Select m random samples from the movement pairs that pass preselection (see 3.4.3), where each sample consists of 2 movements (minimum for tracker alignment).
2. Compute the R_{X_i} and $\vec{t}_{X_i}$ for all samples.
3. Perform another series of movements and determine the error between the copmuted transforms and the obtained movements A and B. Apply a threshold for the error and keep only consistent movement pairs.
4. Compute R_X and $\vec{t}_X$ using all consistent movement pairs left.

3.5 Conclusion

This paper presented an introduction for tracker alignment. Several applications were mentioned as well as general methods how tracker alignment can be accomplished. Two different methods were shown in detail, a two stepped algorithm by Tsai and a more robust solution by Daniilidis. Experiments in [1] demonstrated the superiority of the method using dual quaternions, resulting in lower errors both in rotation and translation.

3.6 Mathematical Definitions

3.6.1 Rotation Matrix

A rotation matrix performing a rotation around unit vector $\vec{n} = (n_1, n_2, n_3)^T$ with angle θ is can be computed from axis-angle representation as follows:

$$\begin{aligned}
R &= \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \\
\textit{where} & \\
r_{11} &= n_1^2 + (1 - n_1^2)\cos\theta \\
r_{12} &= n_1 n_2 (1 - \cos\theta) - n_3 \sin\theta \\
r_{13} &= n_1 n_3 (1 - \cos\theta) + n_2 \sin\theta \\
r_{21} &= n_1 n_2 (1 - \cos\theta) + n_3 \sin\theta \\
r_{22} &= n_2^2 + (1 - n_2^2)\cos\theta \\
r_{23} &= n_2 n_3 (1 - \cos\theta) - n_1 \sin\theta \\
r_{31} &= n_1 n_3 (1 - \cos\theta) + n_2 \sin\theta \\
r_{32} &= n_2 n_3 (1 - \cos\theta) + n_1 \sin\theta \\
r_{33} &= n_3^2 + (1 - n_3^2)\cos\theta
\end{aligned}$$

Note that the rows and columns both are orthogonal to each other and that the matrix is normalized.

3.6.2 Homogeneous Transformation

Using homogeneous transformations, we have a unified way to describe both translation and rotation as matrices, allowing to apply a rigid transformation $R\vec{x} + \vec{t}$ by multipling with a single 4×4 matrix.

Homogeneous transform for rotation only:

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Homogeneous transform for translation only

$$T = \begin{bmatrix} 1 & 0 & 0 & t_1 \\ 0 & 1 & 0 & t_2 \\ 0 & 0 & 1 & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Homogeneous transform for both translation and rotation

$$Tr = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Extracting the rotation matrix R from a transformation Tr is very simple, it is direcly given as the upper left 3×3 matrix. The translation vector is also easily extracted, it's in the upper right column.

3.6.3 Extracting Rotation Axis and Angle from Rotation Matrix

From Shiu and Ahmad [6] there is the following way to extract the rotation axis $\vec{n} = (n_1, n_2, n_3)^T$ and its according angle θ from a rotation matrix[15]:

$$\begin{aligned} \cos\theta &= \frac{1}{2}(r_{11} + r_{22} + r_{33} - 1) \\ \text{and} & \\ \sin\theta &= \pm\frac{1}{2}\sqrt{a + b + c} \\ \text{where} & \\ a &= r_{32} - r_{23}^2 \\ b &= r_{13} - r_{31}^2 \\ c &= r_{21} - r_{12}^2 \end{aligned}$$

[15]rotation matrix is defined in 3.6.1

Since our angle θ is $0 \leq \theta \leq \pi$ this leads us to a unique solution:

$$\theta = \arctan \frac{\sin\theta}{\cos\theta}$$

From θ we can compute the rotation axis $\vec{n}$ by three equations. Which equations we compute depends on whether r_{11}, r_{22} or r_{33} is the most positive.

If r_{11} is most positive

$$\begin{aligned}
n_1 &= \operatorname{sgn}(r_{32} - r_{23})\sqrt{\frac{r_{11} - \cos\theta}{1 - \cos\theta}} \\
n_2 &= \frac{r_{21} + r_{12}}{2n_1(1 - \cos\theta)} \\
n_3 &= \frac{r_{13} + r_{31}}{2n_1(1 - \cos\theta)} \\
\text{where} & \\
\operatorname{sgn} x &= \begin{cases} +1 & \text{if } x \geq 0 \\ -1 & \text{otherwise} \end{cases}
\end{aligned}$$

If r_{22} is most positive

$$\begin{aligned}
n_2 &= \operatorname{sgn}(r_{13} - r_{31})\sqrt{\frac{r_{22} - \cos\theta}{1 - \cos\theta}} \\
n_1 &= \frac{r_{21} + r_{12}}{2n_2(1 - \cos\theta)} \\
n_3 &= \frac{r_{32} + r_{23}}{2n_2(1 - \cos\theta)}
\end{aligned}$$

If r_{33} is most positive

$$\begin{aligned}
n_3 &= \operatorname{sgn}(r_{21} - r_{12})\sqrt{\frac{r_{33} - \cos\theta}{1 - \cos\theta}} \\
n_1 &= \frac{r_{31} + r_{13}}{2n_3(1 - \cos\theta)} \\
n_2 &= \frac{r_{23} + r_{23}}{2n_3(1 - \cos\theta)}
\end{aligned}$$

No matter which factor is the most positive one, we always get a unique θ and $\vec{n}$ for each rotational matrix.

3.6.4 Modified Rodrigues Formula

This modified Rodrigues formula is used by Tsai and Lenz [4] as substitute for a rotation matrix defined in 3.6.1:

$$\vec{r} = 2\sin\frac{\theta}{2}\vec{n}$$

where $0 \leq \theta \leq \pi$. Using this representation, a rotation matrix R can be calculated the following way:

$$\begin{aligned}
R = \left(1 - \frac{|\vec{r}|^2}{2}\right) I + \\
\frac{1}{2}(\vec{r}\vec{r}^T + \sqrt{4 - |\vec{r}|^2}[\vec{r}]_x)
\end{aligned}$$

For definintion of $[\vec{r}]_x$ see 3.6.5.

Note that this conversion does not contain any trigonomic functions in contrast to usual Rodrigues formula to rotation matrix conversions.

3.6.5 Skew-symmetric Matrix $[\vec{a}]_x$

$[\vec{a}]_x$ is a skew-symmetric matrix generated from the three dimensional vector $\vec{a} = (a_1, a_2, a_3)^T$ as follows:

$$[\vec{a}]_x = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}$$

A skew symmetric (also called antisymmetric) matrix A has the following characteristic[16]: $A = -A^T$ – for a symmetric matrix this is $A^T = A$.

A skew symmetric matrix $[\vec{a}]_x$ multiplied with another vector $\vec{b}$ is equivalent to the cross product $\vec{a} \times \vec{b}$, i.e.

$$\vec{a} \times \vec{b} = [\vec{a}]_x \vec{b} = (\vec{a}^T [\vec{b}]_x)^T$$

3.6.6 Complex Numbers

A complex number $z = a + bi = (a, b)^T$ consists of a real part a and an imaginary part b. i is a square root of -1, i.e. $i^2 = -1$. Complex numbers[17] were first used in the 17th century and proved the Fundamental Theorem of Algebra (see 3.6.7). For us, complex numbers are of particular interest as they can be used to implement rotation in $\mathbb{R}^2$ as shown in 3.4.2.

The basic calculation rules addition, subtraction, multiplication and division of complex numbers are defined (both in regular representation and in vector-form) as follows:

Addition

$$(a + bi) + (c + di) = (a + c) + (b + d)i$$
$$\begin{pmatrix} a \\ b \end{pmatrix} + \begin{pmatrix} c \\ d \end{pmatrix} = \begin{pmatrix} a + c \\ b + d \end{pmatrix}$$

Subtraction

$$(a + bi) - (c + di) = (a - c) + (b - d)i$$
$$\begin{pmatrix} a \\ b \end{pmatrix} - \begin{pmatrix} c \\ d \end{pmatrix} = \begin{pmatrix} a - c \\ b - d \end{pmatrix}$$

Multiplication

$$\begin{aligned}(a + bi)(c + di) =&(ac - bd) \\ &+ (ad + bc)i\end{aligned}$$
$$\begin{pmatrix} a \\ b \end{pmatrix} \begin{pmatrix} c \\ d \end{pmatrix} = \begin{pmatrix} ac - bd \\ ad + bc \end{pmatrix}$$

[16]taken from Eric W. Weisstein et al. "Antisymmetric Matrix." From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/AntisymmetricMatrix.html

[17]nice overview is at http://www.clarku.edu/ djoyce/complex/

Division

$$\frac{(a+bi)}{(c+di)} = \frac{(ac+bd)+(bc-ad)i}{c^2+d^2}$$
$$\frac{\begin{pmatrix} a \\ b \end{pmatrix}}{\begin{pmatrix} c \\ d \end{pmatrix}} = \frac{1}{c^2+d^2}\begin{pmatrix} ac+bd \\ bc-ad \end{pmatrix}$$

The conjugate of a complex number $x = a + bi = (a, b)^T$ is denoted as $\bar{a} = a - bi = (a, -b)$; the absolute value of a complex number x is $|x| = \sqrt{a^2 + b^2} = |(a, b)^T|$.

3.6.7 Fundamental Theorem of Algebra

The Fundamental Theorem of Algebra states that every polynom of degree n has exactly n zeroes:

$$\begin{aligned} f(x) &= x^n + a_{n-1}x^{n-1} + \cdots + a_0 \\ &\text{can be rearranged to} \\ f(x) &= (x - z_1)(x - z_2)\cdots(x - z_n) \end{aligned}$$

where a_i and z_i are complex numbers.

3.6.8 Gimbal Lock

A gimbal lock is the coincidence when two rotational axis of an object are pointing in the same direction. This means that an object won't rotate the way it is expected to rotate. Gimbal locks occur when using Euler angles as means to rotation as in this representation, the Euler angles are evaluated for each axis indepenently and in a fixed order.

Here is an example of a gimbal lock[18]:

Assume that an object is being rotated in the order Z,Y,X and that the rotation in the Y-axis is 90 degrees. In this case, rotation in the Z-axis is performed first and therefore correctly. The Y-axis is also rotated correctly. However, after rotation in the Y axis, the X-axis is rotated onto the Z-axis.

Thus, any rotation in the X-axis actually rotates the object in the Z-axis. Even worse, it begins to rotate the object in the X-axis.

A convinient solution to this problem is to make use of quaternions.

3.6.9 Quaternions

A quaternion is a 4-tupel consisting of one real part and three imaginary parts; it can be written as $q = a + bi + cj + dk = (a, b, c, d)^T$ where a, b, c and d are real numbers, a is the real part, b, c and d form the imaginary part, and i, j and k are imaginary units for that the

[18]taken from matrix faq of gamedev.net

following multiplication table applies:

	i	j	k
i	-1	k	$-j$
j	$-k$	-1	i
k	j	$-i$	-1

We can also write a quaternion q as

$$\begin{aligned} q &= \begin{pmatrix} s \\ \vec{q} \end{pmatrix} \\ &= s + q_1 i + q_2 j + q_3 k \end{aligned}$$

with $s \in \mathbb{R}$ and $\vec{q} = (q_1, q_2, q_3)^T \in \mathbb{R}^3$. A quaternion q can be constructed from a vector $\vec{x}$ by setting the real part to zero, i.e. $x = (0, \vec{x})$. The conjugate $\bar{q}$ of a quaternion q is defined as

$$\begin{aligned} \bar{q} &= \begin{pmatrix} s \\ -\vec{q} \end{pmatrix} \\ &= s - q_1 i - q_2 j - q_3 k \end{aligned}$$

General calculation rules using $p = (r, \vec{p})^T$ and $q = (s, \vec{q})^T$ are defined as follows:

Addition

$$\begin{aligned} p + q &= \begin{pmatrix} r \\ \vec{p} \end{pmatrix} + \begin{pmatrix} s \\ \vec{q} \end{pmatrix} \\ &= \begin{pmatrix} r + s \\ \vec{p} + \vec{q} \end{pmatrix} \end{aligned}$$

Subtraction

$$\begin{aligned} p - q &= \begin{pmatrix} r \\ \vec{p} \end{pmatrix} - \begin{pmatrix} s \\ \vec{q} \end{pmatrix} \\ &= \begin{pmatrix} r - s \\ \vec{p} - \vec{q} \end{pmatrix} \end{aligned}$$

Multiplication[19]

$$\begin{aligned} pq =& rs + \vec{q}\vec{p} + s\vec{p} + r\vec{q} + \vec{p} \times \vec{q} \\ =& \begin{pmatrix} rs - p_1 q_1 - p_2 q_2 - p_3 q_3 \\ q_1 r + s p_1 + q_2 p_3 - q_3 p_2 \\ q_2 r + s p_2 + q_3 p_1 - q_1 p_3 \\ q_3 r + s p_3 + q_1 p_2 - q_2 p_1 \end{pmatrix} \end{aligned}$$

Note that since the multiplication of quaternions uses a cross-product, it is not commutative.

The norm of a quaternion is also defined similar to complex numbers:

$$\begin{aligned} |q| &= \sqrt{q\bar{q}} \\ &= \sqrt{\begin{pmatrix} s \\ \vec{q} \end{pmatrix} \begin{pmatrix} s \\ -\vec{q} \end{pmatrix}} && = \sqrt{s^2 + |\vec{q}|^2} \\ &= \sqrt{s^2 + q_1^2 + q_2^2 + q_3^2} \end{aligned}$$

[19]This way of multiplying quaternions is also called the Grassmann product

Unit quaterions are quaternions with a norm of 1, i.e. $|q| = 1$, these are used to represent rotations in 3.4.2.

A quaternion can be used to calculate the cross-product of two vectors $\vec{p}$ and $\vec{q}$ by generating the two quaterions $p = (0, \vec{p})$ and $q = (0, \vec{q})$, multipying them to generate the new quaterion $r = (s, \vec{r})$. The vector $\vec{r}$ is the cross-product $\vec{p} \times \vec{q}$, the scalar part s is the dot-product $\vec{p}\vec{q}$ with opposite sign.

3.6.10 Dual Numbers

A dual number are defined as

$$\check{x} = a + b\epsilon = \begin{pmatrix} a \\ b \end{pmatrix}$$

with $\epsilon^2 = 0$ and $a, b \in \mathbb{R}$.

General calculation rules for two dual numbers $\check{a} = (a_1, a_2)^T$ and $\check{b} = (b_1, b_2)^T$ are defined as follows:

Addition

$$\begin{aligned} \check{a} + \check{b} &= \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \\ &= \begin{pmatrix} a_1 + b_1 \\ a_2 + b_2 \end{pmatrix} \end{aligned}$$

Subtraction

$$\begin{aligned} \check{a} - \check{b} &= \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} - \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \\ &= \begin{pmatrix} a_1 - b_1 \\ a_2 - b_2 \end{pmatrix} \end{aligned}$$

Multiplication

$$\begin{aligned} \check{a}\check{b} &= \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \\ &= (a_1 + a_2\epsilon)(b_1 + b_2\epsilon) \\ &= a_1b_1 + a_1b_2\epsilon + a_2b_1\epsilon + a_2b_2\epsilon^2 \\ &= a_1b_1 + (a_1b_2 + a_2b_1)\epsilon \\ &= \begin{pmatrix} a_1b_1 \\ a_1b_2 + a_2b_1 \end{pmatrix} \end{aligned}$$

Division

$$\frac{\check{a}}{\check{b}} = \begin{pmatrix} \frac{a_1}{b_1} \\ \frac{a_2}{b_1} - \frac{a_1b_2}{b_1^2} \end{pmatrix}$$

for any $b_1 \neq 0$. This is why the dual numbers form an abelian i.e. commutative ring but not a field.

Our main use of dual numbers in this papers is for their representation of lines, see 3.4.2.

3.6.11 Application of Plücker Coordinates

Plücker coordinated can be used to in following applications:

check if two lines are identical Given L_1 and L_2 are Plücker coordinates, they are the same line if $L_1 = \alpha L_2$ with $\alpha \in \mathbb{R}\backslash\{0\}$. If α is negative, the lines have opposite directions.

check if two lines intersect Given $L_1 = (\vec{U_1}, \vec{V_1})$ and $L_2 = (\vec{U_2}, \vec{V_2})$, β is the sum of their dot products $\beta = U_1V_1 + U_2V_2$, the following three cases can occur:

- $\beta = 0$ L_1 and L_2 do intersect.
- $\beta < 0$ L_1 and L_2 pass clockwise (see figure 3.12).
- $\beta > 0$ L_1 and L_2 pass counterclockwise.

check if a line intersects a triangle If a line L hits a triangle given by three lines T_1, T_2 and T_3, which all point either clockwise or counterclockwise, an intersection is applied when

- L **hits one or two lines** $T_i \in \{T_1, T_2, T_3\}$ If L intersects one of the lines defining the triangle directly, one or two of the βs from the above are zero.
- L **hits triangle somewhere else** If L intersects somewhere else, all βs have the same sign.

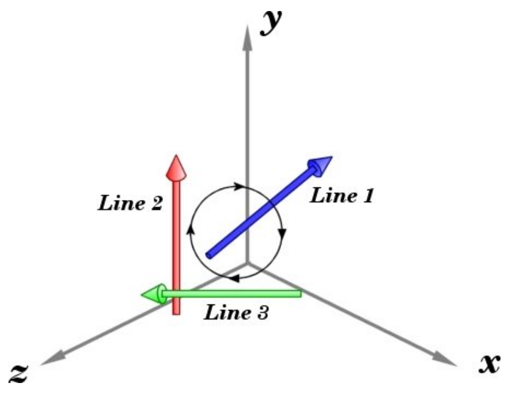


Figure 3.12: Lines 2 and 3 pass line 1 clockwise

3.7 History of Quaternions

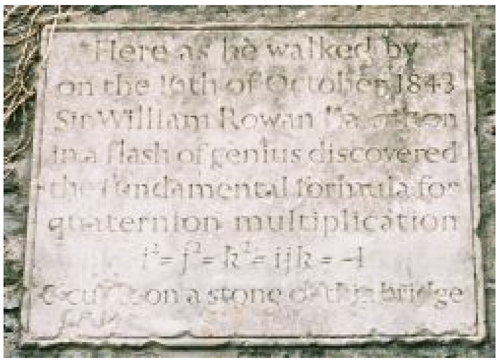

Here as he walked by
on the 16th of October 1843
Sir William Rowan Hamilton
in a flash of genius discovered
the fundamental formula for
quaternion multiplication
$i^2 = j^2 = k^2 = ijk = -1$
& cut it on a stone of this bridge

Figure 3.13: Plaque at the Brougham Bridge in Dublin, Ireland

Sir William Rowan Hamilton is said to have discovered the central concept of quaternions when taking a walk with his wife; on the Brougham[20] Bridge, he had the idea and was so fascinated that he curved it into the bridge. In memory of this moment, there is a plaque (see figure 3.13) with the following text:

> Here as he walked by on the 16th of October 1843 Sir William Rowan Hamilton in a flash of genius discovered the fundamental formula for quaternion multiplication
> $i^2 = j^2 = k^2 = ijk = -1$
> & cut it on a stone of this bridge

[20] The Broughman Bridge is nowadays called Broom Bridge

Bibliography

[1] K. Daniilidis, *Hand-eye calibration using dual quaternions*, Int. Journ Robotics Res, 18 (1999), pp. 286–298.

[2] S. W. R. Hamilton, *On Quaternions*, Proceedings of the Royal Irish Academy, 3 (1844), pp. 1–16.

[3] H. N. J. Schmidt, F. Vogt, *Robust Hand-Eye Calibration of an Endoscopic Surgery Robot Using Dual Quaternions*, Pattern Recognition, 25th DAGM Symposium, LNCS 2781, (1999), pp. 548–556.

[4] R. L. R. Tsai, *A new technique for fully autonomous and efficient 3D robotics hand/eye calibration*, IEEE Transactions on Robotics and Automation, 5 (1989), pp. 345–358.

[5] R. Y. Tsai, *Versatile camera calibration technique for high-accuracy 3-D machine vision metrology using off-the-shelf TV cameras and lenses*, IEEE Journal of Robotics and Automation, 3 (1987), pp. 323–344.

[6] S. A. Y. Shiu, *Calibration of wrist-mounted robotic sensors by solving homogeneous transform equations of the form AX=XB*, IEEE Transactions on Robotics and Automation, 5 (1989), pp. 16–29.

4 Sensor Fusion: The Kalman Filter and its Extensions

Katharina Pentenrieder, Technische Universität München

Sensor Fusion aims to coveniently integrate data from different sensors in order to compute a dynamic systems's state. Here problems arise out of uncertainty. In general the knowledge about the system is incomplete and the given data is corrupted with noise. Hence the state of the system can only be estimated.

The Kalman Filter is a very robust and popular approach for stochastic estimation given noisy measurements. This paper therefore concentrates on the ideas of Kalman Filtering for linear and non-linear process models and the application of the filter for the task of combining different data sets. Furthermore some extensions of the Kalman Filter are presented which are useful for sensor fusion: the SCAAT method and the Federated Kalman Filter.

4.1 Introduction

Our task is sensor fusion, we want to combine data sets received from different sensors. A very simple approach would be to take all the data available together with a nicely derived deterministic process model and calculate a least-squares solution. So given this result, why should we consider approaches which are a lot more difficult? Why should be bother about stochastic models or estimation concepts?

The answer is given by the nature of the regarded systems. Deterministic models and control theories do not suffice for the analysis of general physical systems, whether they handle chemical processes or economy. Our models for these processes will mostly be imperfect and can only approximate the system. Furthermore the data is imperfect or incomplete. And finally the system is driven by disturbances which can neither be modelled nor controlled deterministically [5]. Hence we cannot assume perfect knowledge or perfect control over the system.

Knowing about these difficulties we need to develop a system model which takes these uncertainties into account and optimally estimates the desired quantities. Here we find the Kalman Filter - an optimal linear recursive estimator. The filter incorporates all available system information to estimate the current value of the variables of interest such that the estimated error is minimized when some presumed conditions are met. Because of its simplicity and robustness the Kalman Filter is subject of extensive research and application, for example in Computer Graphics. The filter even works well if the conditions necessary for optimal estimation are not met. It uses knowledge about the system and the measurement device, statistical descriptions of process noise, measurement error and uncertainties and any available information about initial conditions of the variables. Unlike other data processing algorithms the Kalman Filter does not need to reprocess all previous data every time new measurements arrive but computes new estimates based on the last estimate and current measurements. It can therefore be regarded as a recursive algorithm.

The basis of the filter is provided by the "prediction-correction-cycle" shown in figure 4.1. The variables of interest are stored in a so called state vector and an initial estimate is used to start the cycle. Then the current state estimate and error covariance are projected ahead in the time update step in order to get the *a priori* estimates for the next step. After that the current measurements are incorporated as a feedback in the measurement update step to obtain improved *a posteriori* estimates.

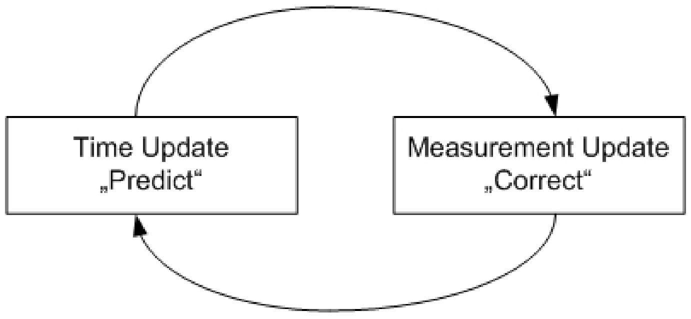


Figure 4.1: Kalman Filter Cycle

As we see, the Kalman Filter is a good approach for stochastic estimation from noisy sensor measurements. Because of its popularity and appropriateness for our purpose of data fusion we will concentrate on the ideas of the filter, its extensions and applications.

In the following sections the filter will be described in detail. First the simple Discrete Kalman Filter, its models and update equations are presented in section 4.3. This filter deals with linear models. For the case of non-linear systems the Extended Kalman Filter can be applied which is discussed afterwards in section 4.4. Finally section 4.6 considers Sensor Fusion using the Kalman Filter and provides examples and applications of the introduced filters. But before we consider the filter and its extensions a short introduction to the necessary probabilistic basics is given.

4.2 Stochastic Basics

This section can by no means give a full introduction to stochastics. It will only provide the definitions which are necessary for the following explanations on the Kalman Filter.

4.2.1 Probability and Random Variables

The *probability* that the outcome of a discrete event favors a particular event A is given by

$$p(A) = \frac{\text{possible outcomes favoring A}}{\text{total number of possible outcomes}}$$

For two outcomes A and B the probability $p(A \cup B) = p(A) + p(B)$ and for independent outcomes $p(A \cap B) = p(A)p(B)$. The conditional probability of outcome A given B is defined as

$$p(A|B) = \frac{p(A \cap B)}{p(B)}$$

A *random variable* is essentially a function that maps all points in the sample space to numbers. X(t) for example might map time to the expected position of an object. For continuous random variables the probability of a single discrete event is 0, hence we rather evaluate intervals. A common function is the *cumulative distribution function* $F_X(x) = p(-\infty, x]$. Even more important is its derivative, the *probability density function*:

$$f_X(x) = \frac{\partial}{\partial x} F_X(x)$$

With f, the probability over an interval $[a, b]$ is defined as

$$p_X[a, b] = \int_a^b f_X(x)dx$$

4.2.2 Mean and Variance

Given random variables we would like to know about their average. This *mean* or so called expected value for a discrete random variable is calculated by averaging the probability-weighted events:

$$E(X) = \sum_{i=1}^{n} p_i x_i$$

Similar for continuous random variables we get

$$E(X) = \int_{-\infty}^{\infty} x f_X(x)dx$$

The expected value alone doesn't provide that much information about a random variable. We are also interested in the variance of the signal around this mean value because it would give us an idea of how much noise there is in the signal. The *variance* is

$$Var(X) = E[(X - E(X))^2] = E(X^2) - E(X)^2$$

Its square root is known as *standard deviation* $\sigma_X = \sqrt{Var(X)}$.

4.2.3 Gaussian distribution

The *Gaussian distribution* or normal distribution is a special probability distribution which is very popular for modeling random systems [2]. The reason for this is that many random processes occurring in nature actually appear to be normally distributed or at least very close. A normally distributed random process $X \sim \mathcal{N}(\mu, \sigma)$ with mean μ and standard deviation σ has the probability density function

$$f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}}$$

4.2.4 White noise

The mean and variance which were introduced in the previous paragraphs tell us something about the spatial characteristics of a signal, about how much noise there is in the signal. In order to get an idea of the noise rate over time, the spectral characteristics, the term of *autocorrelation* must be introduced. Autocorrelation is the correlation of a signal with itself over time and is given as

$$R_X(\tau) = E[X(t)X(t+\tau)]$$

White noise is an important case of a random signal as its autocorrelation function is

$$R_X(\tau) = \begin{cases} a & \text{if } \tau = 0 \\ 0 & \text{otherwise} \end{cases}$$

Hence white noise is a signal which is completely uncorrelated with itself at any time except the present. That's why white noise signals are also called *independent.* Although it is impossible to achieve this signal in practice it is a very important building block for design and analysis [2].

4.3 Discrete Kalman Filter (DKF)

The first section already introduced the Kalman Filter as an optimal linear estimator implementing prediction and correction steps. Now we have a more detailed look at its underlying models and equations.

4.3.1 Process and Measurement Models

The state $x \in \mathcal{R}^n$ of a discrete time-controlled process shall be estimated. The process model is given by the linear stochastic difference equation

$$x_k = Ax_{k-1} + Bu_k + w_{k-1}$$

and the measurement model is

$$z_k = Hx_k + v_k$$

Hence the next process state x_k is related to the previous state x_{k-1} by an $n \times n$ matrix A. u_k is an optional control input related to x_k by the $n \times l$ matrix B. The $m \times n$ matrix H relates the state x_k to the measurement z_k. w_k and v_k represent the process and measurement noise. They are assumed to be independent and normally distributed:

$$p(w) \sim \mathcal{N}(0, Q)$$

$$p(v) \sim \mathcal{N}(0, R)$$

with process and measurement covariances Q and R.

4.3.2 Origins of the filter

To approach the Kalman Filter and its computational and probabilistic origins we define an *a priori* and an *a posteriori* state estimate: $\hat{x}_k^-$, $\hat{x}_k \in \mathcal{R}^n$. $\hat{x}_k^-$ is estimated with information given prior to step k and $\hat{x}_k$ is the estimate at step k given measurements z_k. Using these estimates we can define *a priori* and *a posteriori* estimate errors e and covariances P:

$$\begin{aligned} e_k^- &= x_k - \hat{x}_k^- \quad & e_k &= x_k - \hat{x}_k \\ P_k^- &= E[e_k^- e_k^{-T}] \quad & P_k &= E[e_k e_k^T] \end{aligned}$$

Computational Origin For the computational origin of the filter we have a look at the "prediction-correction-cycle". For the prediction step we can use the process model on the state estimate of the previous cycle step $\hat{x}_{k-1}$ giving us an *a priori* estimate $\hat{x}_k^-$. Now the equation that computes an *a posteriori* state estimate shall combine the results of the prediction and the information gained from a new measurement z_k.

$$\hat{x}_k = \hat{x}_k^- + K(z_k - H\hat{x}_k^-)$$

The correction is done using a linear combination of the *a priori* estimate and a difference $z_k - H\hat{x}_k^-$ weighted by the so called Kalman gain K. The difference is also called innovation or residual and reflects the discrepancy between the predicted and the actual measurement.

The Kalman Filter was introduced as an optimal estimator. It minimizes the *a posteriori* error. In order to fulfill this criterion the Kalman gain K in the upper equation has to be chosen accordingly. The derivation of K is presented in section 4.3.5 on optimality.

$$K = \frac{P_k^- H^T}{HP_k^- H^T + R}$$

If the measurement noise covariance R approaches 0, K weights the residual more heavily and the measurements are trusted more. Whereas if P, the estimate error covariance, approaches 0 the gain weights the residual less. Hence the measurements are trusted less.

Probabilistic Origin The Kalman Filter minimizes the mean squared error of the estimated parameters under some presumed conditions. One of them is that the noise is assumed to be normally distributed. An important property of the Gaussian distribution is, that it is completely determined by its mean and variance. In the Kalman Filter process exactly these two parameters, the first and second moments of the state distribution, are propagated from time step to time step.

$$E[x_k] = \hat{x}_k$$

$$E[(x_k - \hat{x}_k)(x_k - \hat{x}_k)^T] = P_k$$

The *a posteriori* state estimate $\hat{x}_k$ reflects the mean (the first moment) of the state distribution and the *a posteriori* error covariance P_k represents the variance of the state distribution (the second moment). Hence if the conditions are met and the noises are independent and normally distributed, the state distribution follows

$$p(x_k|z_k) \sim \mathcal{N}(\hat{x}_k, P_k)$$

4.3.3 Discrete Kalman Filter Cycle

The filter estimates a process by using a form of feedback control. The process state at some time is estimated and then a feedback in form of measurements is obtained. This results in the "prediction-correction-cycle".

In the time update step the current state estimate is used to obtain the *a priori* state estimate for the next process step. In addition the *a priori* estimate error covariance is updated.

Time update equations

$$\begin{aligned} \hat{x}_k^- &= A\hat{x}_{k-1} + Bu_k \\ P_k^- &= AP_{k-1}A^T + Q \end{aligned}$$

The measurement update equations are responsible for the feedback. New measurements are used to obtain an improved *a posteriori* estimate.

Measurement update equations

$$\begin{aligned} K_k &= P_k^- H^T (HP_k^- H^T + R)^{-1} \\ \hat{x}_k &= \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \\ P_k &= (1 - K_k H)P_k^- \end{aligned}$$

Here we find the equation which was introduced in section 4.3.2 on the computational origin. First the Kalman gain K_k is computed. This $n \times m$ matrix minimizes the *a posteriori* error covariance equation $P_k = E[e_k e_k^T]$ (see section 4.3.5 on optimality). Then the *a posteriori* state estimate $\hat{x}_k$ is computed as a linear combination of the *a priori* estimate $\hat{x}_k^-$ and the weighted residual $(z_k - H\hat{x}_k^-)$. This difference is also called innovation. Finally the error covariance is updated using the Kalman gain and the *a priori* estimate error covariance P_k^-.

Influence of Q and R Section 4.3.2 already observed some of the influence of the covariances. Now we have a more detailed look at the influence of the process and measurement noise covariances Q and R.

The process noise covariance Q contributes to the overall uncertainty. When Q is large, the Kalman Filter tracks large changes in the data more closely than for smaller Q. R, the measurement noise covariance, determines how much information from the measurement is used. If R is high, the Kalman Filter considers the measurements as not very accurate. For smaller R it will follow the measurements more closely.

Figure 4.2 shows two visualizations of a Kalman Filter proceeding for different covariances Q and R. The red lines represent the measurement data, the green lines are the estimated states. Corresponding to their influence a small covariance Q combined with a large covariance R results in a very smooth curve which follows the measurements only slowly (a). For an opposite constellation the estimate curve sticks close to the measurement curve (b).

4.3.4 Assumptions

Three basic assumptions are made in the Kalman Filter formulation. Are they too restrictive or can they be considered as reasonable?

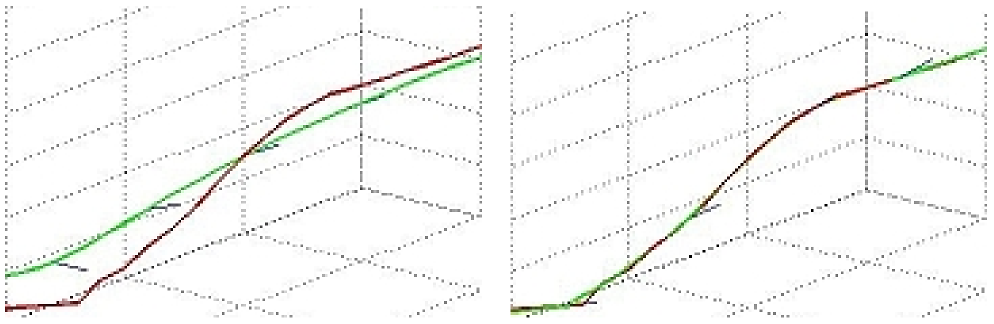

(a) Q small and R large (b) Q large and R small

Figure 4.2: Influence of Q and R

First of all the underlying models are linear. This is justifiable for several reasons because such a simple approach is very often adequate. And in case of non-linearities the typical engineering approach is to linearize about some point. Furthermore linear models are more easily manipulated and the theory is much more complete than for the non-linear case.

The second assumption concerns the probability distribution. Here again some practical reasons apply. The Gaussian distribution is completely determined by its mean and covariance and - as the section on stochastic basics already stated - a lot of random processes in nature are normally distributed or close to it. It can be shown mathematically that the sum of independent random variables with any distribution tends towards a Gaussian distribution.

Finally the noise is considered to be white, hence it is not correlated over time. We already know that white noise cannot really exist but still there are some good reasons for this assumption. Physical systems have a certain frequency bandpass, a range of input frequencies to which they can respond. Furthermore they are typically driven by wideband noise. Now the first motive for a white noise assumption is that within this system bandpass the wideband noise and the white noise look identical. But the real advantage of the whiteness assumption lies in the mathematics which are vastly simplified.

Although the three assumptions are reasonable we'd like to know about the results of the Kalman Filter in situations where they do not hold. If the noise is not Gaussian the Kalman Filter is the best linear estimator given only the mean and standard deviation of the noise. Thus non-linear estimators may be better.

For the case of non-linearity in the process or measurement model the Filter can be modified to linearize about the current mean and covariance. This so called Extended Kalman Filter is discussed in section 4.4.

4.3.5 Optimality

Given the assumptions, the filter is optimal in the sense that it minimizes the estimated error covariance P. This optimality is based on the computation of the Kalman gain K which is then used to update the *a priori* state and error covariance estimates. Hence the goal is to find a Kalman gain that minimizes the *a posteriori* estimate error covariance

$$P_k = E[e_t e_t^T] = E[(x_k - \hat{x}_k)(x_k - \hat{x}_k)^T]$$

The first step to achieve this is to set

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-)$$

$$\text{and } z_k = Hx_k + v_k$$

This results in $P_k = E[s_k s_k^T]$ with

$$s_k = (I - K_k H)(x_k - \hat{x}_k^-) - K_k v_k$$

$(x_k - \hat{x}_k^-) = e_k^-$ is the error of the prior estimate. It is uncorrelated with the measurement noise and therefore the expression can be modified to

$$\begin{aligned} P_k &= (I - K_k H)E[e_k^-(e_k^-)^T](I - K_k H)^T + K_k E[v_k v_k^T]K_k^T \\ &= (I - K_k H)P_k^-(I - K_k H)^T + K_k R_k K_k^T \end{aligned}$$

The diagonal of P_k contains the mean squared errors, hence minimizing the sum of the diagonal - called the trace of the matrix - means minimizing the sum of the mean squared errors. So in order to derive the Kalman gain we take the trace of P_k, differentiate it with respect to K_k, set the result to 0 and solve for K_k.

$$\frac{\partial T[P_k]}{\partial K_k} = -2(HP_k^-)^T + 2K_k(HP_k^- H^T + R) = 0$$

$$K_k = \frac{P_k^- H^T}{HP_k^- H^T + R}$$

4.3.6 Examples

To get a better idea of the functioning of the Kalman Filter some basic examples are presented next.

1D voltage measurement For this 1D case the process and measurement model are very simple.

$$x_k = x_{k-1} \text{ and } z_k = x_k$$

The process and measurement noise are assumed as $Q = 10^{-5}$ and $R = 10^{-2}$. For simulation a random voltage value m is generated and the measurements are calculated through a normal distribution $z \sim \mathcal{N}(m, 0.1)$. Figure 4.3 shows the results of the Kalman Filter process. The red line indicates the voltage measurements and the green line represents the predicted state estimates. As we see, the Kalman Filter allows to smooth the irregular and noisy measurements.

3D position measurement We assume a motion with constant acceleration. Our state vector maintains the positional data together with velocity and acceleration information.

$$\mathbf{x} = (x, y, z, \dot{x}, \dot{y}, \dot{z}, \ddot{x}, \ddot{y}, \ddot{z})^T$$

The process and measurement model are given by the following linear equations:

$$\mathbf{x}(t + \delta t) = A(\delta t)\mathbf{x}(t) + \mathbf{w}$$

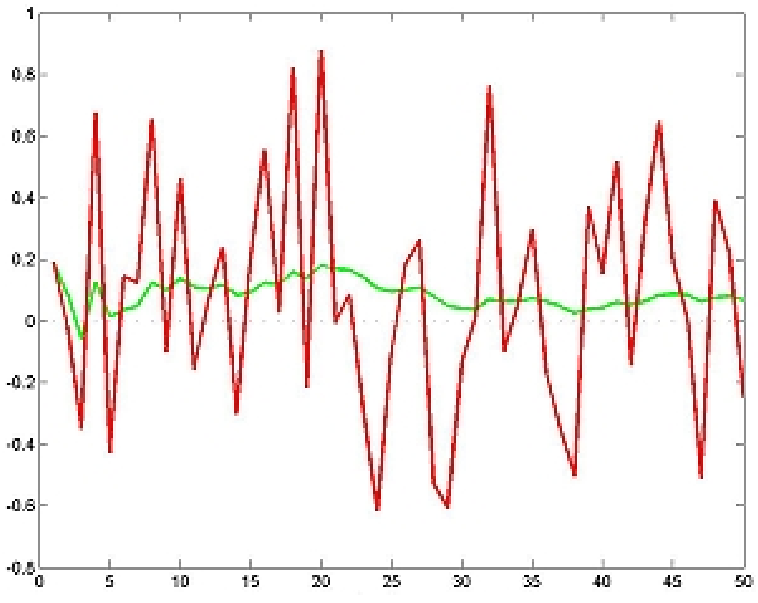


Figure 4.3: 1D voltage measurements

$$A = \begin{pmatrix} 1 & 0 & 0 & \delta t & 0 & 0 & \frac{(\delta t)^2}{2} & 0 & 0 \\ 0 & 1 & 0 & 0 & \delta t & 0 & 0 & \frac{(\delta t)^2}{2} & 0 \\ 0 & 0 & 1 & 0 & 0 & \delta t & 0 & 0 & \frac{(\delta t)^2}{2} \\ 0 & 0 & 0 & 1 & 0 & 0 & \delta t & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & \delta t & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \delta t \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} zx \\ zy \\ zz \end{pmatrix} = \mathbf{z}(t + \delta t) = H\mathbf{x}(t) + \mathbf{v}$$

$$H = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

As the process and measurement noise values are not known, they are assumed to be 0. Again we run the discrete Kalman Filter cycle on a set of measurements, received from a tracker system.

1. compute δt since previous estimate
2. compute state transition matrix $A(\delta t)$
3. do the prediction and correction steps

In the 1D example we took fixed values for the error covariances Q and R. Generally these values are set manually or determined by running a general purpose optimizer offline, over a pre-recorded test data set. The results for two different parameter sets (Q, R) are given in figure 4.2.

4.4 Extended Kalman Filter (EKF)

4.4.1 Non-Linearity

For a lot of interesting applications the linearity assumption for the discrete Kalman Filter does not hold. Still the filter can be applied successfully using an extended version. The Extended Kalman Filter linearizes about the current mean and covariance of the process to be estimated. It does that using the partial derivatives of the process and measurement functions which will be introduced in the next paragraph. Thus it allows us to compute estimates in face of non-linear relationships by modifying the models and update equations.

4.4.2 Process and Measurement Models

The linear models of the discrete Kalman Filter are now replaced by non-linear stochastic difference equations:

$$x_k = f(x_{k-1}, u_{k-1}, w_{k-1})$$
$$z_k = h(x_k, v_k)$$

Instead of using matrices the mapping from the previous state x_{k-1} to the current state x_k is done by the process function f and the measurement function h relates the current state x_k to the measurement z_k.
As we do not know the individual values of the noise w_k, v_k we assume a zero-mean Gaussian distribution and approximate the state and measurement vectors by setting the noise to 0. $\hat{x}_{k-1}$ is some *a posteriori* estimate from a previous time step.

$$\tilde{x}_k = f(\hat{x}_{k-1}, u_{k-1}, 0)$$
$$\tilde{z}_k = h(\tilde{x}_k, 0)$$

4.4.3 Linearization

To estimate this non-linear process its model equations are modified to linearize about the state and measurement vector approximations from above.

$$x_k \approx \tilde{x}_k + A(x_{k-1} - \hat{x}_{k-1}) + Ww_{k-1}$$
$$z_k \approx \tilde{z}_k + H(x_k - \tilde{x}_k) + Vv_k$$

Here x_k and z_k are the actual state and measurement vectors, $\tilde{x}_k$ and $\tilde{z}_k$ are their approximations. The matrices A, H, W, V are Jacobians with partial derivatives of the functions f and h.

$$A_k = \frac{\partial f}{\partial x}(\hat{x}_k, u_k, 0)$$
$$W_k = \frac{\partial f}{\partial w}(\hat{x}_k, u_k, 0)$$
$$H_k = \frac{\partial h}{\partial x}(\hat{x}_k^-, 0)$$
$$V_k = \frac{\partial h}{\partial v}(\hat{x}_k^-, 0)$$

4.4.4 Extended Kalman Filter Cycle

The prediction and correction steps run similar to the discrete cycle. But due to the non-linearity the update equations slightly differ from the linear case.

Time update equations

$$\begin{aligned} \hat{x}_k^- &= f(\hat{x}_{k-1}, \mathbf{u}_k, 0) \\ P_k^- &= A_k P_{k-1} A_k^T + W_k Q_{k-1} W_k^T \end{aligned}$$

The current state x_{k-1} is projected ahead to determine an *a priori* estimate. The process function is used but as the individual noise values are unknown w_{k-1} is set to 0 and an approximation is computed. Then the *a priori* estimate for the error covariance is calculated. The update equation uses the Jacobian matrices A and W.

Measurement update equations

$$\begin{aligned} K_k &= P_k^- H_k^T (H_k P_k^- H_k^T + V_k R_k V_k^T)^{-1} \\ \hat{x}_k &= \hat{x}_k^- + K_k(z_k - h(\hat{x}_k^-, 0)) \\ P_k &= (I - K_k H) P_k^- \end{aligned}$$

For the measurement update the Kalman gain is computed similar to the discrete approach. Because of the non-linearity the measurement Jacobians H and V are applied. Afterwards the state and the error covariance are updated and their *a posteriori* estimates are retrieved. Again the values of the measurement noise v_k are not known and are thus set to 0.

4.4.5 Example

Let's consider again the 3D example of section 4.3.6. If we extend it to rotation tracking an Extended Kalman Filter is needed to face the rotation update.

State and Models The state vector now contains the position data (x, y, z) with its first and second derivatives and the rotation as quaternion $\mathbf{r}$ together with the angular velocity and acceleration.

$$\mathbf{x} = (x, y, z, \dot{x}, \dot{y}, \dot{z}, \ddot{x}, \ddot{y}, \ddot{z}, r_x, r_y, r_z, r_w, \omega_1, \omega_2, \omega_3, \dot{\omega_1}, \dot{\omega_2}, \dot{\omega_3})^T$$

Using a shorter description of the state $\mathbf{x} = (\mathbf{p}^T, \dot{\mathbf{p}}^T, \ddot{\mathbf{p}}^T, \mathbf{r}^T, \omega^T, \dot{\omega}^T)^T$ the process and measurement models are defined as follows:

$$\begin{pmatrix} \mathbf{p}_k \\ \dot{\mathbf{p}}_k \\ \ddot{\mathbf{p}}_k \end{pmatrix} = \begin{pmatrix} I & \Delta t I & \frac{1}{2}(\Delta t)^2 I \\ \mathbf{0} & I & \Delta t I \\ \mathbf{0} & \mathbf{0} & I \end{pmatrix} \begin{pmatrix} \mathbf{p}_{k-1} \\ \dot{\mathbf{p}}_{k-1} \\ \ddot{\mathbf{p}}_{k-1} \end{pmatrix}$$

$$\begin{aligned} \mathbf{r}_k &= \mathbf{r}_{k-1} \otimes d_{k-1} = \\ &= \mathbf{r}_{k-1} \otimes e^{\Delta t \omega_{k-1} + \frac{1}{2}(\Delta t)^2 \dot{\omega}_{k-1}} = \\ &= \mathbf{r}_{k-1} \otimes e^{\mathbf{v}} = (cos(\frac{1}{2}|\mathbf{v}|)\frac{\mathbf{v}}{|\mathbf{v}|}, cos(\frac{1}{2}|\mathbf{v}|)) \end{aligned}$$

$$z_k = h(\mathbf{x}) = \begin{pmatrix} \mathbf{p} \\ normalize(\mathbf{r}) \end{pmatrix} = \begin{pmatrix} \mathbf{p} \\ \frac{\mathbf{r}}{|\mathbf{r}|} \end{pmatrix}$$

The position data is updated linearly as before. But for the rotation the quaternions are updated according to the formula given above. The measurement model requires a normalization of the quaternion entries but the position vector simply can be extracted from the state vector. The quaternion update and the normalization are non-linear and thus are the reason for the application of the Extended Kalman Filter.

Filter Cycle We already learnt the update equations of the EKF. What we need to do is to compute the Jacobian matrices. A and H can be derived in a straightforward way.

$$A_k = \frac{\partial f}{\partial \mathbf{x}} = \begin{pmatrix} A_T & \mathbf{0} \\ \mathbf{0} & A_R \end{pmatrix}$$

Translation and Rotation do not interfere with each other. A_T is the matrix from the linear case above. The Jacobian for the rotation A_R is too large to be displayed here. A detailed representation can be found in [6].

$$H_k = \begin{bmatrix} I & 0 & 0 & ... & 0 & 0 & 0 \\ 0 & 0 & 0 & H_R & 0 & 0 & 0 \end{bmatrix}$$

H_k extracts the position data and normalizes the quaternion from the state vector through the matrix H_R. Again we refer to [6] for a complete representation of H_R.

4.5 Discussion of the Kalman Filter

The Kalman Filter faces the question "How do we get accurate information out of inaccurate data?". Through its prediction and correction cycle it is a robust and stable estimator. It copes with missing or uncertain data and is able to fuse information from multiple sensors. In the previous sections we introduced the Discrete and the Extended Kalman Filter.

The DKF is applicable to many system processes with a linear process model. For these types of systems it is an optimal estimator if the noise can be assumed white and Gaussian. And here we find the disadvantage of the Discrete Filter. It is only optimal if the three conditions of linearity and white and Gaussian noise hold. In any other case it can be used as an estimator but non-linear approaches may be better.

The EKF tries to fill this gap by providing a filter for non-linear models. It linearizes about the current mean and covariance of the process to be estimated. But nevertheless it cannot take account of the problem of non-Gaussian models. There the Extended Kalman Filter tends to be unreliable [4].

For both filter their usage requires the construction of a process and a measurement model. Furthermore it can be costly and difficult to determine an adequate value for the process noise covariance. The measurement noise covariance can be computed by running a general purpose optimizer over a pre-recorded data set.

4.6 Sensor Fusion with the Kalman Filter

In the previous section we introduced the Kalman Filter, its models and equations. Now we discuss its usability for sensor fusion and present some benefitting extensions.

4.6.1 DKF and EKF

What we have is a tool providing us with information as accurate as possible, given uncertain data. Now we want to apply the Kalman Filter for sensor fusion.

One Filter The easiest way to combine data from multiple sensors is to sum them up in one filter. This filter takes the measurements from all the sensors as they come. It then updates the state estimate when enough information is gathered. This is method is applied for the UNC hybrid landmark-magnetic presented at SIGGRAPH 96 [7]. This system uses a magnetic tracking system along with a vision-based landmark recognition system. Thereby it achieves superior image registration for augmented reality assisted medical procedures. When multiple landmarks are identified in a single image, the correction step is applied to the magnetic readings.

Separate Filters A different approach would be to use separate filters for each sensor. That way they can be optimally adjusted for the special cases. In our example in section 4.4.5 we used an Extended Kalman Filter as the quaternion update requires a non-linear model. Still for the position update we used the linear approach. So in splitting position and orientation data we could use separate filters: a DKF for the position estimation and an EKF for the rotation estimation. This approach is taken by Azuma in his thesis [1]. He wants to predict future head locations in order to reduce dynamic registration errors. For the estimation process he applies a Discrete Kalman Filter for translation and an Extended Filter for the orientation components. The translation parts are additionally broken up into the three axes. This separation greatly simplifies the filters at the cost of ignoring potential correlations across the three axes. The linear translation update was already presented in the example above so we will only present the orientation update. Azuma uses a model which holds rotation as quaternions. They are updated by the formula $q_k = \frac{1}{2}q_{k-1}\omega$. For the measurement model the quaternion is normalized as in 4.4.5. Again the filter cycle requires the computation of Jacobians because of the non-linearity.

4.6.2 SCAAT

Introduction The systems which were presented in the previous section combine multiple sequential measurements for one single update step. It might be too costly to try a different non-sequential way. Or they need a certain number of individual observations - which can only be obtained sequentially - in order to have sufficient state information for estimation. The big problem resulting from this approach is the so called *simultaneity assumption*. Several measurements are necessary for the computation. They are taken sequentially but they are assumed to be collected simultaneously. For a moving target, even with slow motion, this can cause severe errors.

Consider the landmark-tracker introduced in the previous section. Its cameras provide images with multiple landmarks, but only one of them is processable at a time. Several landmarks must be located in order to allow an update step. Besides the simultaneity error another problem arises. What if the number of visible landmarks is smaller than the number needed by the system to do an estimation? The system is no longer able to work.

SCAAT idea The SCAAT (single-constraint-at-a-time) approach faces this problem of assumed simultaneity. It allows to combine information from different sensors in a flexible way. A system that uses multiple constraints which are individually incomplete must incorporate a sufficient set of these incomplete measurements in order to become observable and allow estimation. The idea behind SCAAT is that this global observability is obtained *over time.* Each measurement provides some information about the current state and can therefore help to incrementally improve the previous estimates.

Single measurements are isolated from each sensor and directly included in the estimation process. That way the system uses more recent data and works faster. Individual errors are more easily identified and dealt with. No simultaneity assumption must be made and hence no motion restriction is needed to avoid errors. And for the UNC landmark-tracker example we solve the problem of landmark visibility. In each image only one landmark is processed and the state estimate for this landmark is updated. After that the image is discarded and a new one is used for the next processing step.

Tracking Algorithm In order to discuss the SCAAT algorithm in detail we use the example presented in the SCAAT paper [8]. There they have a position-velocity model with six parameters for position, six for orientation and constant velocity.

$$\mathbf{x} = (x, y, z, \dot{x}, \dot{y}, \dot{z}, \psi, \theta, \phi, \dot{\psi}, \dot{\theta}, \dot{\phi})^T$$

The state transition matrix $A(\delta t)$ projects the state forward in time and implements the following relationships:

$$\mathbf{x}(t + \delta t) = \mathbf{x}(t) + \dot{\mathbf{x}}(t)\delta t$$
$$\dot{\mathbf{x}}(t + \delta t) = \dot{\mathbf{x}}(t)$$

The process noise $\mathbf{w}(\delta t)$ is normally distributed with covariance $Q(\delta t)$.

For each sensor type σ we have a measurement vector $\mathbf{z}_\sigma(t)$ and a corresponding measurement function $\mathbf{h}_\sigma$.

$$\mathbf{z}_\sigma(t) = \mathbf{h}_\sigma(\mathbf{x}(t), \mathbf{b}_t, \mathbf{c}_t) + \mathbf{v}_\sigma(t)$$

$\mathbf{b}_t$ and $\mathbf{c}_t$ are the tracker source and tracker sensor parameters. For an ideal SCAAT application only a single sensor and source pair should be observed for each SCAAT Kalman Filter measurement update ($|\mathbf{z}_\sigma| = 1$). A reasonable usage for this example is $|\mathbf{z}_\sigma| = 3$ for a separate handling of the position and the orientation data.

The measurement noise follows a Gaussian distribution, too. Its covariance matrix is $R_\sigma(t)$. As the measurement model is non-linear we need to determine the corresponding Jacobian matrix

$$H_\sigma(\mathbf{x}(t), \mathbf{b}_t, \mathbf{c}_t)[i, j] = \frac{\partial}{\partial \mathbf{x}(j)} \mathbf{h}_\sigma(\mathbf{x}(t), \mathbf{b}_t, \mathbf{c}_t)[i]$$

Algorithm The initial state estimate and error covariance estimate are $\hat{\mathbf{x}}(0)$ and $P(0)$. Whenever a discrete measurement $\mathbf{z}_\sigma(t)$ is available:

1. compute δt since previous estimate

2. predict state and error covariance

$$\hat{\mathbf{x}}^- = A(\delta t)\hat{\mathbf{x}}(t - \delta t)$$
$$P^- = A(\delta t)P(t - \delta t)A^T(\delta t) + Q(\delta t)$$

3. predict measurement and compute corresponding Jacobian

$$\hat{\mathbf{z}} = \mathbf{h}_\sigma(\hat{\mathbf{x}}^-, \mathbf{b}_t, \mathbf{c}_t)$$

$$H = H_\sigma(\hat{\mathbf{x}}^-, \mathbf{b}_t, \mathbf{c}_t)$$

4. compute Kalman gain

$$K = P^- H^T (H P^- H^T + R_\sigma(t))^{-1}$$

5. correct state estimate and error covariance

$$\hat{\mathbf{x}}(t) = \hat{\mathbf{x}}^- + K(\mathbf{z}_\sigma(t) - \hat{\mathbf{z}})$$

$$P(t) = (I - KH)P^-$$

Discussion The SCAAT method integrates individual measurements - which alone are incomplete constraints - into the complete state estimate. This is accomplished through the Kalman gain. K minimizes the error covariance if the conditions are met. It reflects the relative uncertainties of the state and the measurement. K is influenced by the Jacobian H which reflects the rate of change of each measurement with respect to the current state. K is recomputed in each step and thus it indicates the amount and direction of information in state space provided by the individual constraint.

SCAAT has been implemented for the UNC Hiball tracker, a wide-area optoelectronic tracking system [8]. Simulation results show that the method leads to smoother curves and higher update rates compared to a conventional approach. There the estimated curves appear very jerky, mostly because of the low estimate rate. Furthermore SCAAT reduces the root mean square error.

All in all SCAAT is a stable method which provides estimation with individual measurements. It hereby avoids the simultaneity assumption and is faster and more accurate because it uses more recent data. Hence it allows better prediction and eases the discrimination of bad measurements.

4.6.3 Federated Kalman Filter (FKF)

Introduction Multisensor systems which apply classical techniques can get severe problems with computational load. In addition simplifications are not always reliable and can lead to poor accuracy, instability or divergence [3].

The idea of using decentralized filtering is a first step to face this difficulties. Independent local state estimates are constructed which can then be combined in a straightforward way by a master filter. But what we need to take into account is the master filter rate. We would like to do some sort of pre-filtering at the lower level in order to achieve a rate reduction at the master filter level.

Federated Kalman Filter The Federated Kalman Filter allows this local data compression and presents estimation with globally optimal or suboptimal accuracy as a function of the selectable master filter rate. Furthermore it offers a high degree of fault tolerance. Its general structure is shown in figure 4.4. Each local filter is dedicated to a separate sensor subsystem. The outputs of these local filters are subsequently processed and combined by the larger master filter.

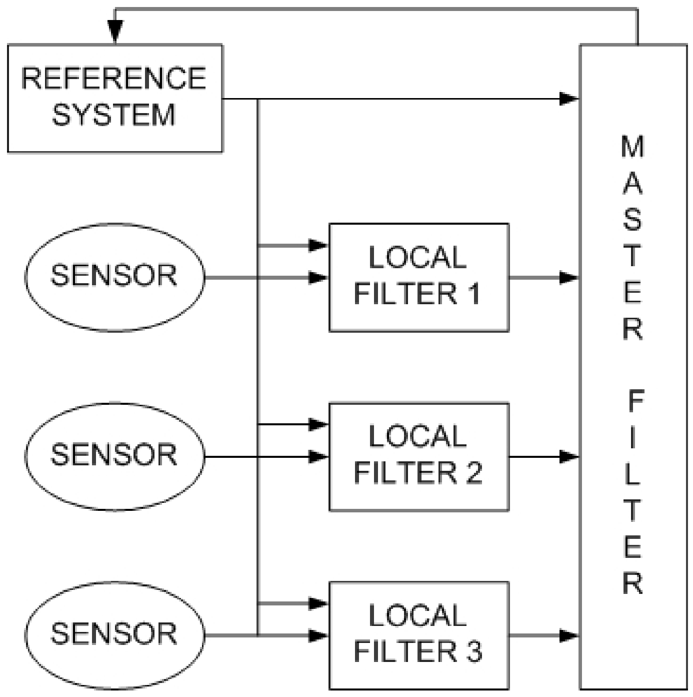


Figure 4.4: Structure of the FKF

Filter Structure Now let's have a look at the underlying models and equations.
The process and measurement models are the following:

$$\mathbf{x}_k = A\mathbf{x}_{k-1} + G\mathbf{w}$$

The states a related over time via the state transition matrix A. G is the process noise distribution matrix and $\mathbf{w}$ represents white normal process noise with its corresponding process noise covariance Q. The initial state $\hat{\mathbf{x}}^0$ and the noise values $\mathbf{w}(t)$ are uncorrelated.

$$\hat{\mathbf{z}}_i = H_i\mathbf{x} + \mathbf{v}_i$$

The external measurements from the different sensors $i = 1..N$ are independent and are related to the true state via the measurement observation matrices H_i. $\mathbf{v}_i$ is the measurement noise, assumed to be white and Gaussian, too. Its covariance matrix is R_i.
First we define the composite global filter, its state and error covariance.

$$\mathbf{x} = [\mathbf{x}_1 ... \mathbf{x}_N]^T \quad \mathbf{x}_i = \begin{bmatrix} \mathbf{x}_{c_i} \\ \mathbf{x}_{b_i} \end{bmatrix}$$

$$P = \begin{bmatrix} P_{11} & ... & P_{1N} \\ ... & ... & ... \\ P_{N1} & ... & P_{NN} \end{bmatrix}$$

The full state vector is the composite of all N local state partitions. Each of them contains the common system states $\mathbf{x}_{c_i}$ and the sensor bias states $\mathbf{x}_{b_i}$. The composite covariance P can contain cross-partitions P_{ji} as well as local partitions P_{ii}.

Now given a set of N local state estimates $\hat{\mathbf{x}}_i$ and their covariances P the global optimal state estimate of the full system $\mathbf{x}$ minimizes the weighted least squares cost index $\psi = \mathbf{e}^T P^{-1} \mathbf{e}$ with $e = (\hat{\mathbf{x}} - \mathbf{x})$. If the cross partitions P_{ji} are zero, the expression is reduced to

$$\psi = \sum_{i=1}^{N} ||(\hat{\mathbf{x}}_i - \mathbf{x}_i)||^2_{Pii^{-1}}$$

In this case the globally optimal solutions for the state and the error covariance are

$$\hat{\mathbf{x}}_m = P_{mm}[P_{11}^{-1}\hat{\mathbf{x}}_1 + ... + P_{NN}^{-1}\hat{\mathbf{x}}_N]$$

$$P_{mm} = [P_{11}^{-1} + ... + P_{NN}^{-1}]^{-1}$$

These equations represent the optimal combination of the local filter results when the local estimates are uncorrelated.

In order to prove this noncorrelation we have a look at the composite global variables and the global time propagation step.

1. Processing of local measurement from sensor i

$$\hat{\mathbf{z}}_i = H\mathbf{x} + \mathbf{v}_i; \quad H = [0...H_i...0]$$

$$K = P_{ji}H_i^T(HPH^T + R)^{-1} = P_{ji}H_i^T(H_iP_{ii}H_i^T + R_i)^{-1}$$

$$\hat{\mathbf{x}}_j = \hat{\mathbf{x}}_j^- + K(\hat{\mathbf{z}}_i = H_i\hat{\mathbf{x}}_i)$$

$$P_{jk} = P_{jk}^- - KH_iP_{ki}^T$$

2. Global time propagation step

$$\begin{bmatrix} \mathbf{x}_1 \\ \cdot \\ \cdot \\ \mathbf{x}_N \end{bmatrix}_k = diag[A_{ii}] \begin{bmatrix} \mathbf{x}_1 \\ \cdot \\ \cdot \\ \mathbf{x}_N \end{bmatrix}_{k-1} + \begin{bmatrix} G_1 \\ \cdot \\ \cdot \\ G_N \end{bmatrix} \mathbf{w}$$

$$\begin{bmatrix} P_{11} & ... & P_{1N} \\ ... & ... & ... \\ P_{N1} & ... & P_{NN} \end{bmatrix}_k = diag[A_{ii}] \begin{bmatrix} P_{11} & ... & P_{1N} \\ ... & ... & ... \\ P_{N1} & ... & P_{NN} \end{bmatrix}_{k-1} diag[A_{ii}^T] + \begin{bmatrix} G_1 \\ \cdot \\ \cdot \\ G_N \end{bmatrix} \mathbf{w}[G_1^T...G_N^T]$$

$$[P_{ji}]_k = A_{jj}[P_{ji}]_{k-1}A_{ii}^T + G_iQG_i^T$$

In (1) equations with $j = i$ indicate that sensor i measurements affect the state i and its covariance as if only i existed. i measurements do only affect other local states $i \neq j$ if the cross-covariances P_{ij} are nonzero. Thus if the local filter states are initially uncorrelated then the local measurement sets can be processed independently, they remain uncorrelated.

With (2) we see that the common process noise $\mathbf{w}$ cross-correlates the separate local filter estimates. In doing some modifications and in introducing an upper bound through the factor γ_i we can overcome this correlation.

$$[P_{ii}]_k = A_{ii}[P_{ii}]_{k-1}A_{ii}^T + G_i\gamma_iQG_i^T$$

$$[P_{ji}]_k = A_{jj}[P_{ji}]_{k-1}A_{ii}^T = 0 \text{ if } [P_{ji}]_{k-1} = 0$$

The same upper-bounding approach can be applied to the state covariance matrix. The result are that A and the local measurements introduce no cross-correlation. The initial covariance matrix and process noise covariance bounds are uncorrelated by construction. Hence the local filter estimates can be combined via the simple equations for $\hat{\mathbf{x}}_m$ and P_{mm} given above.

Algorithm

1. set initial local covariances to $\gamma_i\times$ common system value
2. local filters process own measurements via locally optimal KF
3. master filter combines local filter solutions after each update cycle via the equations

$$\hat{\mathbf{x}}_m = P_{mm}[P_{11}^{-1}\hat{\mathbf{x}}_1 + ... + P_{NN}^{-1}\hat{\mathbf{x}}_N]$$

$$P_{mm} = [P_{11}^{-1} + ... + P_{NN}^{-1}]^{-1}$$

4. master filter resets local filter states to master value and local covariances to $\gamma_i\times$ master value

Discussion The Federated Kalman Filter offers a highly fault-tolerant, rate-reduced decentralized filtering approach for sensor fusion. It can be applied to systems with fixed local filter design for stand-alone operations as well as to very flexible local designs which allow best support for federated filtering operations. The filter provides globally optimal or suboptimal estimation accuracy depending on the selected master filter rate.

4.7 Conclusion

The fist part of this paper concentrated on the basic concepts of the Kalman Filter. We presented the models and equations of the Discrete Kalman Filter, an optimal estimator for linear systems assuming white Gaussian noise distribution. For the non-linear case the Extended Kalman Filter was introduced. This filter linearizes about the mean and covariance of the current estimate and presumes independent and normally distributed noise, too. Several examples have been described to ease the understanding of the presented ideas.

In the second part multiple approaches for sensor fusion are shown. An easy way is to use the DKF or the EKF directly for the sensor measurements. The information is taken and - when sufficient data is collected - used for a new update cycle. Depending on the complexity of the system and the models it can be useful to split the work and use several filters for different sets of data. This approach creates a simpler structure but in using separate and independent filters possible correlations between the data sets are ignored.

In addition to these general possibilities two special extensions of the Kalman Filter are introduced. The SCAAT method allows incremental improvement of previous estimates by integrating single measurements (or constraints), ideally one at a time. That way an update is possible even when the given data is not sufficient for a complete state estimate. Furthermore SCAAT enables more accurate estimation as more recent data is used for update. This is especially helpful for motion tracking where several landmarks would have be processed for a full update which at the same time would causes prediction errors because of the simultaneity assumption.

The last extension which was presented is the so called federated Kalman Filter. It uses a decentralized filtering system to achieve higher performance and tries at the same time to reduce the master filter rate by the invention of a pre-filtering technique. It supports a variety of system requirements and facilitates performance trade.

Bibliography

[1] R. Azuma, *Predictive Tracking for Augmented Reality*, PhD thesis, University of North Carolina at Chapel Hill, 1995.

[2] G. Bishop and G. Welch, *An Introduction to the Kalman Filter.* SIGGRAPH 2001, Course Notes.

[3] N. A. Carlson, *Federated Square Root Filter for Decentralized Parallel Processing*, in IEEE Transactions on Aerospace and Electronic Systems, 1990.

[4] D. A. Forsyth and J. Ponce, *Tracking with Nonlinear Models.* Chapter that did not make it into the book "Computer Vision - A Modern Approach".

[5] P. S. Maybeck, *Stochastic Models, estimation and control, Vol. 1*, in Academic Press New York, 1979.

[6] D. Pustka, *Handling Errors in Ubiquitous Tracking Setups*, Master's thesis, Technical University Munich, Department of Computer Science, 2003.

[7] A. State, G. Hirota, D. T. Chen, B. Garrett, and M. Livingston, *Superior Augmented Reality Registration by Integrating Landmark Tracking and Magnetic Tracking*, in ACM SIGGRAPH, Addison Wesley, 1996.

[8] G. Welch and G. Bishop, *SCAAT: Incremental Tracking with Incomplete Information*, in SIGGRAPH, 1997.

5 Adaptive transform of the color space in image compression

Kirill Yourkov, State Univ. of Aerospace Instrumentation St. Petersburg

BMP (bit map) is one of the main formats in color image representation. The standard representation of pixels in this format is RGB, it uses 24 bits for one pixel. Every color component (Red, Green, Blue) is represented by one byte. Since components in this representation are correlated, their independent compression is redundant. The linear transform is usually used at the first stage of image and video compression. Mostly this is the RGB-to-YUV nonsingular linear transform with fixed coefficients. This transform decreases correlation between color components and improve energy localization. In general this transform is not optimal since its coefficients are fixed. So, one can choose coefficients of the linear transform for a given image on the base of some optimality criterion. Information on the applied transform depends on the image and must be transmitted to the decoder. The author introduces some optimality criterion. The optimal transform coefficients are found according to this criterion and analyzed. Then the algorithm for efficient transmission of information about the transform is proposed and estimated.

6 Decimation of color-difference components by wavelet filtering

Alexander Chuikov, State Univ. of Aerospace Instrumentation St. Petersburg

Usually all pixels in color images are presented by the color space transform, e.g., RGB-to-YUV. After such transform the color-difference components U,V are redundant and decimation of them is used for decreasing the redundancy. Standard decimation uses 1:4:4 format and decreases the number of U,V- components in 4 times. Such decimation may be considered as a result of low-pass filtering by the Haar wavelet filter. We generalize such approach considering the wavelet filtering with more complex filters. As the example we consider the application of the low-pass 3/5-wavelet filtering of color-difference components providing the wavelet decimation. Such filtering improves compression without visible destruction of images. We present the investigation results in this area.

7 Real-Time: The Zerberus System

Christian Buckl, Technische Universität München

In this paper a development model for the implementation of augmented reality application is suggested. The Zerberus System, originally designed for the development of safety-critical real-time applications, provides a tool chain and offers amongst other things an automatic code generation. By its platform independancy, by the use of commercial-of-the-shelf hardware and by the acceleration of the development process the costs for system design and implementation can be reduced.
This paper makes suggestions how the Zerberus System can be used in the context of augmented reality.

7.1 Introduction

In this paper a development model for the implementation of augmented reality is suggested. The need for development models can be derived from the different requirements imposed by the application scenarios.
Since several requirements are repeatedly demanded in different applications, an automatic or tool-aided realisation of mechanisms to satisfy these requirements can be very useful. This is especially true since for most of the requirements experts of domains different from the application domain are needed.
This paper is structured as follows: in section 7.2 three typical applications of augmented reality are described and the requirements on the system solutions are discussed. Section 7.3 introduces the Zerberus System, a development system with a complete tool-chain. A special focus is set on the Zerberus Language that is used for the specification of the functional design of the application. But also the other steps are described and it is shown that the Zerberus System can satisfy the requirements worked out in section 7.2.
Section 7.4 describes modifications that could be useful in the context of augmented reality development.
Finally section 7.5 summarizes this paper.

7.2 Requirement elicitation

In this section typical requirements imposed by the application scenarios are worked out. Therefore three different applications are discussed: a medical assistence application for minimal-invasive surgery, a robot remote control and a maintenance support application.

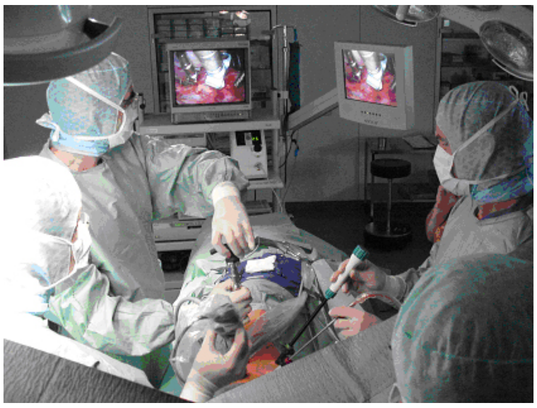


Figure 7.1: Medical assistence

7.2.1 Example applications

Medical assistence The first example that will be discussed is a medical assistence application. Since minimal-invasive surgery is increasingly applied to improve the surgery results and minimize the risk of complications, this is a typical application for augmented reality. Figure 7.1 shows one possible scenario. Via video, magnetic resonance imaging, ultrasound or computed tomography scans the instruments position can be determined and the system can provide useful information to the surgeon.
In the application description the first requirement can be easily detected. Augmented reality applications must often handle different sensor devices. These devices are often unreliable and erroneous. For this reason different sensors are frequently used and different algorithms have to be executed in parallel. Since these algorithms are very complex and time consuming in some cases it may be necessary to use a distributed system to satisfy the temporal requirements.
Another requirement of this application is of course safety: the system must in no case produce an erroneous result. This requirement is especially hard to fulfill due to the addressed sensor problems.
A last obvious requirement is the real-time ability of the system. It is not acceptable for the surgeon to deal with long delays of the computer system.

Robot remote control The second example deals with a remote robot control application. A possible application scenario is described in [1]. Due to long delays in the communication link it can be preferable to control a virtual version of the robot instead the real robot. The user plans and specifies the robot's actions by manipulating the local virtual version in real time, see figure 7.2. Once the plan is tested amd determined the user tells the real robot to execute the specified plan.
This application introduces a new requirement: since the robot may be in a place where maintenance or repair is very difficult (for example in case the robot operates in a environment dangerous for human) or even impossible (e.g. in space) the application must be very reliable. Of course also the safety requirement is valid.

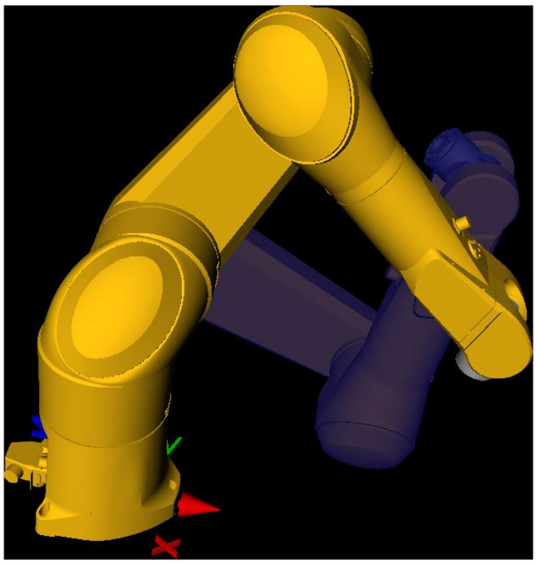


Figure 7.2: Remote control

Maintenance support Another established application domain are solutions for maintenance assisting systems. One possible form of such solutions is shown in picture 7.3. The user wears a head-mounted display. On the display the system can show instructions and mark objects. The user can communicate with the system by voice or buttons.
This system also has strict requirements to the real-time ability. A small delay can lead to a divergence between the illustrated videos and the reality. Therefore strict real-time requirements have to be met.

7.2.2 Requirements summary

As seen in the discussed applications different non-functional reqirements have to be fulfilled by augmented reality solutions. These requirements are:

Figure 7.3: Maintenance application

1. Safety
2. Reliability
3. Real-time ability
4. Ability to cope with different erroneous sensors
5. Ability for distributed computing

The last item leads to another problem. If several processors are used, the system must guarantee a correct and sufficient synchronization algorithm.
Another demand is focussed on the performance of the system. Since many algorithms are very complex and time-consuming it might be preferable to change the hardware as soon as new, more powerful hardware is available. This leads to the requirement that the system have to be designed in a way that a platform tranformation can be realized with minimal effort.

7.2.3 Requests on a development system

Due to the specified requirements there are different demands on a development model. The most important demand is the cost-efficiency of the approach. The development costs should be reduced by using such a development model. To allow a fast familiarisation with the development model the system has to be simple and intuitive. This way the error rate can also be reduced.
As many requirements are repeatedly needed for AR applications, the development system should also assist the developer in designing appropriate mechanisms to fulfill these requirements. In an optimal case the realization of these mechanisms could be automatic.
Another important aspect is the safety. An increasing number of applications can be classified as safety critical. Therefore the development process should provide an inherent safety.
Finally the system should allow an application domain expert to develop the system without needing assistence by other experts.

7.3 Zerberus system

In this section the Zerberus System, a development system with a complete tool-chain, is introduced. In the first paragraph the original background is illustrated. Afterwards the process model in described. A focus is thereby set on the Zerberus language that allows the platform independent specification of the functional model of the application.

7.3.1 Background

The Zerberus System was originally designed for the development of safety critical real-time applications. Due to this reason the focus of the system is set on fault-tolerance mechanisms. These mechanisms are achieved by active structural redundancy as depicted in figure 7.4. At least three equivalent units process the application in parallel. Voting, sychronization and integration protocols are provided to achieve a safe and reliable execution of the applications. Besides the pure structural redundancy diversity in hardware and software is supported by the system. To reduce the costs of the developed applications the system is based on the use of commercial-of-the-shelf hardware.

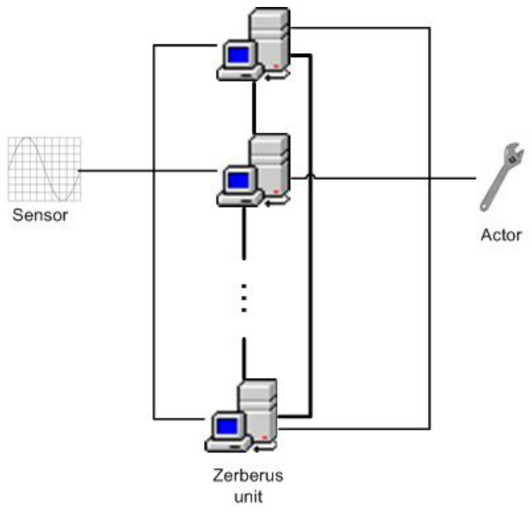


Figure 7.4: Hardware structure

Highlights of the system are the Zerberus language that allows the platform independent specification of the functional model with all temporal constraints and the automatic code generation.

7.3.2 Development process

The Zerberus System suggests different steps in the development process for dependable systems. In each step the system assists the developer to accelerate the process and to improve the results by tool support or by providing guidelines. The individual steps to produce executable code are illustrated in fig. 7.5 and are described below.

Step 1: Specification of the Functional Model Within this step the user has to specify the functional objects of the application, their relationship towards each other and the temporal constraints. The specification is realized by the use of the Zerberus Language. Since the specification of the functional model should be independent of a specific platform the Zerberus Language has to be designed in a way to support this independency. A platform in the context of the Zerberus System is understood as the hardware, the operating system and the programming language.

To avoid an error source and a long-lasting learning process the Zerberus Language has to be intuitive. For complying with the generality of the Zerberus approach the language should as well do not pose too many constraints on the applications.

The Zerberus language is described more detailed below.

7.3.3 Analysis of the Requirements on the Dependability

Currently the Zerberus System offers active structural redundancy as fault-tolerance mechanism. At least three Zerberus units compute the application in parallel. At specified points in time the units perform a distributed voting and synchronization algorithm. Erroneous

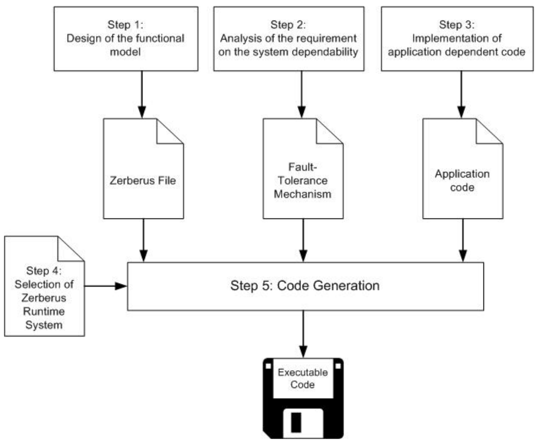

Step 1: Design of the functional model
Step 2: Analysis of the requirement on the system dependability
Step 3: Implementation of application dependent code
Zerberus File
Fault-Tolerance Mechanism
Application code
Step 4: Selection of Zerberus Runtime System
Step 5: Code Generation
Executable Code

Figure 7.5: Development process

units are excluded from the computation and can perform error detection algorithms. After a successful completion the system allows the reintegration into the running system.

Since a replication of identical units allows no toleration of design errors, the system also supports diversity of hardware and software. While hardware diversity leads to no or only few extra costs because of the support of COTS hardware, N-Version programming is often not considered due to the extra costs for the development of the individual versions.

As a result of these considerations several requirements are posed to the Zerberus Language. First of all the language must support the replica determinism: during the voting the systems must be in a state that allows a comparison of these states and an error detection. Especially due to the support of N-Version programming this is a non-trivial requirement.

Another requirement that arises from the voting is the existence of deterministic points in time, when the voting should be performed.

To support the re-integration of a previously excluded Zerberus unit, the system must offer facilities for state synchronization. Since the algorithms are realized automatically by the system a derivation of the state of the individual units must be possible out of functional model.

Finally to achieve a reduction of the implementation effort for N-Version programming the code that has to be implemented by the developer should be restricted to the pure application dependant code.

7.3.4 Implementation of Application Dependent Code

In this step the developer has to implement code for the application. As already implied in the previous section this code is restricted to the pure application dependant functionality of the main parts which were identified within the design process of the formal model. By this restriction the implementation effort can be minimized to the absolute minimum.

The implementation step is platform dependant. This implies that for every used platform the code has to be reimplemented by the developer.

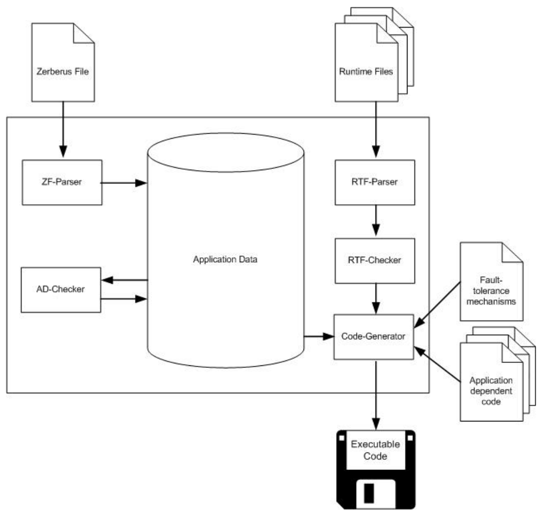


Figure 7.6: Code generation process

7.3.5 Code Generation

The transformation of the functional model, the application dependant code and the selected fault-tolerance mechanisms into an executable code is performed automatically by the Zerberus code generator. Since the code generator is designed platform independent another mean has to be used in addition to achieve the platform mapping. These means are the Zerberus run-time systems. Zerberus run-time systems are platform dependent and realize the execution on the specific platform. To allow the reuse of Zerberus run-time systems another language, the Zerberus tags, are introduced. By the use of Zerberus tags locations in the run-time code can be marked that need to be replaced by application dependent data. The Zerberus code generator parses the runtime files and realizes the replacement by using the information contained in the functional model of the application.

In addition the Zerberus code generator performs syntactical and semantical checks on the formal model and on the runtime system files. The whole code generation process is depicted in fig. 7.6.

7.3.6 Zerberus language

In the last preceding paragraph the requirements on the Zerberus language were discussed in the context of the different development process steps. In this section the Zerberus language is described informal and it is shown that the requirements can be satisfied by the Zerberus language. The language was influenced by the language Giotto introduced in Berkeley [4]. Giotto is a language to describe the functional model of distributed control application independent of the platform. Giotto was changed and extended in a way that the Zerberus language was suited for the use of fault-tolerance mechanisms.

The main attribute to support voting, synchronization and integration algorithms is replica determinism. This is a non-trivial issue since different platforms can be used to achieve fault-tolerance. This includes the simultaneous use of different hardware, operating systems, programming languages and control algorithms in one control system. To achieve nevertheless replica determinism the Zerberus language is based upon the time-triggered paradigm [5].

Similar to the approach in [6] replica determinism can be achieved by using the knowledge about the execution times. In the context of control applications the execution times can be related to the frequency of control cycles.

Basing the voting, synchronization and integration algorithms on the frequency of control cycles has different positive outcomes: by specified frequencies of control cycles in the functional model there exist on the one hand deterministic points in time, when the synchronization and voting algorithms can take place, and on the other hand the execution and scheduling of the different processes can be carried out in different ways on the Zerberus units between these points.

The realization of the voting, sychronisation and integrations algorithms by the run-time systems guarantees in addition the consistency of the algorithm executions, since the execution of the voting algorithms can be made independent from the execution order of the different control processes.

To achieve the claimed simplicity of the language, the Zerberus language consists of only seven different object types: ports, actors, sensors, guards, modes and modechanges. In this section the different object types are explained informally.

Port All communication in the Zerberus System is performed via ports. A port is a unique space in memory with a predetermined size and a specified representation. The available port types are platform independent, but are based on the fundamental types of the most common programming languages.

The values of the ports represent the state of the Zerberus units. Therefore a comparison of the different Zerberus units can be based on the values of these ports. It is required that there are no spaces in memory to store internal states besides the ports. Thus the state synchronization can be based on the values of the ports during the reintegration of a Zerberus unit. The platform independent specification of the size and the representation of the port values is the foundation to allow also N-Version programming using different programming languages and operating systems.

In the following the attributes of ports are described. Ports are persistent, that means the ports keep their values over time until the port is updated. The update access has to be performed deterministic. It is not allowed that more than one write access is performed at a certain point in time. This condition is checked by the code generator while parsing the formal model and in addition at run-time (due to guards,see sec.7.3.6).

Replica indeterminism can also have reasons in the small clock differences (since the synchronization algorithm can only guarantee a deviation of the local clock from the global clock smaller than ε) or in the different implementations of the different versions. Due to these facts the correct port values are normally situated in a small interval. To support this fact the comparison of ports can also be based on an interval decision. This can be done by declaring a voting function for the port that has to be implemented by the developer. In case no voting function is specified the voting of the port values are based on the bit-by-bit comparison.

The voting on the value of a specific port takes place at least every time an output is performed based on this port value. For a faster detection of errors the developer can also specify shorter voting intervals.

Task The separation of the pure functionality of the application and the run-time system with the fault-tolerance mechanisms is realized by tasks. Tasks are periodically called functions and realize the actual control system functionality. The simultaneous execution of different tasks is allowed but to achieve determinism in the execution the tasks have to be independent of each other and synchronization points are forbidden. Thus the implementation of the task functions is simplified and accelerated since they represent only sequential programs and the requirement of the strict partitioning of the integrated modules to reduce the certification effort is satisfied.

The communication of the tasks between each other and with the environment is exclusively performed via ports. The access of tasks on ports occurs in a time-triggered manner. At the begin of every invocation the task reads the values of the input ports, at the end of the invocation the results are written into the output ports of the task. The begin and the end refers here to the invocation period as specified in the functional model. The port access is realized by the Zerberus run-time system and is performed in logical zero time.

The actual execution of the task on the CPU is scheduled by the Zerberus run-time system and is transparent to the developer. Nevertheless the developer has to guarantee that the worst-case execution times (WCETs) of the tasks allow a completion of the tasks satisfying the temporal restrictions as specified in the functional model.

Sensor and Actor Sensors and actors realize the communication of the application with the environment and should not be mistaken for the hardware devices. Sensors are functions that are executed to read values from the environment and to write these values into ports, actors are functions to read values from the port and write these values to the environment.

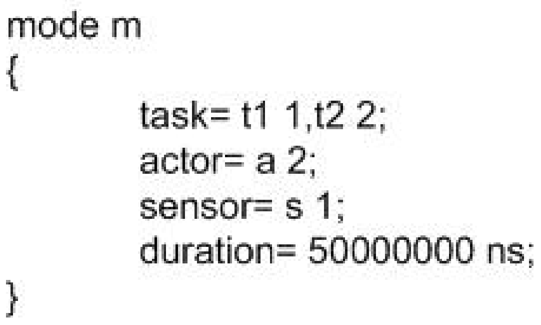
The execution of the sensors and actor function is also performed time-triggered. The execution frequency has to be specified by the developer. The sensor execution takes thereby place at the begin of each interval, the actor execution at the end of each interval. Both executions are regarded as instantaneous. To legitimate this assumptions the functions must represent short sequential code without synchronization points and blocks. For example in case of a network device the sensor functions may check the arrival of a message and copy the message into a port but a block until the receive event of a new message is not allowed.

Mode Applications can have different operational mode. To support this feature the Zerberus language introduces modes. A mode is a set of tasks, sensors and actors those are currently active on the Zerberus units. In addition a mode is specified by the mode cycle duration. Within each mode cycle the tasks, sensors and actors are executed according to their frequency as specified in the mode declaration.

Figure 7.7 shows the declaration of an example mode m. m contains two tasks t1 with frequency 1 and t2 with frequency 2, a sensor s with frequency 1 and an actor with frequency 2. The duration of one mode cycle is set to 50 ms.

The formal execution model is depicted in fig. 7.8 under the assumption that the mode cycle starts at time t. At time t the function of sensor s is executed and the tasks t1 and t2 are started. At time t+25ms the task t2 is stopped and the actor function is executed. Afterwards the task t2 is started for a second time. At the end of the mode cycle at t+50ms both tasks are stopped and the actor a is executed a second time. The execution of the sensor and actor functions appear instantaneous in the execution model.

Figure 7.9 shows a possible actual execution of the mode cycle on the machine. In addition to the task execution also the times required for the actor and sensor function execution, as well as the time consumed for run-time system execution have to be considered. The



```
mode m
{
          task= t1 1,t2 2;
          actor= a 2;
          sensor= s 1;
          duration= 50000000 ns;
}
```

Figure 7.7: Mode declaration

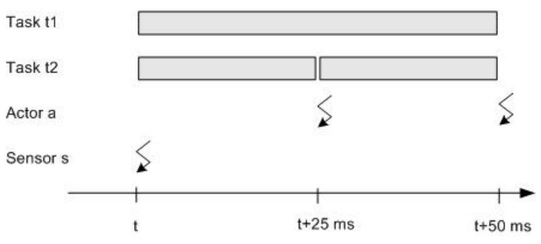


Figure 7.8: Formal execution model

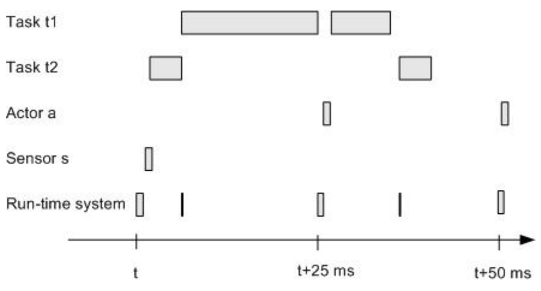


Figure 7.9: Actual execution

run-time system realizes the scheduling of the tasks, the port accesses and the voting and synchronization with the other Zerberus units.

The scheduler used in the example of fig. 7.9 uses a Earliest-Deadline-First strategy for the task execution. Sensors and actors are executed within the run-time system context.

Modechange To enable the switch between different operation modes modechanges can be used. A modechange is a function implemented by the developer that evaluates if a mode should be switched or not. The developer has to specify within the modechange declaration the target mode and a non-empty set of source modes. The evaluation of the function, which is based on the values of the assigned ports, takes place always at the end of the source mode cycles.

Mode switches must be deterministic, this means that for every achievable configuration (port values and modes) at most one assigned modechange can reach a positive evaluation for a modechange. This condition is checked in the Zerberus System at run-time.

Guard Guards are another possibility to change the behavior of a Zerberus program. Guards are similar to modechanges functions based on port values but while modechanges should be used for different operation modes, guards can be used to control individual tasks. Thereto the guard is assigned to a certain tasks. At the begin of every invocation of this task, the guard function is evaluated and only in case of a positive evaluation the according task is started. The main advantage of guards over modechanges is therefore their flexibility. A guard can be used also within a mode cycle and not only at the end of the mode cycle.

Introductory example In this paragraph an example application is used to illustrate the Zerberus language. The application scenario is depicted in 7.11. A robot control should be designed with the following tasks: the robot is placed in a biochemical laboratory and should move around, pick up some objects and deliver these to their goal. For the movement collision avoidance algorithms should be used.
The robot used several devices to accomplish its tasks: by radio the exact position of the robot can be determined, new jobs are received via wireless LAN, and a camera can detect objects.
The first step in the design is to extract the functional elements of the application. Here on the first glance there exist two functionalities: the control of the robots movements (robot path planer) and the control of the robots arm (arm path planer). As the algorithms are very complex and time-consuming (WCETs 1 second), other tasks are introduced to achieve a smoother track of the robot: the tasks robot control and joint control. These tasks use the results of the previous invocation of the time-consuming tasks and are executed with a much higher frequency (200Hz).
The sensor functions are executed with a frequency of 1Hz (since they are used only by the slower tasks), while the actor functions are executed with the frequency of the faster tasks.
Regarding the application it becomes obvious that it can be split up into two different application modes: in the first mode the robot moves towards the object or the goal, in the second mode the object is picked up or placed to its destination. The current mode is switched dependent on the associated path planer (if no action is contained in the planer, a switch can occur) and on the state of the job queue. Guards are not used within this application.

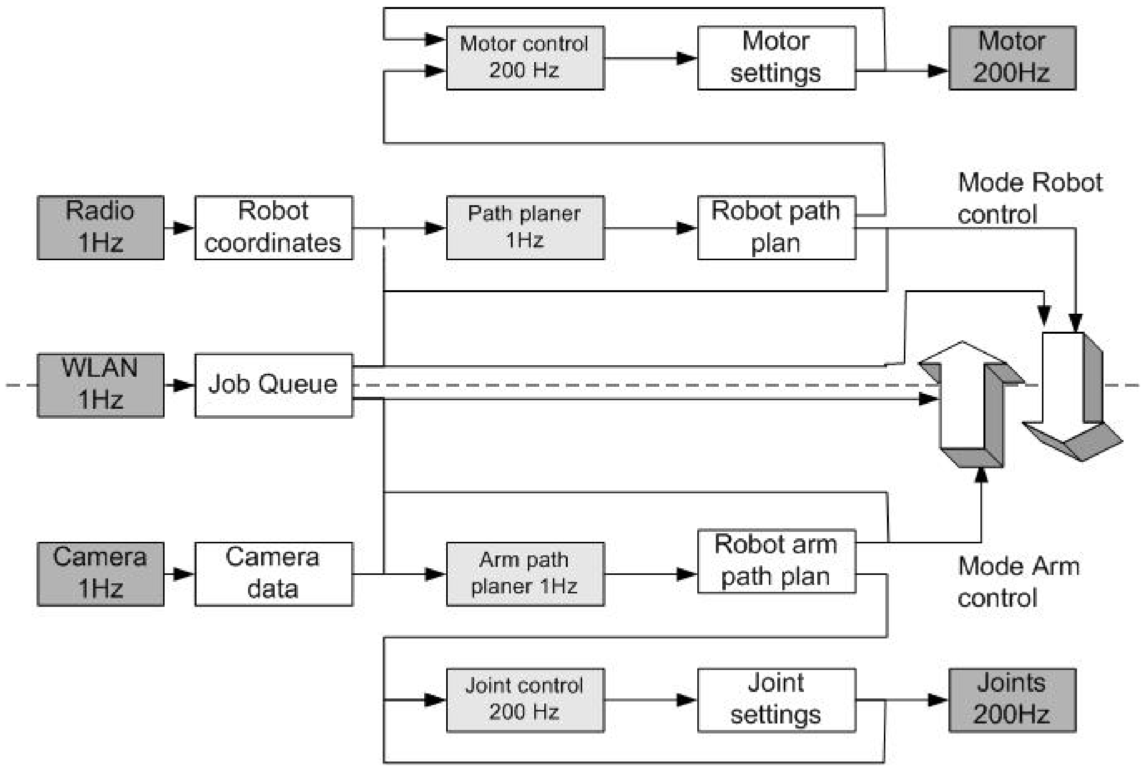


Figure 7.10: Formal model of example application

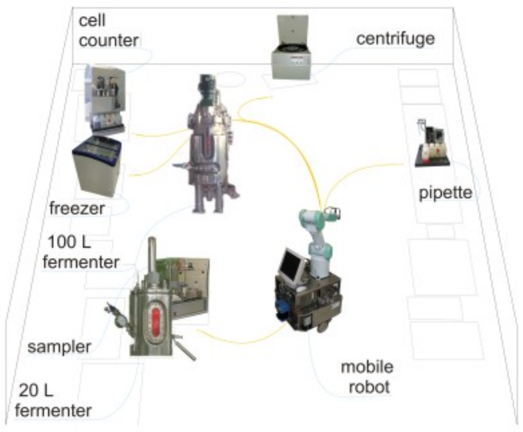


Figure 7.11: Introductory example

7.4 Modifications of Zerberus System regarding augmented reality applications

The demands as stated in section 7.2 are satisfied by the Zerberus System. But there exist some problems: first of all the current version of the Zerberus System requires the use of structural redundancy. This may not be desirable since the extra costs for the redundancy may not be countervailed by the safety and availability attributes. Especially in case of non safety-critical systems this is a reason that would forbid the use of the Zerberus System.
Another problem is the destructive voting: in case of the medical application example instead of a pure voting a merge of the results based on the different sensor data may be the appropriate technique.
For these two problems a solution must be suggested.

Abdiction of the requirement for redundancy The mechanisms provided for the redundancy are very useful in case the application is process on distributed processors. In particular the synchronisation algorithms are very useful in this scenario. But often one processor is absolutely satisfying or an application is indeed processen distributed but no redundant processes are required. Out of this reason the Zerberus System has to be extended in a way that the different tasks can be assigned to the used processors. Some tasks may then be calculated redundant, others may be singular.
In case one unit might fail also algorithms for task relocation might be very useful.
Since a support of different fault-tolerance mechanisms that are not based on structural redundancy is required and foreseen, this modification is already in progress.

Constructive voting As already mentioned other fault-tolerance mechanisms will be supported soon by the Zerberus System. This will be achieved by seperating the functionality of the application from the fault-tolerance mechanisms. Another language will be introduced in the style of an Event/Exception/Handling method.

- Event: An event marks the point in time where fault-tolerance mechanisms are executed. These mechanisms can be a voting, a plausibility test or a user defined function. An appropriate interface will be offered.
- Exception: An exception marks the detection of a fault within one component.
- Handling: For the fault handling the user has to specify an application dependent reaction to the fault.

By the proposed mechanism also the realisation of a constructive voting could be realized. Therefore the applicability of the Zerberus System in the domain of augmented reality can be achieved.

7.5 Summary

In this paper the requirements of application in the domain of augemented reality were specified. With the Zerberus System, a development model with integrated tool-chain was supposed that helps the developer to fulfill these requirements. The remaining problems were addressed and modifications were suggested that allow the usage of the Zerberus System for the development of augmented reality application.

Bibliography

[1] R. Azuma, *A survey of augmented reality*, 1995.

[2] C. Buckl, *Implementierung eines Entwicklungssystems für fehlersichere und zuverlässige Kontrollsysteme*, Feb. 2004.

[3] C. Buckl, *Zerberus Language Specification Version 1.0*, Tech. Rep. TUM-I0501, TU München, Jan. 2005.

[4] T. A. Henzinger, B. Horowitz, and C. M. Kirsch, *Giotto: A time-triggered language for embedded programming*, Proceedings of the First International Workshop on Embedded Software (EMSOFT), (2001), pp. 166 – 184.

[5] H. Kopetz and G. Bauer, *The Time-Triggered Architecture*, Proceedings of the IEEE, 91 (2003), pp. 112 – 126.

[6] S. Poledna, A. Burns, A. Wellings, and P. Barrett, *Replica Determinism and Flexible Scheduling in Hard Real-Time Dependable Systems*, IEEE Transactions on Computers, 49 (2000), pp. 100–110.

[7] D. K. Pradhan, *Fault-Tolerant Computer System Design*, Prentice Hall, 1996.

8 Holography

Alexej Minin, St. Petersburg State University

Holography dates from 1947, when British/Hungarian scientist Dennis Gabor developed the theory of holography while working to improve the resolution of an electron microscope. Gabor, who characterized his work as "an experiment in serendipity" that was "begun too soon," coined the term hologram from the Greek words holos, meaning "whole," and gramma, meaning "message." So we can use not only a visible light to produce holograms but also we can use particles, which have properties of waves. Holograms using electrons (considered in their "wave" manifestation, not as particles) provide sharp pictures, but because the electrons cannot penetrate far into a solid sample, the imaging process is usually restricted to surface regions. Holograms using x rays go can penetrate much farther, but their limitation consists of the fact that the penetration depth improves as the square of the atomic number. Therefore x-holography is not very good for materials with light elements. Holograms with neutrons are different; rather than scattering from the electrons in the atoms of the sample, neutrons scatter only from nuclei, which are 100,000 times smaller than the atoms in which they reside. In an experiment carried out with a beam of neutrons from a reactor at the Institute Laue-Langevin in Grenoble, a group of scientists has produced, for the first time, an atomic-scale map of a crystal.

8.1 History

Holography dates from 1947, when British/Hungarian scientist Dennis Gabor developed the theory of holography while working to improve the resolution of an electron microscope. Gabor, who characterized his work as "an experiment in serendipity" that was "begun too soon," coined the term hologram from the Greek words holos, meaning "whole," and gramma, meaning "message."

Gabor's first paper on holography evoked immediate response from scientists worldwide. Among those who made important contributions to the development of the technique were G.L. Rogers, A.B. Baez, H. El-Sum, P. Kirkpatrick and M.E. Haine. In these early years, the mercury arc lamp was the most coherent light source available for making holograms. Because of the low coherency of this light, it was not possible to produce holograms of any depth, thus restricting research. Despite equipment limitations, these researchers identified many of the properties of holography and further elaborated on Gabor's theory. Most important, they extended their understanding of the process and its potential to another generation of scientists.

Gabor's holography was limited to film transparencies using a mercury arc lamp as the light source. His holograms contained distortions and an extraneous twin image. Further development in the field was stymied during the next decade because light sources available at the time were not truly "coherent" (monochromatic or one-color, from a single point, and of a single wavelength).

This barrier was overcome in 1960 with the invention of the laser, whose pure, intense light was ideal for making holograms. For the next ten years, holography techniques and applications mushroomed.

In 1962 Emmett Leith and Juris Upatnieks of the University of Michigan recognized from their work in side-reading radar that holography could be used as a 3-D visual medium. In 1962 they read Gabor's paper and "simply out of curiosity" decided to duplicate Gabor's technique using the laser and an "off-axis" technique borrowed from their work in the development of side-reading radar. The result was the first laser transmission hologram of 3-D objects (a toy train and bird). These transmission holograms produced images with clarity and realistic depth but required laser light to view the holographic image.

Also in 1962 Dr. Yuri N. Denisyuk of the U.S.S.R. combined holography with 1908 Nobel Laureate Gabriel Lippmann's work in natural color photography. Denisyuk's approach produced a white-light reflection hologram which, for the first time, could be viewed in light from an ordinary incandescent light bulb. Once Denisyuk's work became known in the US, three teams of workers set out to take the off-axis recording technique used in laser transmission holography and apply it to reflection holography. These researchers were: E. Leith, J. Upatnieks, A. Kozma, J. Marks and N. Massey (University of Michigan); G. Stroke, A. Labeyrie (University of Michigan) with K. Pennington and L. Lin (Bell Labs); and C. Schwartz and N. Hartmann (Batelle Memorial Institute). By the Fall of 1965, each group had successfully recorded off-axis reflection holograms within months of each other. The U.S. patent for the process was issued to Hartmann, and marked a further advance for holography as a display medium.

8.2 What is Holography

Before you start building this display holography system and producing holograms, you need to see a visual overview of what a basic system looks like when it is completely set up and ready to record a hologram. Figure 1 shows a photographic view (left) as well as a top-view diagram (right) of a basic system. The system I will be describing is tried and true, but there are other systems that can be built (like a sandbox system). I have found this system, though, to be perfect for creating excellent 4" x 5" white-light reflection display holograms. That's the objective of all this: to produce a high quality white-light reflection display hologram that can be hung on your wall and shown-off.

Figure 8.1: (Left) Photograph of a transmission holography setup. (Right) Diagrammatic top view of the setup. (L = laser, DM = directional mirror, BS = beamsplitter, O = object beam, R = reference beam, DL = diverging lens, PH = plate holder, PM = parabolic mirror, OS = object scene)

This particular system has five basic optical components. They are the laser (L), beamsplitter (BS), directional mirrors (DM), diverging lenses (DL), and the parabolic mirror (PM).

In addition to the optical components, there is the object or scene (OS), the photographic plate holder (PH), the table mounts and the optical holders. Although this system shown in Figure 1 uses the same optical table and components that you will build, you will not be using this particular optical arrangement since it does not lend itself well to creating high quality display holograms

You must use a laser to make a hologram. A laser is a source of coherent light necessary to produce a high-quality hologram. Fully coherent light sources, such as lasers, are both spatially coherent and temporally coherent. A laser emits light in a very narrow beam and is considered a point source (spatially coherent), as opposed to an extended source (spatially incoherent) such as an incandescent bulb or a fluorescent lamp. A laser also emits light of a single color or wavelength (temporally coherent) whereas the light bulb or fluorescent lamp (temporally incoherent) emits light of many wavelengths.

8.2.1 How is a hologram made?

A laser beam is split into two beams: The reference beam is spread by a lens or curved mirror and aimed directly at the film plate The object beam is spread and aimed at the object. The object reflects some of the light on the holographic film-plate. The two beams interact forming an interference pattern on the film. This is the hologram. Laser light is needed because it is made of coherent waves (of same wavelength and phase). The principle of holography was discovered in Britain by Dennis Gabor in 1948. He was awarded the Nobel price for this discovery in the early 70's.

8.2.2 How is a hologram viewed?

When the hologram is illuminated from the original direction of the reference beam, a 3-dimensional image of the object appears where the object was originally. Some holograms must be viewed with laser or monochromatic (single color) light, and others with white light.

8.2.3 What are the main types of holograms?

Transmission Holograms: Viewable with laser light. They are made with both beams approaching the film from the same side. Reflection (White Light) Holograms: Viewable with white light from a suitable source such as spotlight, flashlight, the sun, etc. They are made with the two beams approaching the holographic film from opposite sides.

Multiple channel holograms

Two or more images are visible from different angles. There are different types of multiple channel holograms:

1. Simple ones with 2, 3, or a few images each viewed from a different angle.
2. Multiplex: A large number of "flat" pictures of a subject viewed from different angles are combined into a single, 3-dimensional image of the object. A composed hologram.
3. Rainbow holograms: The same image appears in a different color when viewed from different angles.

Real Image Holograms (H-2's)

These are usually reflection holograms made from a transmission original (H-1). The image dramatically projects in front of the plate towards the viewer. Most holograms in holography museums are of this type. The procedure for making them is quite elaborate and demands precise control of angles. Mass-Produced Holograms

1. Embossed – Made by stamping on foiled backed mylar film using a metal master (most common method).
2. Polymer – Made from light sensitive plastic. The Polaroid Corporation mass produces holograms by this method.
3. Dichromates – Very vivid holograms on jewelry, watches, etc. They are recorded on a light sensitive coating of gel that contains dichromate.

8.3 X-ray and g-ray Holography Improve Views of Atoms in Solids

New developments make possible the imaging of light atoms and the removal of image distortions. Although optical and electron holography have by now become commonplace, they have their limitations. In optical holography, the resolution is restricted by the wavelength of the light to several hundred nanometers. Lens aberrations in electron microscopes have prevented the achievement of atomic resolution in electron holography. In addition, the strong electron interactions in solids complicate extracting information from the holographic interference pattern and confine application of low-energy electron holography to surface studies.

In contrast, the short wavelengths of hard x rays and g rays (on the order of 1) offer the potential for obtaining atomic resolution from the bulk. Furthermore, x rays are only weakly scattered in solids, which simplifies the interpretation of the images. Whereas x-ray diffraction relies on the long-range translational periodicity, x-ray and g-ray holography are local methods that image the environment around selected atoms. Recent advances in x-ray and g-ray holograms are increasing their information content, improving the quality of the reconstructed images, and allowing the imaging of light atoms and noncrystalline and doped samples. From the inside out

X-ray holography initially presented a greater challenge to experimenters than optical or electron holography because of the difficulty of obtaining an x-ray source with sufficient coherence. Without good coherence, the phase information is washed out. Also, for atomic resolution, the position of an external source must be stable with respect to the sample to within an angstrom. An elegant solution to these difficulties was proposed 15 years ago by Abraham Szke at Lawrence Livermore National Laboratory: Use atoms or nuclei within the sample as the source.

First implemented for holography with photoelectrons and Auger electrons, the technique also works with x rays and g rays.I illustrates the concept. Within a sample, atoms stimulated with an external source such as x rays or high-energy electrons, or nuclei undergoing radioactive decay, can emit radiation, which can reach the detector directly (the reference wave) or after scattering off the electrons of nearby atoms (the object wave). The pattern of interference between the direct and scattered radiation can be mapped out by varying the angular position of the detector. This pattern can be viewed as a hologram from which a real-space image can be reconstructed numerically. Using this approach, Mikls Tegze and Gyula Faigel at the Research Institute for Solid-State Physics and Optics in Budapest, Hungary, reported the first x-ray hologram with atomic resolution five years ago (see Physics Today, May 1996, page 9*). They used an external x-ray source to eject core electrons from strontium atoms in single-crystal SrTiO3. As the atoms relaxed, they emitted fluorescent photons with a wavelength of 0.87 . Since photons were emitted not from a single Sr atom but from all the atoms in the crystal, the real-space image reconstructed from the hologram represented the average local environment around the Sr atoms. The amplitude of the holographic fringes was only about 0.3% of the background, but that was sufficient to allow the reconstruction of the 3D arrangement of the Sr atoms in the crystal.

A powerful extension of this technique of x-ray fluorescence holography (XFH) was demonstrated soon afterward by Thomas Gog and coworkers from the German Electron Synchrotron Laboratory (DESY), the University of California, Davis, and Lawrence Berkeley National Laboratory. They switched the positions of the source and detector to obtain what's essentially the time-reversed process. Here, atoms within the sample served as detectors, and the strength of the fluorescence they emitted depended on the interference between the external radiation reaching them either directly (the reference wave) or after scattering off nearby atoms (the object waves). In this approach, the fluorescence detector was fixed and the incident direction of the source radiation was varied with respect to the crystal axes.

8.3.1 The twin problem

The time-reversed approach has the advantage of being able to record holograms at multiple energies above the x-ray absorption edge–hence the name multiple-energy x-ray holography (MEXH). John Barton (now with Hewlett-Packard) had previously shown that photoelectron holograms taken at multiple energies can be used in a Fourier-transform-like approach to remove so-called twin images and aberrations, ubiquitous problems for all holographic methods. Holography doesn't really record the phase of the object wave with respect to the reference wave, but only the cosine of the phase difference. A twofold sign ambiguity therefore remains. As a result, in the reconstruction of a 3D image from the hologram, one gets not only the real image but also a twin image that's inverted about the reference point (here, the fluorescing atom). The superposed images can be out of phase with each other, which can lead to cancellations, distortions, and problems with atom identification. But with holograms acquired at 8-10 different energies, one can solve the local structure unequivocally–something quite difficult to do with ordinary x-ray diffraction.

Demonstrations of the potential of x-ray holography are continuing. The scattering cross section for x rays scales with atomic number, and so heavy atoms are easier to see than light ones. However, last fall, Tegze, Faigel, and colleagues, working at the European Synchrotron Radiation Facility, achieved sufficient sensitivity with MEXH to image light atoms such as oxygen. They have also recorded the local atomic structure in the icosahedral quasicrystal AlPdMn. Although it lacked long-range translational order, the quasicrystal had sufficient orientational order to allow the direct visualization of the icosahedral arrangement of atoms, shown in figure 2, averaged over the handful of different Mn environments. Kouichi Hayashi and coworkers from Kyoto University and the Japan Synchrotron Research Institute have explored MEXH for imaging the local environment around zinc atoms doped into gallium arsenide. And Larry Sorensen's group at the University of Washington has generated holograms with x rays produced by means of bremsstrahlung. g-ray holography

"The biggest difficulty with all x-ray holography is that the contrast is very low," says

Szke. Nuclei with low-lying excited states can also serve as the radiation source for atomic-resolution holography. Such nuclei, used for Mssbauer spectroscopy, can emit ? rays with very well-defined energy. These photons scatter not only off electrons, but also off nuclei through a resonant process that has a cross section two orders of magnitude larger. Consequently, g rays can provide much better contrast, but only a few dozen isotopes have the requisite Mssbauer resonance to serve as reference nuclei and as scatterers. Fortunately, the most popular Mssbauer isotope, iron-57, has a resonance at a wavelength of 0.86 , suitable for atomic-resolution holography. Using an epitaxial Fe film enriched in this isotope, Pawel Korecki at Jagiellonian University (now at DESY) and coworkers at the University of Mining and Metallurgy in Krakow, Poland, first demonstrated atomic-resolution holography with g rays. The g rays were provided through the radioactive decay of cobalt-57. The researchers used the inverse, internal-detector configuration, thereby avoiding the need to embed 57Co within the sample. The absorption of g rays by 57Fe, sensitive to the interference between direct and scattered radiation reaching the nuclei, was monitored through the flux of conversion electrons emitted as the nuclei relaxed. Having the g-ray source outside the sample has another advantage. As in Mssbauer spectroscopy, the frequency of the photons can be tweaked by adjusting the relative motion of the source with respect to the sample. In February, Korecki and colleagues capitalized on that ability to suppress twin images using "complex" holograms[1] Working with the Mssbauer resonance essentially fixes the photon wavelength, but it's possible to change the photon phase. As with any resonance, the g rays that scatter off the nuclei experience a phase shift that varies with the detuning from the center of the resonance. Using g rays equally detuned on either side of the narrow Mssbauer resonance peak, the researchers could take linear combinations of two holograms to separate the real and imaginary parts. From the combination of those complex components, an accurate, twin-free real-space image can be reconstructed, as shown in figure 3.

The exploitation of the g-ray phase shifts near a resonance is similar to the x-ray crystallography technique of multiple-wavelength anomalous dispersion. When x rays close to absorption edges are used in diffraction, they also undergo phase shifts; by comparing diffraction patterns from x rays of different wavelengths, phase information normally lost in x-ray diffraction can be inferred, yielding additional insight into the underlying structure.

8.3.2 Work in progress

To a large extent, x-ray and g-ray holography are currently detector-limited. But they place less stringent demands on the crystalline perfection of samples than standard x-ray diffraction does. "Whereas a 1 misalignment can kill normal x-ray diffraction," says Charles Fadley of UC Davis, "x-ray holography can tolerate many crystallites with a misalignment of up to a few degrees." Given the difficulty of crystallizing many materials, including biological and other large molecules, such samples may in the future be amenable to study by holographic methods even though they aren't candidates for diffraction investigation. Because the Mssbauer resonance can be shifted due to hyperfine coupling to electrons, g-ray holography has the potential for imaging the local magnetic environment. It may also find use in imaging low-dimensional systems.

[1]http://www.physicstoday.org/pt/vol-54/iss-4/p21.html

8.3.3 Neutron holography

Now the American Institute of Physics, in its Physics News 609. from October 15, 2002, informs us that neutron holography with atomic-scale resolution has been performed, for the first time, with an "inside-detector" approach. Physics News explains some more things: Holograms with visible light are common enough: they adorn most credit cards. Holograms using electrons (considered in their "wave" manifestation, not as particles) provide sharp pictures, but because the electrons cannot penetrate far into a solid sample, the imaging process is usually restricted to surface regions. Holograms using x rays go can penetrate much farther, but their limitation consists of the fact that the penetration depth improves as the square of the atomic number. Therefore x-holography is not very good for materials with light elements. Holograms with neutrons are different; rather than scattering from the electrons in the atoms of the sample, neutrons scatter only from nuclei, which are 100,000 times smaller than the atoms in which they reside. This is important when it comes time to reconstruct an image of the interior of a crystal lattice. In an experiment carried out with a beam of neutrons from a reactor at the Institute Laue-Langevin in Grenoble, a group of scientists has produced, for the first time, an atomic-scale map of a crystal, in particular a sample of lead atoms, using a technique in which the "detector," a trace amount of atoms (cadmium-113) whose nuclei readily absorb neutrons, are embedded inside the sample itself. The holographic process unfolds as follows: neutron waves can strike a Cd nucleus directly (reference beam) or by first scattering from a Pb nucleus. In either case, the absorption of a neutron stimulates a Cd nucleus to emit a high energy photon observable in a nearby detector. The overall interference pattern for these two processes (absorbing scattered or direct neutron waves) is monitored as the profile of the sample to the beam is stepped through various angles. The result: a crisp picture of a unit cell of 12 lead atoms (see figure from AIP.org). This process should be great for spotting foreign atoms in a solid (dopants if the atoms are desired, impurities if they're not). Since the neutron has a magnetic moment, n-holography might also contribute to an understanding of the magnetic nature of the sample atoms, in addition to imaging their whereabouts. (Cser et al., Physical Review Letters, 21 October 2002). As you can see in the illustration, neutron holography is a thing of beauty and a joy forever.

8.4 About My Task

Electric field in some point r(x,y) is describing with such formula 1) Where a(r,t) is amplitude of the field. ? is a phase. Lets use Eiler formulas to rewrite (1) formula So E(r,t) we can rewrite in this way:

And in optics we write E(r,t) in this way: Where f(r,t) is an optic signal a(r)-is an amplitude of the signal And at list we have the expression we want to count in every point of hologram: Intensity in some point r (x,y) . So the problem is that it is very difficult to count such integrals with enough speed. Because the typical size of the hologram is about 1mln x 1mln pixels. So the number of integrals we have to use is 1012 . I used different approaches to this problem:

1. Upper darboux summes and lower darboux summes (Very big error)
2. (Upper Darbu + lower Darbu)/2 ? (It takes more time, but the fault is smaller)
3. Simpson method. (Interpolation or exterpolation using polynoms)

The main idea is that we have to count integrals very fast and enough accurate. (fault ?10But it is just an overview of the task.!!!

9 Planes and Homographies for Augmented Reality

Irina Bobkova, State University St. Petersburg

This is the abstract of that chapter. Try to be not longer than 10 lines here.

10 Context coding of overlapped DCT coefficients

Denis Bessonov, State Univ. of Aerospace Instrumentation St. Petersburg

Two-dimension DCT (Discrete Cosine Transform) is the commonly used orthogonal transform applying to the image compression at the time. The main steps of image compression are: DCT, quantizing of transform coefficients, scanning the block in zig-zag order and encoding zero series and nonzero coefficients. However, for getting the high quality compression quantization steps are to be chosen small enough. In this case zero series become very short or disappear at all and this method of their encoding becomes inefficient. In order to get a better performance at high bit rates it is reasonable to use the context encoding on ?coefficient-by-coefficient? basis (i.e. when there are no zero series). But the generic DCT doesn?t give much correlation between adjacent blocks. In addition it destroys the original distribution of transform coefficients because of rectangle-shaped window using in transforming of separate blocks. So the overlapped DCT with sinusoidal window covering 2 neighboring blocks has been proposed and investigated as well as context model for encoding of coefficients. The author presents the analysis showing that the compression is improved up to 10% at 40-44 dB PSNR comparatively with the standard not overlapping DCT.

11 Particle Filters

Gordana Stojceska, Technische Universität München

In recent years, particle filters have found widespread application in domains with noisy sensors, such as computer vision and robotics, as well as in many other technology fields. Particle filters are powerful tools for Bayesian state estimation in non-linear systems. The key idea of particle filters is to approximate a posterior distribution over unknown state variables by a set of particles, drawn from this distribution. This paper addresses a primary definition and methods of particle filters: Particle filters are insensitive to costs that might arise from the approximate nature of the particle representation. Their only criterion for generating a particle is the posterior likelihood of a state. This paper gives also short introduction to Bayesian filters, continuing with the most known methods, as well as the advantages and disadvantages of the particle filters as one of the most use tracking methods. There is also one short overview of particle filter aplication in the area of tracking people.

11.1 Introduction

In many application areas it is becoming very important to include elements of nonlinearity and non-Gaussianity, so that one can model accurately as good as possible the underlying dynamics of a physical system. Moreover, it is typically crucial to process data on-line as it arrives, both from the point of view of storage costs as well as for rapid adaptation to changing signal characteristics. In this paper it is introduced one of the most successful method used mostly for nonlinear/non-Gaussian tracking problems, the particle filters method. Particle filters are sequential Monte Carlo methods [2] based on point mass (or "particle") representations of probability densities, which can be applied to any state-space model and which generalize the traditional Kalman filtering methods.

Many scientific problems require estimation of the state of a system that changes over time using a sequence of noisy measurements made on the system. Here the accent is made on the state-space approach to modelling dynamic systems, and the focus it is on the discrete-time formulation of the problem. Thus, difference equations are used to model the evolution of the system with time, and measurements are assumed to be available at discrete times. For dynamic state estimation, the discrete-time approach is widespread and convenient. The state-space approach to time-series modelling focuses attention on the state vector of the system. The state vector contains all relevant information required to describe the system under investigation. For example, in tracking problems, this information could be related to the kinematics' characteristics of the target. Alternatively, in an econometrics problem, it could be related to monetary flow, interest rates, inflation, etc. The measurement vector represents (noisy) observations that are related to the state vector. The measurement vector is generally (but not necessarily) of lower dimension than the state vector. The state space approach is convenient for handling multivariate data and nonlinear/non-Gaussian processes, and it provides a significant advantage over traditional time-series techniques for these problems. A full description is provided in [3]. In addition, many varied examples illustrating the application of nonlinear/non-Gaussian state space models are given in [4].

In order to analyze and make inference about a dynamic system, at least two models are required: First, a model describing the evolution of the state with time (the system model) and, second, a model relating the noisy measurements to the state (the measurement model). We will assume that these models are available in a probabilistic form. The probabilistic state-space formulation and the requirement for the updating of information on receipt of new measurements are ideally suited for the Bayesian approach. This provides a rigorous general framework for dynamic state estimation problems. In the Bayesian approach to dynamic state estimation, one attempts to construct the posterior probability density function (pdf) of the state based on all available information, including the set of received measurements. Since this pdf embodies all available statistical information, it may be said to be the complete solution to the estimation problem. In principle, an optimal (with respect to any criterion) estimate of the state may be obtained from the pdf. A measure of the accuracy of the estimate may also be obtained. For many problems, an estimate is required every time that a measurement is received. In this case, a recursive filter is a convenient solution. A recursive filtering approach means that received data can be processed sequential rather than as a batch so that it is not necessary to store the complete data set, nor reprocess existing data if a new measurement becomes available.

One such a filter consists of essentially two stages: prediction and update. The prediction stage uses the system model to predict the state pdf forward from one measurement time to the next. Since the state is usually subject to unknown disturbances (modelled as random noise), prediction generally translates, deforms, and spreads the state pdf. The update operation uses the latest measurement to modify the prediction pdf. This is achieved using Bayes theorem, which is the mechanism for updating knowledge about the target state in the light of extra information from new data.

11.2 Non-Linear Bayesian Tracking

First of all it the problem of tracking should be defined. For that reason, one should consider the evolution of the state sequence $\{x_k,\ k \in N\ \}$ of a the target being tracked, which is given by the following equation:

$$x_k = f_k(x_{k-1}, v_{k-1}) \tag{11.1}$$

where f_k: $\Re^{n_x} \times \Re^{n_v} \rightarrow \Re^{n_x}$ is a non-linear function of the state x_{k-1}, $\{v_{k-1},\ k \in N\}$ is a process noise sequence, n_x is a dimension of the state vector, n_x is a dimension of the process noise vector, and N is the set of natural numbers. On the other hand, there should be introduced the measurement model, because the aim of tracking is to recursively estimate x_k from the measurements. The measurement vector is defined as follows:

$$z_k = h_k(x_k, n_k) \tag{11.2}$$

where h_k: $\Re^{n_x} \times \Re^{n_n} \rightarrow \Re^{n_z}$ is a non-linear function, $\{n_k,\ k \in N\}$ is a measurement noise sequence, n_n is a dimension of the measurement noise vector, n_z is a dimension of the measurement vector, and N is the set of natural numbers. Based on this information, what we need is filtered estimate for x_k given all available measurements $z_{1:k} = \{z_i, i = 1, .., k\}$ up to time k.

From Bayesian point of view, the tracking problem should solve and recursively compute some degree of probability in the state x_k at time k, taking different values, when the data $z_{1:k}$ are given. Therefore it is required to construct the pdf $p(x_k|z_{1:k})$. Almost always should be assumed that the initial pdf, $p(x_0|z_0) \equiv p(x_0)$, of the state vector, also called ***prior***, is avaliable (here z_0 is known as set of no measurements). Then the required pdf $p(x_k|z_{1:k})$ may be obtained recursively in two stages: ***prediction*** und ***update***. If we suppose that the required pdf in time $p(x_{k-1} \mid z_{1:k-1})$ at time k-1 is avaliable, then the prediction stage should obtain the prior pdf of the state at time k, using the system model (11.1). This is done with use of the Chapman-Kolmogorov equation:

$$p(x_k|z_{1:k-1}) = \int p(x_k|x_{k-1})p(x_{k-1}|z_{1:k-1})dx_{k-1} \tag{11.3}$$

It should be mentioned that in the equation (11.3) it has been made use of the fact that $p(x_k|x_{k-1}, z_{1:k-1}) = p(x_k|x_{k-1})$, since (11.1) describes a Markov process of order one. The probabilistic model of the state evolution, $p(x_k|x_{k-1})$, is defined by the system equation (11.1) and the known statistics of v_{k-1}.

At the time step k, the measurement z_k is avaliable and this may be used to update the prior (update stage) using the Bayes' rule:

$$p(x_k|z_{1:k}) = \frac{p(x_k|z_{1:k-1})p(z_k|z_{1:k-1})}{p(z_k|z_{1:k-1})} \tag{11.4}$$

where the normalising constant has the following form:

$$p(z_k|z_{1:k-1}) = \int p(z_k|x_k)p(x_k|z_{1:k-1})dx_k \tag{11.5}$$

This constant depends on the likelihood function $p(z_k|x_k)$, which is defined by the measurement model given by (11.2) and the already known statistics of n_k. In the update stage (11.4), the measurement z_k is used to modify the prior pdf to obtain the required posterior pdf of the current state (Fig 11.1).

The optimal Bayesian solution for the problem of tracking is based on the recurrence relations (11.3) and (11.4). Optimal Bayesian solution solves the problem of recursively calculating the exact posterior density function. The recursive propagation of the posterior density is only a conceptual solution, that in general, cannot be determined analytically, although solutions do exist in a very restrictive cases, including the Kalman and grid-based filters, which will be mentioned below. But in the cases when the analytical solution is

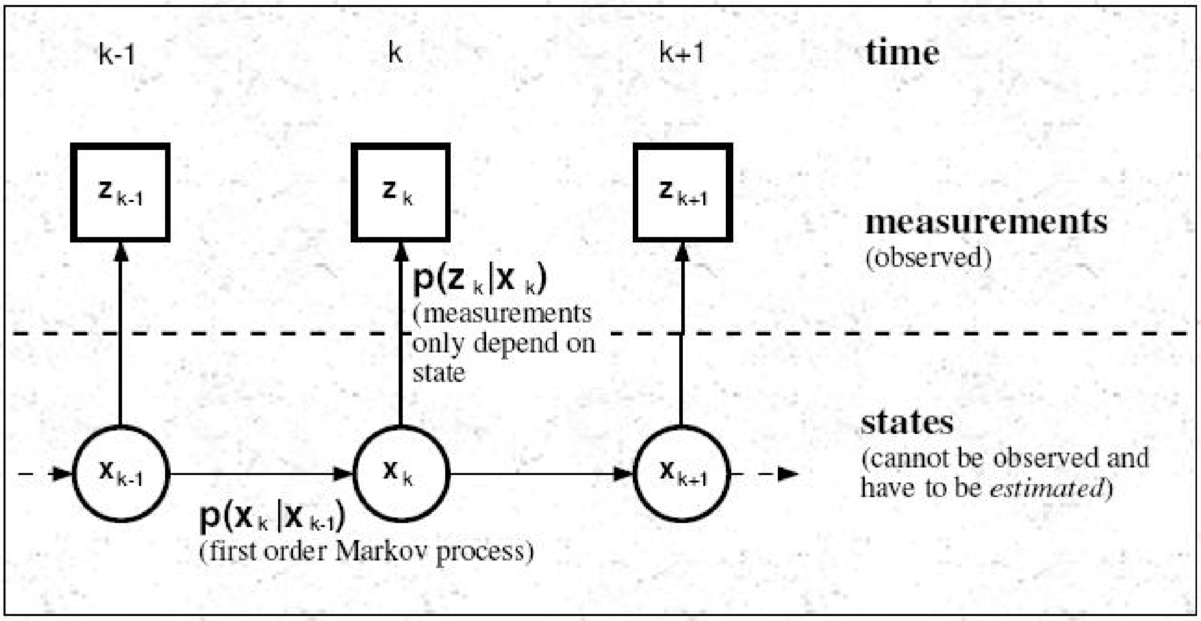


Figure 11.1: Visual representation of the measurement-state model

intractable, there could be found an approximation to the optimal Bayesian solution using the extended Kalman filter, approximate grid-based filter, as well as the particle filters. In this paper the main aspect is turned to the particle filters.

11.3 Suboptimal Algorithms

The Kalman and grid-based filter assume that the posterior density function at every time step is Gaussian and hence parameterised by a mean and covariance. Doing some computation, one can derive the optimal Bayesian solution to the tracking problem, under the abovementioned assumption. But in many situation of interest, the assumption might do not hold. The Kalman and grid-based methods can not be used, because in that case we are interested in approximation of the optimal solution. Then one can chose between the following approximate non-linear Bayesian filters (or in other words, suboptimal algorithms): the extended Karman filter, the approximate grid-based methods and the particle filters.

The extended Kalman filter always approximates $p(x_k|z1:k)$ to be Gaussian. If the true probability density function is non-Gaussian (eg. if it is bi-modal or heavily skewed) then a Gaussian can never describe it well. In such cases, approximate grid-based filters and particle filters will yield an improvement in performance in comparison to that of the extended Kalman filter [1]. Recently, the unscented transform has been used in an EKF (Extended Kalman Filter) framework, for example [6], [7], [8]. The resulting filter, known as the "Unscented Kalman Filter" considers a set of points that are deterministically selected from the

Gaussian approximation to $p(x_k|z1 : k)$. First these points are propagated through the true non-linearity and the parameters of the Gaussian approximation are then re-estimated. For some cases, this filter showed better performance than a standard EKF since it approximates the non-linearity much better than the extended Kalman filter. The parameters of the Gaussian approximation are improved. On the other hand, the grid-based method besides the advantages, has also some disadvantages. Gridbased approaches' disadvantage is the computational and space complexity required to keep the position grid in memory and to update it for every new observation. Because the complexity grows exponentially with the number of dimensions, we can apply grid-based approaches only to low-dimensional estimation problems, such as estimating a person's position and orientation [12]. Here our main focus is on the particle filters. In the next section there will be presented the particle filter method as well as the advantages and disadvantages of this method.

11.4 Particle Filters

The Sequential Importance Sampling (SIS) algorithm forms the basis for most particle filters that have been developed so far. Besides this one, there exists various algorithms, but almost all of them are derived from the SIS algorithm. Therefore this algorithms in the literature are regarded as special case of the general SIS algorithm. In order to see how they are related to each other, first we should show how the general particle filter algorithm works.

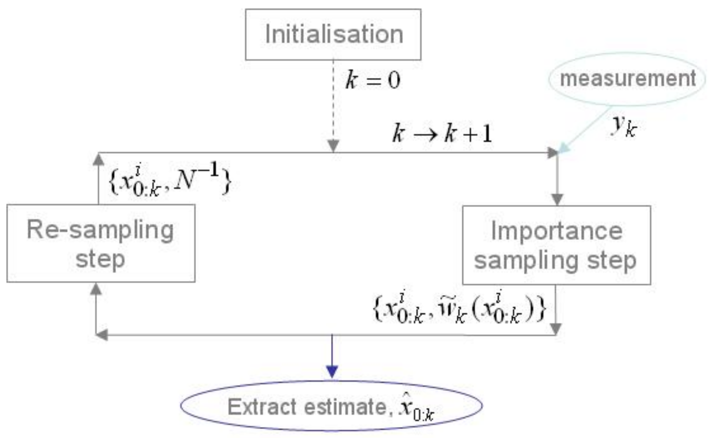


Figure 11.2: Schematic Particle Filter Method

The SIS algorithm is a Monte Carlo method thar forms the basis for most sequential Monte Carlo filters developed so far [10],[11]. This sequential monte carlo approach is known under different names in the literature as bootstrap filtering [12], the condensation algorithm [13], particle filtering [13], interacting particle approximations [14],[15] and survival of the fittest

[16]. This so called technique implements a recursive Bayesian filter by Monte Carlo Simulations. The main idea used for this technique is to respresent the required posterior density function by a set of random samples with assosiated weights and to compute estimates based on these samples and weights (Fig.11.2). As the number of samples becomes very large, the Monte Carlo approach becomes equivalent representation to the usual functional description of the posterior probability density function and then the SIS algorithm approaches the optimal Bayesian solution.

Now follows the derivation of the algorithm. Let

$$\{x^i_{0:k}, w^i_k\}^{N_s}_{i=1} \tag{11.6}$$

denote a ***Random Measure*** that characterises the posterior probability density function $p(x_{0:k}|z_{1:k})$, where

$$\{x^i_{0:k}, i = 0, ..., N_s\} \tag{11.7}$$

is a set of support points with associated weights

$$\{w^i_k, i = 1, ..., N_s\} \tag{11.8}$$

and $x_{0:k} = \{x_j, j = 0, ..., k\}$ is the set of all states up to time k. The weights are normalised such that the following holds:

$$\sum_i w^i_k = 1 \tag{11.9}$$

Then the posterior density function at time moment k can be approximated according to the following equation:

$$p(x_{0:k}|z_{1:k}) \approx \sum_{i=1}^{N_s} w^i_k \delta(x_{0:k} - x^i_{0:k}) \tag{11.10}$$

where $\delta(\circ)$ represents the Dirac delta measure. So in this way we can get a discrete weighted approximation to the true posterior, $p(x_{0:k}|z_{1:k})$. The weights are chosen with respect to the principle ***Importance Sampling*** [17],[18] which relies on the following: suppose $p(x) \propto \pi(x)$ is a probability density function from which it is somehow difficult to draw samples, but for which $pi(x)$ can be evaluated (and on this way also $p(x)$ up to proportionality. Let

$$x_i \sim q(x), i = 1, ..., N_s \tag{11.11}$$

be samples that are easily generated from a proposal $q(\circ)$, called an ***Importance density***. In such case, a weighted approximation to the density $q(\circ)$ is given by the following equation:

$$p(x) \approx \sum_{i=1}^{N_s} w^i \delta(x - x^i) \tag{11.12}$$

where

$$w^i \propto \frac{\pi(x^i)}{q(x^i)} \tag{11.13}$$

is the normalised weight of the i-th particle.

So, if the samples, $x^i_{0:k}$, were drawn from an importance dencity, $q(x_{0:k}|z_{1:k})$, then the weights in (11.10) are defined by (11.13) to be

$$w^i_k \propto \frac{p(x^i_{0:k}|z_{1:k})}{q(x^i_{0:k}|z_{1:k}} \tag{11.14}$$

In the sequential case, in each iteration one could have samples constituing an approximation to $p(x_{0:k-1}|z_{1:k-1})$, and probably wants to approximate $p(x_{0:k}|z_{1:k})$ with a new set of samples. If the importance density is chosen in such a way so that it factorise according to the following equation

$$q(x_{0:k}|z_{1:k}) = q(x_k|x_{0:k-1}, z_{1:k})q(x_{0:k-1}|z_{1:k-1}) \tag{11.15}$$

then one can obtain samples $x^i_{0:k} \sim q(x_{0:k}|z_{1:k})$ by augmenting each of the already klnown samples $x^i_{0:k-1} \sim q(x_{0:k-1}|z_{1:k-1})$ with the new state $x^i_k \sim q(x_k|x_{0:k-1}, z_{1:k})$. To derive the weight update equation, $p(x_{0:k}|z_{1:k})$ is first expressed in terms of $p(x_{0:k-1}|z_{1:k-1})$, $p(z_k|x_k)$ and $p(x_k|x_{k-1})$, and one can get:

$$p(x_{0:k}|z_{1:k}) = \frac{p(z_k|x_{0:k}, z_{1:k-1})p(x_{0:k}|z_{1:k-1})}{p(z_k|z_{1:k-1})} \propto p(z_k|x_k)p(x_k|x_{k-1})p(x_{0:k-1}|z_{1:k-1}) \tag{11.16}$$

With a substitution of (11.15) and (11.16) in (11.14), the weight update equation has the following form:

$$w^i_k \propto w^i_{k-1}\frac{p(z_k|x^i_k)p(x^i_k|x^i_{k-1})}{q(x^i_k|x^i_{0:k-1}, z_{1:k})} \tag{11.17}$$

Moreover, if $q(x_k|x_{0:k-1}, z_{1:k}) = q(x_k|x_{k-1}, z_k)$, then the importance density becomes only dependent on x_{k-1} and z_k. This is useful when only a filtered estimate of $p(x_k|z_{1:k})$ is required in each time step. Then one can store only the values of x^i_k and in the same time discard the path $x^i_{0:k-1}$ and the history of measurements $z_{1:k-1}$. Then the weight looks like:

$$w^i_k \propto w^i_{k-1}\frac{p(z_k|x^i_k)p(x^i_k|x^i_{k-1})}{q(x^i_k|x^i_{k-1}, z_k)} \tag{11.18}$$

and the posterior filtered density $p(x_k|z_{1:k})$ could be approximated as:

$$p(x_k|z_{1:k}) \approx \sum_{i=1}^{N_s} w^i_k\delta(x_k - x^i_k) \tag{11.19}$$

where the weights w^i_k are defined with (11.18). It could be also shown that as $N_s \longrightarrow \infty$ the approximation (11.19) approaches the true posterior density $p(x_k|z_{1:k})$.

Now we can summarize the results and show how the SIS algorithm according to these computation should look like. The algorithm consists of recursive propagation of the weights and support points as each measurement is received sequentially. The most important steps are:

- $[\{x^i_k, w^i_k\}^{N_s}_{i=1}] = SIS[\{x^i_k, w^i_k\}^{N_s}_{i=1}, z_k]$
- $FOR\ i = 1 : N_s$

  - *Draw samples* $x_k^i \sim q(x_k|x_{k-1}, z_k)$
  - *Assign the particle a weight,* w_k^i*, according to* (11.18)

- *END FOR*

Very common problem with the SIS particle filter algorithm is shown to be the degeneracy phenomenon. That means, after a few iterations, all but one particle will have negligible weight. It has been proved [18] that the variance of the importance weights can only increase over time, and therefore it is impossible to avoid the degeneracy phenomenon. A suitable measure of degeneracy of the algorithm is the so called effective sample size N_{eff} introduced in [15],[16] and defined as:

$$N_{eff} = \frac{N_s}{1 + Var(w_k^{*i})} \tag{11.20}$$

where $w_k^{*i}) = \frac{p(x_k^i|z1:k)}{q(x_k^i|x_{k-1}^i, z_k)}$ is referred to as the "true weight". Although this cannot be evaluated exactly, one can obtain an estimate using the following formula:

$$\widehat{N_{eff}} = \frac{1}{\sum_{i=1}^{N_s} (w_k^i)^2} \tag{11.21}$$

where w_k^i is normalised weight using (11.17). Because this problem is undesirable, one should find solution to avoid it. The first one is the brute force approach to use a very large N_s. But this is not that helpful, thus one can choose between two other methods:

1. Good choise of the importance density
2. Use of resampling

11.4.1 Good choise of the importance density

This method presents the importance density $q(x_k|x_{k-1}^i, z_k)$ minimising $Var(w_k^{*i})$ so that N_{eff} is maximised. The optimal importance density function has been shown [18] to be:

$$q(x_k|x_{k-1}^i, z_k)_{opt} = p(x_k|x_{k-1}^i, z_k) = \frac{p(z_k|x_k, x_{k-1}^i)p(x_k|x_{k-1}^i)}{p(z_k|x_{k-1}^i)} \tag{11.22}$$

Substitution of (11.22) into (11.18) lieds to:

$$w_k^i \propto w_{k-1}^i p(z_k|x_{k-1}^i) = w_{k-1}^i \int p(z_k|x_k^{'})p(x_k^{'}|x_{k-1}^{'})dx_k^{'} \tag{11.23}$$

But it is very often convinient to choose th importance density function to be the prior,

$$q(x_k|x_{k-1}^i, z_k) = p(x_k|x_{k-1}^i) \tag{11.24}$$

Substitution of (11.24) into (11.18) leads to:

$$w_k^i \propto w_{k-1}^i p(z_k|x_k^i) \tag{11.25}$$

This seems to be the most common way to choose the importance density function since it is intuitive and easy to implement. However, one can use many differen ways, since this choise is the crucial step in the design of a particle filter.

11.4.2 Resampling

The second method which could help to reduce the degeneracy is resampling. It can be used wehnever a significant degeneracy is observed, or that means when N_{eff} is below some threshold, N_T. The main idea of this method is the following: those particles which have small weights should be eliminated and those with large weights should be preserved. The resampling step involves generation of a new set $\{x^{i*}_k\}_{i=1}^N$ by resampling with replacement N_s times from an approximate discrete respresentation of $p(x_k|z_{1:k})$ given by (11.19). The resulting sample is in fact sample from the discrete density (11.19) and according to that the weights are now reset to $w^i_k = \frac{1}{N_s}$. There are various efficient resampling schemes [14],[16]. Figure (11.3) represents one example of the algorithms, namely the systematic resampling algorithm [14].

```
$[\{\mathbf{x}_k^{j*}, w_k^j, i^j\}_{j=1}^{N_s}] = \text{RESAMPLE } [\{\mathbf{x}_k^i, w_k^i\}_{i=1}^{N_s}]$
• Initialise the CDF: $c_1 = 0$
• FOR $i = 2 : N_s$
  – Construct CDF: $c_i = c_{i-1} + w_k^i$
• END FOR
• Start at the bottom of the CDF: $i = 1$
• Draw a starting point: $u_1 \sim \mathbb{U}\left[0, N_s^{-1}\right]$
• FOR $j = 1 : N_s$
  – Move along the CDF: $u_j = u_1 + N_s^{-1}(j-1)$
  – WHILE $u_j > c_i$
    * $i = i + 1$
  – END WHILE
  – Assign sample: $\mathbf{x}_k^{j*} = \mathbf{x}_k^i$
  – Assign weight: $w_k^j = N_s^{-1}$
  – Assign parent: $i^j = i$
• END FOR
```

Figure 11.3: ALGORITHM 2: Systematic Resampling

Although this method reduces the effects of the degeneracy problem, it indroduces other practical problems, like limit of the opportunity for parallelization since all the particles should be combined. One problem more is the statistical selection over and over again of the particles with a high weight, and this leads to loss of diversity among the particles as the resultant sample will contain many repeated points.

One special case of the general SIS algorithm is Sampling Importance Resampling (SIR) filter [14], one Monte Carlo method that can be applied to recursive Bayesian filtering problems. The assumptions required to use the SIR filter are very weak. The state dynamics and measurements functions, $f_k(\circ,\circ)$ and $h_k(\circ,\circ)$ in (11.1) and (11.2) respectively, need to be known, and also it is required to be able to sample realisations from the process noise distribution of v_{k-1} and from the prior. Finally, the likelihood function $p(z_k|x_k)$ needs to be avaliable for pointwise evaluation. Then the SIR algorithm can be easily derived from the SIS algorithm by an appropriate choise of: first, the important density ($q(x_k|x^i_{k-1}, z_{1:k})$ is chosen to be the prior density $p(x_k|x_{k-1}^i)$) and second, the resampling step (it should be applied to every time step/index).

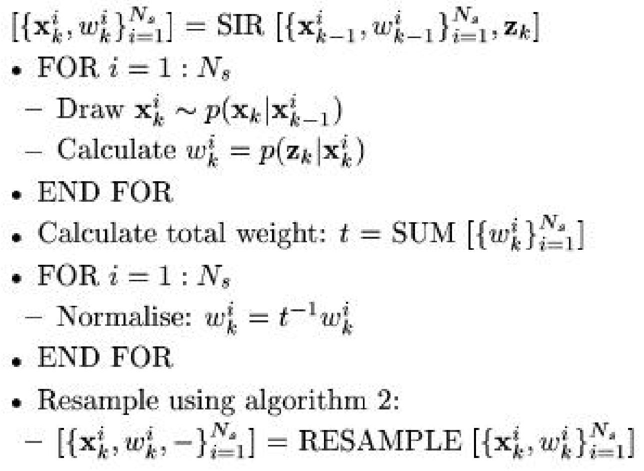
Such made choise of the importance density implies that one needs samples from $p(x_k|x_{k-1}^i)$. Such sample $x_k^i \sim p(x_k|x_{k-1}^i)$ could be generated by generatin a process noise sample $v_{k-1}^i \sim p_v(v_{k-1})$ and setting $x_k^i = f_k(x_{k-1}^i, z_{k-1}^i)$, where $p_v(\circ)$ is the probability density function of v_{k-1}. For this choise of the importance density the weights are computed as:

$$w_k^i \propto w_{k-1}^i p(z_k|x_k^i) \tag{11.26}$$

Since the resampling is applied at every time step, $w_{k-1}^i = \frac{1}{N} \forall i$ and then

$$w_k^i \propto p(z_k|x_k^i) \tag{11.27}$$

The weights given in (11.27) are normalised before the resampling stage. The SIR algorithm is represented in the figure (11.4).



```
$[\{\mathbf{x}_k^i, w_k^i\}_{i=1}^{N_s}] = \text{SIR}\ [\{\mathbf{x}_{k-1}^i, w_{k-1}^i\}_{i=1}^{N_s}, \mathbf{z}_k]$
• FOR $i = 1 : N_s$
  – Draw $\mathbf{x}_k^i \sim p(\mathbf{x}_k|\mathbf{x}_{k-1}^i)$
  – Calculate $w_k^i = p(\mathbf{z}_k|\mathbf{x}_k^i)$
• END FOR
• Calculate total weight: $t = \text{SUM}\ [\{w_k^i\}_{i=1}^{N_s}]$
• FOR $i = 1 : N_s$
  – Normalise: $w_k^i = t^{-1} w_k^i$
• END FOR
• Resample using algorithm 2:
  – $[\{\mathbf{x}_k^i, w_k^i, -\}_{i=1}^{N_s}] = \text{RESAMPLE}\ [\{\mathbf{x}_k^i, w_k^i\}_{i=1}^{N_s}]$
```

Figure 11.4: ALGORITHM 3: Sampling Importance Resampling

The advantages of this filter is that the importance weights are easily evaluated and the importance density can be easily sampled. But unfortunatelly it has also some disadvantages, like rapid loss of diversity in particles, because of resampling in each iteration.

11.5 Tracking People with particle filters

Tracking people is difficult. The first difficulty is that there is a great deal of state to a human: there are many joint angles, etc. that may need to be represented. The second difficulty is that it is currently very hard to find people in an image. This means that it can be hard to initiate tracks. Most systems come with a rich collection of constraints that must be true before they can be used. This is because people have a large number of degrees of freedom: bits of the body move around, we can change clothing, etc., which means it is quite difficult to predict appearance. People are typically modelled as a collection of body segments, connected with rigid transformations. These segments can be modelled as cylinders (Fig. 11.5), in which case, one can ignore the top and bottom of the cylinder and any variations in view, and represent the cylinder as an image rectangle of fixed size or as ellipsoids. The state of the tracker is then given by the rigid body transformations connecting these body segments (and perhaps, various velocities and accelerations associated with them).

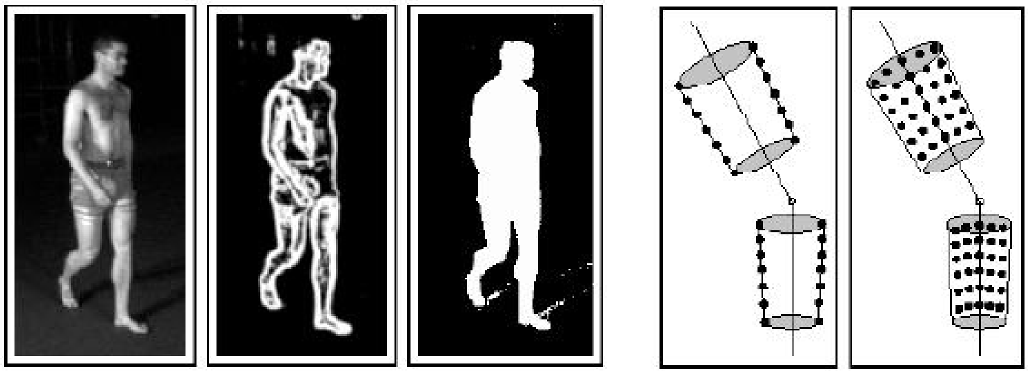


Figure 11.5: Tracking People with Particle Filters

Both particle filters and (variants of) Kalman filters [9] have been used to track people. Each approach can be made to succeed, but neither is particularly robust. There are two components to building a particle filter tracker: first, one needs a motion model and second, a likelihood model. One can use either a strong motion model, which can be obtained by attaching markers to a model and using them to measure the way the model's joint angles change as a function of time, or a weak motion model, perhaps a drift model. Strong motion models have some disadvantages: perhaps the individual that are tracked moves in a funny way; then a different models for walking are neccessary, walking carrying a weight, jogging and running (say). The difficulty with a weak motion model is that each frame is a poor guide to the next [5].

Likelihood models are another source of difficulties, because of the complexity of the relationship between the tracker's state and the image. The likelihood function tends to have many local extrema. This is because the likelihood function is evaluated by, in essence, rendering a person using the state of the tracker and then comparing this rendering to the image. Assume that one knows the configuration of the person in the previous image; to assess the likelihood of a particular configuration in the current image, it should be used the configuration to compute a correspondence between pixels in the current image and in the previous image. The simplest likelihood function can be obtained using the sum of squared differences between corresponding pixel values and this assumes that clothing is rigid with respect to the human body, that pixel values are independent given the configuration, and that there are no shading variations. These are all extremely dubious assumptions.

11.6 Advantages and disadvantages of particle filters

One of the greatest advantage of the paticle filters most probably is that they deals with a non-Gaussian noise. The ability to represent arbitrary densities is also one big step forward toward tracking different aims. This framework also allows including multiple methods, or in other words, one can achieve succesful tracking maneuvering targets. However, there are also some disadvantages, as mentioned above. The high computational complexity is a result of the great number of particles used. Moreover, it is very dificult to determine the optimal number of particles, since this number increases with the model's dimension. The problems arrising are already mentioned: degeneracy and loss of diversity. Also, one should be very careful by the choise of the importance density, since this step is the cruical one.

11.7 Conclusion and future work

In conclusion, the Particle Filter approach for tracking proposed in the paper has significantly improved in the last years among the research community. There is a lot of work to be done, improving the likelihood models used by tracking people od even by tracking of multiple interacting targets. Any invention might be helpful in these cases. In future work, we hope that the interacting multi-target tracking [10], [11] would be one of the most exciting topic where the particle filters could contrubute in some way.

Bibliography

[1] S. Arulampalam and B. Ristic, *Comparison of the Particle Filter with Range parametrised and Modified Polar EKF's for Angle-Only Tracking, Signal and data Processing of Small Targets.* 2000, SPIE Volume 4048, pp.288-299

[2] C.P. Robert and G. Casella, *Monte Carlo Statistical Methods (second edition).* 2004, New York: Springer-Verlag.

[3] M. West and J. Harrison , *Bayesian Forecasting and Dynamic Models.* 1997, Springer Series in Statistics, 2^{nd} Edition, Springer-Verlag, New York

[4] G. Kitagawa and W. Gersch, *Smoothness priors analysis of time series.* 1996, New York: Springer-Verlag.

[5] D. A. Forsyth, J. Ponce, *Tracking with Nonlinear Dynamic Models.* 2001, Prentice Hall,

[6] S. Julier, *A skewed Approach to filtering, Signal and Data Processing of small Targets.* 1998, SPIE Volume 3373, pp. 271-282

[7] E.A. Wan and R. Van der Merwe, *The Unscented Kalman Filter for Non-linear Estimation.* 200, Oct. Proceedings of Symposium 2001 on Adaptive Systems for Signal Processing, Communication and Control, Lake Louise, Alberta, Canada.

[8] E.A. Wan and R. Van der Merwe, *To appear in Kalman Filtering and Neural Networks.* 2001, Chapter 7: The Unscented Kalman Filter, Wiley Publishing

[9] Target Tracking, *http://www.ie.ncsu.edu/kay/msf/tracking.htm.*

[10] Z. Khan, T. Balch, and F. Dellaert, *An MCMC-based Particle Filter for Tracking Multiple Interacting Targets.* College of Computing,Georgia Institute of Technology Atlanta, GA, USA

[11] Schulz, D., Burgard, W., Fox, D., Cremers., A.B., *Tracking multiple moving targets with a mobile robot using particle filters and statistical data association.* 2001, In: IEEE Intl. Conf. on Robotics and Automation (ICRA).

[12] D.Fox, J. Hightower, L. Liao, D. Schulz, G. Borriello, *Bayesian Filters for location estimation.* University of Washington, Dept. of Computer Science and Engineering, Seattle, WA, Intel Research Seattle, Seattle, WA

[13] N. Gordon, D. Salmond and A. F. M. Smith, *Novel Approach to Non-linear and Non-Gaussian Bayesian State Estimation.* 1993,IEE Proceedings-F, Volume 140, pp.107-113

[14] G. Kitagawa, *Monte Carlo Filter and Smoother for Non-Gaussian Non-Linear Space State Models.* 1996, Journal of Computational and Graphical Statistics, Volume 5(1), pp1-25

[15] N. Bergman, *Recursive Bayesian Estimation: Navigation and Tracking Applications.* 1999, PhD Thesis, Linkoping University, Sweden

[16] J. S. Liu and R. Chen, *Monte Carlo Methods for Dynamical Systems.* 1998, Journal of the American Statistical Association, Volume 93, pp.1032-1044

[17] N. Bergman, A. Doucet and N. Gordon, *Optimal Estimation and Cramer-Rao Bounds for Partial Non-Gaussian State Space Models.* Ann. Inst. Statist. Math., 2001, Volume 53

[18] A. Doucet, *On Sequential Monte Carlo Methods for Bayesian Filtering* 1998, Technical Report, University of Cambridge, UK, Department of Engineering