EINDHOVEN UNIVERSITY OF TECHNOLOGY

A modular architecture for object tracking and migration in self-organizing, distributed systems

Citation for published version (APA):
Sijs, J., Baan, J., Meer, van der, S. M., & Kruithof, M. C. (2009). *A modular architecture for object tracking and migration in self-organizing, distributed systems*. https://doi.org/10.3182/20090924-3-IT-4005.00034

DOI:
10.3182/20090924-3-IT-4005.00034

Document status and date:
Published: 01/01/2009

Document Version:
Accepted manuscript including changes made at the peer-review stage

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 08. Jun. 2023

A modular architecture for object tracking and migration in self-organizing, distributed systems

J. Sijs * **S.M. v.d. Meer** * **M.C. Kruithof** * **J. Baan** *

* *TNO Science & Industry, 2600 AD Delft The Netherlands (e-mail: joris.sijs@tno.nl).*

Abstract: Object tracking in networked systems is a widely used scenario to assess newly designed methods in the field of association and estimation. This is mainly due to a great interest in object tracking applications. Current research is focused on designing dedicated tracking algorithms suitable for a particular setup of a tracking system. As a result certain choices in software design are taken implicitly, resulting in dedicated functions for which comparing and adapting the algorithm requires a significant effort. In contrast this paper describes a modular software-architecture suitable for all object tracking methods in self-organizing, networked systems. This modular architecture not only enables a designer the freedom of choice for the different tracking methodologies but it also provides the ability to re-upload parts of the algorithm or even adapt the on-line algorithm to the situation at hand. The designed architecture is demonstrated on a real-life tracking application.

Keywords: self-organizing systems, distributed systems, object tracking and object migration.

1. INTRODUCTION

The combination of multiple object tracking and networked systems, for example wireless sensor networks (WSNs), Akyildiz et al. (2002), is an active research area, Durant-Whyte et al. (1990); Songhwai et al. (2005, 2004); Bar-Shalom and Li (1995). The goal is to track all objects in a certain area using connected sensor-systems, or nodes, graphically depicted in Figure 1. Current solutions cover one of the three steps that are required during multiple object tracking: 1. Object detection; 2. Association of a detection to an object; 3. Estimation of the object's position (and speed), Bar-Shalom and Li (1995); Roth et al. (2008); Blackman and Popoli (1999); Songhwai et al. (2004); Karlsson and Gustafsson (2001); Songhwai et al. (2005); Poore and Gadaleta (2006); Ward et al. (2003).

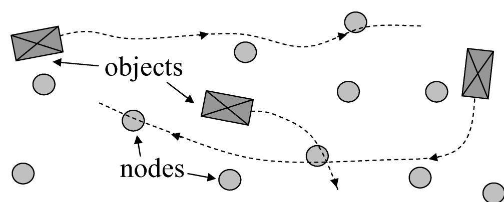

Fig. 1. Multiple objects manoeuvring through a set of nodes. which can detect the objects.

With the emergence of WSN new implementation issues related to tracking multiple objects using a networked system arise. However, current research is limited on estimating the object's position by fusing multiple sensors, Durant-Whyte et al. (1990); Songhwai et al. (2005, 2004); Bar-Shalom and Li (1995). As a consequence, another more fundamental issue, remained unanswered: the migration of tracked objects from one node to another. This fundamental issue arises from the fact that objects move through the network. Therefore all data corresponding to the object, for example the object's ID and position, should also migrate through the network. This is because all information of a tracked object should be located at a node which also observes the object. Present literature describes dedicated algorithms for object migration dependent on the application and situation at hand, Loy et al. (2002); Chong et al. (2000). As a result changes in communication or processing capacity cannot be coped with. In order to meet these dependability issues in networked system, a modular software architecture for the node's local tracking algorithm is proposed. The benefit of such an architecture is that every part of a local tracking algorithm can be rewritten and uploaded on-line. Therefore implementation of algorithms in the networked system consumes less time. Furthermore, adaptive algorithms are also feasible. Notice that this also allows implementing different tracking methods at different nodes.

2. PRELIMINARIES

In this paper we address the software architecture of a node rather than the network topology among the nodes or communication between them. As such, sending data to other nodes results in defining the identifiers of the receiving nodes, defined as *nodeID*. All sent data is collected in the data-set *out* while all received data is collected in the set *in*. Neighbors of a node are all those nodes which have a one-hop communication link with that specific node. Furthermore, because a node's tracking algorithm depends on local information such as the node's position, local surroundings, neighboring nodes, transformation matrices to world-coordinates etc, we define the data-set *static_data* in each node containing this information. This set can be used by any function in the node.

At each node we denote the set of locally tracked objects as *objects*. Within this set the jth object contains at least the following fields:

- $objects(j).objectID$: globally unique number representing the identifier of the object.

- $objects(j).track$: trajectory of the object, i.e. estimated positions at certain time instants, in world-coordinates.
- $objects(j).datenum$: time corresponding to each estimated position of the object's track.
- $objects(j).det$: the detections that were associated to the object with their corresponding position and time, i.e. $objects(j).det(i).position$ and $.det(i).datenum$.
- $objects(j).probability$: a value to indicate the probability that the tracked object is in fact an object.
- $objects(j).propagate$: information whether the object was sent to other nodes in the network.
- $objects(j).takeover$: information whether the object is tracked/detected at other nodes in the network.

Of course more fields can be added to this set if a particular tracking methodology is used. An example is the Kalman filter, Kalman (1960), for which fields such as the state-vector $objects(j).x$ and error-covariance matrix $objects(j).P$ could be added.

We assume that a parallel algorithm exists which is able to detect objects from the sensor signal, called "ObjectDetection". The data-set of current detections, i.e. between two sampling instances, is denoted with det. within this set the kth detection contains at least the following fields:

- $det(k).position$: position-vector of the detection in world-coordinates.
- $det(k).datenum$: time of the detection.

The detection-sets det_A and det_{NA} are subsets of det. det_A represents the detections that were associated to an object while det_{NA} represents the detections that were not associated. Both sets contain similar fields as det, although det_A has an additional field $det_A(k).objectID$ representing the object's identifier to which the detection was associated.

Figure 2 graphically depicts the setup of a node with its different data-sets and functionalities. Although "ObjectDetection" is an important part of the tracking system, this paper focusses on tracking and migration of objects performed by the function "LocalTracker".

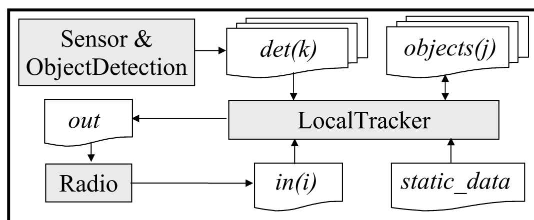

Fig. 2. The different components of a single node.

3. PROBLEM FORMULATION

Let us assume a self-organizing, networked system consisting of a large amount of nodes, all having an equivalent setup as shown in Figure 2. All objects within a node's local surrounding can be detected. A schematic top-view of a 2D situation is presented in Figure 3. The goal of this networked system is to track all objects that move within the observed area.

The implementation of a multiple object tracking algorithm in a self-organizing, networked system gives two more challenges compared to multiple object tracking in a central system. First, to track all objects in the network, each node must be able to track all objects in its local surroundings and propagate the ones that are of interest to neighboring nodes. Therefore all data corresponding to a single object, i.e. $objects(j)$, cannot be fixed at a node in the network but should be migrated from one node to another. Second, the algorithm should have the flexibility to be adapted when changes in communication/processing capacity, hardware or environmental circumstances occur.

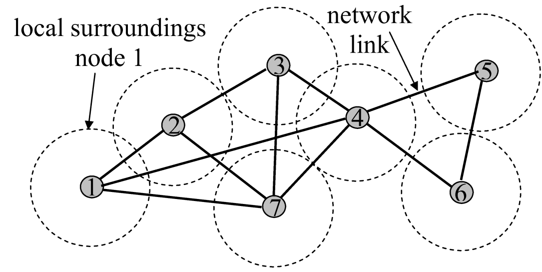

Fig. 3. Network topology and local surroundings of each node.

Current literature related to this topic aims on solving specific parts of the tracking problem. The topic on which most research focuses is sensor fusion applied to object tracking in networked systems, Mallick et al. (2004); Bar-Shalom and Li (1995); Blackman and Popoli (1999). However, these methods assume a given hardware setup and aim to find dedicated algorithms for object tracking. As a results a lot of design choices, for example association and data transfer, are implicit consequences of the hardware design and are therefore embedded in the resulting tracking algorithm. Other than dedicated descriptions, Loy et al. (2002) presents an adaptive block-scheme for object tracking, although it is based on one specific estimator and cannot be adapted to different tracking methods. Finally Chong et al. (2000) presents an architecture for association and estimation in multiple object tracking but it only describes the issues around object migration.

However, the typical challenges in self-organizing, networked systems such as replacing parts of the on-line tracking algorithm (due to improved methods or changes in resources) and adding/removing nodes to/from the network, demand an integral solution rather then a specific tracking method. To that extend this paper presents a modular software-architecture. This architecture is a blueprint of the different functionalities for any node's "LocalTracker" (Figure 2) to perform object tracking and migration. The flexibility in adaptation of the multiple object tracker is guaranteed as each function of the node's "LocalTracker" can be implemented differently. The architecture should be compatible with different tracking algorithms, for example deterministic or probabilistic ones and hierarchically, where a single object is tracked in one master-node, or fully distributed ones, where multiple nodes can track the same object.

4. MODULAR ARCHITECTURE FOR OBJECT TRACKING AND MIGRATION

We start with a basic tracking architecture of a single node without communication, i.e. stand alone. After a description of the stand-alone case, the functionalities due to communication with neighboring nodes are added.

4.1 Stand-alone node

A stand-alone node has no communication with any other node in the network. Therefore the node's tracking algorithm consists of the basic tracking functions such as described in Bar-Shalom and Li (1995); Roth et al. (2008); Blackman and Popoli (1999); Karlsson and Gustafsson (2001); Songhwai et al. (2005):

- NextSampleTime: Determines the next sample instant depending on current *time*, detections and objects.
- AssociateDetections: Identifies which detections correspond to which object, i.e. determine det_A and det_{NA}.
- Detections2Objects: Adds the detections of det_A to their corresponding object and initialize new objects based on det_{NA}.
- ProbabilityUpdate: Updates the probability of each object and determine which objects should be deleted (not tracked anymore).
- TrackUpdate: Updates the track and/or state of each object.

Figure 4 presents a modular software-architecture for local object tracking in a stand-alone node. In this architecture the function “Associate” combines “AssociateDetections” and “Detections2Objects” while “Update” combines “ProbabilityUpdate” and “TrackUpdate”.

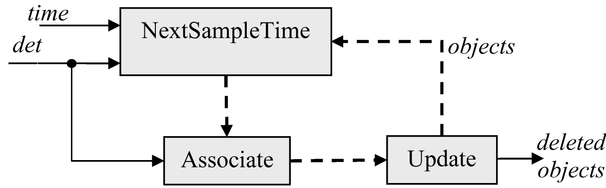

Fig. 4. Architecture of a node's tracking algorithm in case of no communication. The dashed line represents *objects*.

In the following section we discuss the changes in the software's functional architecture when communication is added.

4.2 Communicating nodes

To keep the architecture as general as possible, no initial communication restrictions on the network's capacity are assumed.

A node can use communication for two purposes. First to improve the estimation of an object's track by fusing a node's local detections and/or estimated tracks with those of neighboring nodes. Second to guarantee correct object migration through the networked system by propagating objects to neighboring nodes. These two purposes require additional functionalities.

Fusion of detections/tracks Estimated tracks can be improved by fusing a node's local detections and/or object tracks with those of neighboring nodes. As such (processed) detections and tracks need to be sent by each node to the neighboring nodes. To enable this functionality, let us first define the data-set $fuse_out$ with the following fields:

- $fuse_out.det$: Data of a node's local detections, for example based on det or $objects(j).det$, added with the field $fuse_out.det(k).nodeID$ representing the identifier(s) of the receiving node(s).
- $fuse_out.track$: Data of a node's local objects, based on $objects$, added with the field $fuse_out.track(k).nodeID$ representing the identifier(s) of the receiving node(s).

Depending on the application and methods, some of the fields of $fuse_out$ can be empty. Notice that some methods require that detections are sent right after they were produced. This means that we should distinguish between processed detections sent after each sample instant and raw detections sent independent of sampling instants. Furthermore, as a node sends $fuse_out$ to its neighboring nodes, it will also receive a similar set of data, denoted with $fuse_in$, from its neighbors. This set needs to be processed before it can be used to improve the estimation. Therefore the following functions are introduced:

- SendRawDetections: Adds each new detection of det to $fuse_out.det$.
- SendProcessedDetections: Builds both $fuse_out.det$ as well as $fuse_out.object$, based on det and $objects$, at each sample instant.
- ReceiveFuse: Stacks the different detections of $fuse_in.det$ into one local set of received detections, denoted with r_det. The same holds for $fuse_in.track$ and r_track.
- DetecFusion: Fuses the local detections, i.e. det, with the received ones, i.e. r_det, to determine the fused detection set f_det.
- TrackFusion: Fuses the local tracks (states) with the received ones, i.e. r_track, to result in one estimated track of an object.

The resulting architecture is presented in Figures 5 and 6.

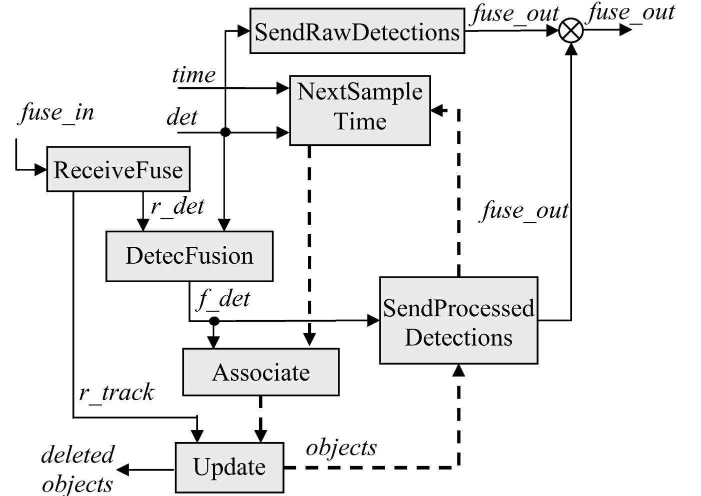

Fig. 5. Architecture of a node's tracking algorithm capable of fusing detections and/or tracks. The dashed line represents *objects*.

Migration of objects: Communication also permits object migration through the network. Before designing the additional functions and adding them to the architecture of Figure 5, let us first point out that a node should be able to perform two important tasks:

(1) When should it propagate object-data and to which neighbors.
(2) Based on what information and criteria should it remove an object.

In order for the node to perform the first task, the set $propagate_objects$ is defined as the set of objects that are to be sent to neighboring nodes. In case only one local object, for example $object(k)$, is to be propagated to a neighboring node,

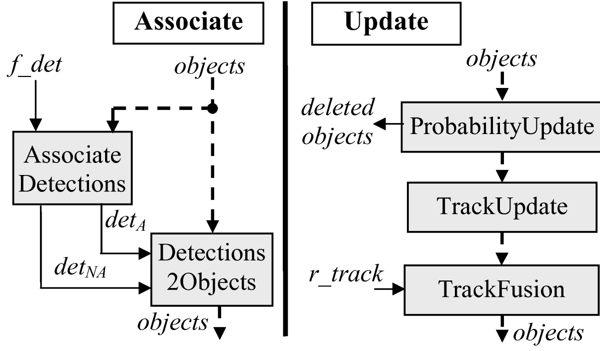

Fig. 6. Architecture of the functions "Associate" and "Update". The dashed line represents *objects*.

then $propagate_objects(1)$ is a copy of $objects(k)$. Another field $propagate_objects(1).nodeID$ should be added representing the identifier(s) of the receiving node(s). It is important to determine the criteria, based on *objects* combined with the node's *static_data*, when to propagate an object. In order to be able to implement such criteria in the "LocalTracker", the following two functions are defined:

- When2Propagate: Checks which *objects* fit the criteria for propagation to neighboring nodes. In case $objects(j)$ fits the criteria, then this function copies $objects(j)$ to the set *propagate_objects*, adds the field of the receiving node ($propagate_objects(k).nodeID$) and updates the field $objects(j).propagate$.
- AddObjects: Adds received objects of *propagate_objects* properly to a node's local set *objects*.

In order for the node to perform the second task, i.e. when to remove an object, it can receive information from neighboring nodes. An example could be to notify a neighboring node that the object which it propagated is currently detected. Notice that removing an object is part of the function "ProbabilityUpdate", where such information from a neighboring node should be available. As such the object-set *object_takeover* is defined which can be sent by a node to its neighbors in order to assist them in the decision of removing an object. Each element in this set, denoted with $object_takeover(j)$, corresponds to an object which was first propagated to the node and for example is currently tracked or detected by it. As such, each element $object_takeover(j)$ consists of at least the following fields:

- $object_takeover(j).objectID$: the identifier of the corresponding object.
- $object_takeover(j).nodeID$: the identifier(s) of the receiving node(s).

Furthermore, a node could also check the received set *fuse_in* on whether one of its propagated objects was detected at a neighboring node. Therefore in order to be able to implement such assistance in the "LocalTracker", the following two functions are defined:

- Help@ObjectMigration: Checks which local *objects*, that fit certain criteria, contain useful information for the neighboring node that propagated the object and adds this information in *object_takeover*.
- SuccesfullObjectMigration: Updates certain fields of local *objects*, for example $objects(j).takeover$, using the received sets *object_takeover* and *fuse_in*. These updated fields are then to be used in "ProbabilityUpdate".

Figures 7 and 8 show the final architecture of the function "LocalTracker". This modular architecture is capable of local object tracking and migration and enables uploading and adapting parts of the on-line tracking algorithm.

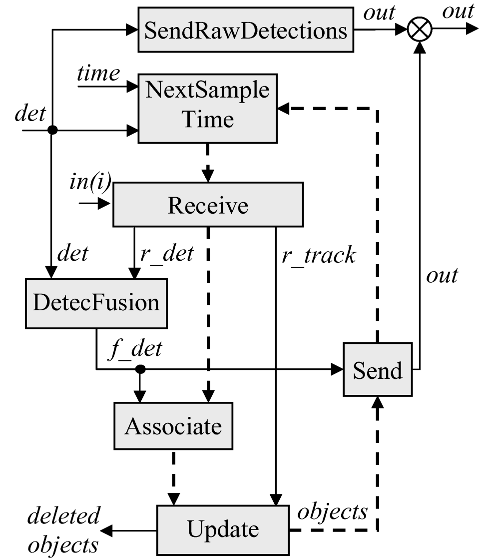

Fig. 7. A node's architecture for object tracking & migration. The dashed line represents *objects*.

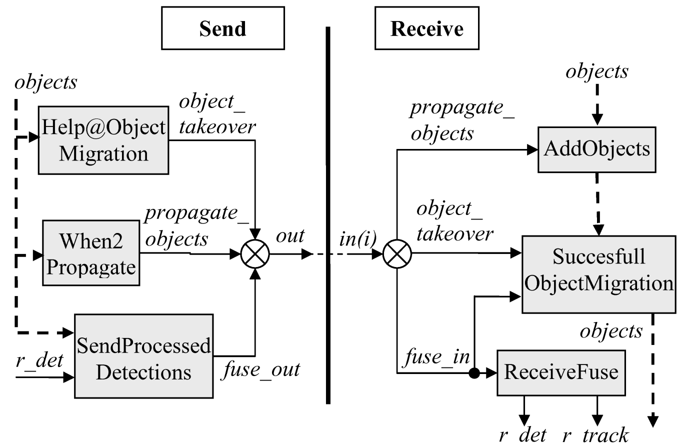

Fig. 8. Architecture for the functions "Send" and "Receive". The dashed line represents *objects*.

The designed modular software-architecture is demonstrated in a real-life application.

5. TRACKING APPLICATION

The goal of the real-life application is to track all vehicles on a part of the highway. The system consists of three cameras. We define the position of the first camera as $(0,0)$ and the x-direction parallel to the road. The second and third camera are positioned at $(106,0)$ and $(306,0)$ respectively. An overview of the system's setup is presented in Figure 9. The camera-images are used as input to the function "ObjectDetection" which is capable of detecting vehicles. The positions of each detection, together with its corresponding detection-time and covariance-matrix (uncertainty), define the fields of the set *det*.

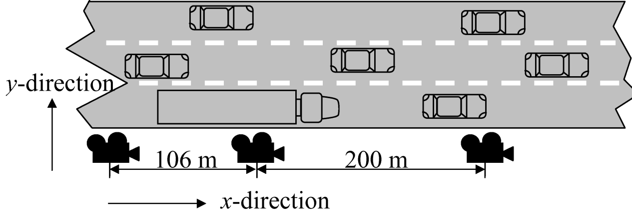

Fig. 9. Setup of the vehicle tracking system with cameras.

Three different tracking algorithms of the "LocalTracker" were implemented to show the diversity of the designed architecture; Individual Tracking, Distributed Tracking and Central Tracking. Let us first present Individual Tracking, which forms the basis of the other trackers. In Individual Tracking a camera doesn't communicate with any other camera due to which "SendRawDetections", "Receive" and "Send" are empty functions. The other functions are:

- NextSampleTime: Equal to the time of the next detection, either a local or a received one.
- DetecFusion: Adds the received detections r_det to the local ones, i.e. det, to result in the set f_det.
- AssociateDetections: Uses Gating and Nearest Neighbor (Blackman and Popoli (1999); Bar-Shalom and Li (1995)) to associate the detections of f_det.
- Detections2Objects: Adds the associated detections det_A to the detection field of the objects, i.e. $objects(j).det$ and clusters the non-associated detections det_{NA} to define new objects.
- ProbabilityUpdate: In case a new detection was associated to the object, the object's probability is added with value dependent on the equivalence between the predicted position and the position of the detection. If no detection was associated, the object's probability is decreased with a constant. The object's probability is bounded between 0 and 1. An object is deleted if its probability equals 0 and it is not detected for the time-period t_d $[s]$.
- TrackUpdate: Is a state-estimator equal to the Kalman filter Kalman (1960); Durant-Whyte et al. (1990)).
- TrackFusion: Remains empty as no tracks are sent.

Figure 10 shows the tracking results of Individual Tracking. In this figure time is plotted versus the object's x-position. Each track is represented by a solid line and a symbol. The solid line represent the estimated track while the symbol represents the camera which estimated the corresponding track; $\circ$ $*$ or $\diamond$ for a track of either camera-node 1, 2 or 3 respectively. Notice that vehicles drive in the positive x-direction. Therefore the slope of each track in this t-x-plot corresponds to the speed of the vehicle.

Notice that Figure 10 shows that an object, in case of Individual Tracking, does not migrate through the network as a track of a single object consists of multiple symbols. In contrast, Distributed Tracking does allow object migration, for which the "LocalTracker" propagates an object to a neighboring camera if the object crosses a line L_i; At $L_1 = 90[m]$ camera 1 sends the object to camera 2 while camera 2 sends each object which crosses $L_2 = 280[m]$ to camera 3. All detections that correspond to a propagated object are also sent to that neighbor. As such each camera's algorithm of the "LocalTracker" is defined similar the one of Individual Tracking, except for the following non-empty functions:

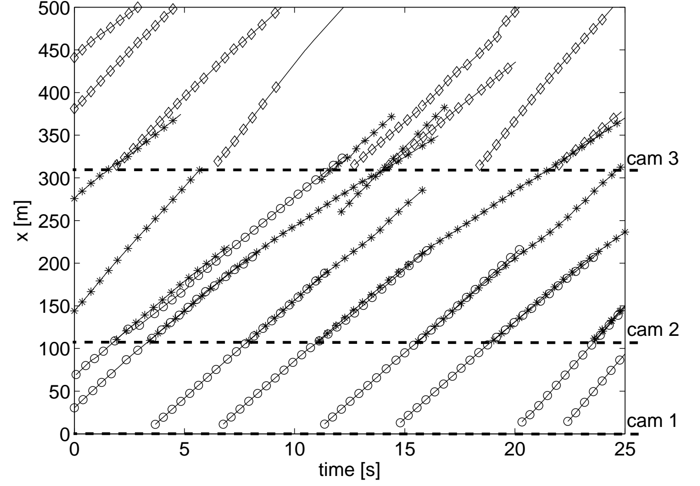

Fig. 10. The object-tracks of Individual Tracking ($\circ$ = cam 1, $*$ = cam 2, $\diamond$ = cam 3).

- Help@ObjectMigration: If an object was propagated from a neighboring node and it is detected in the local camera-image, then send its object ID to the camera from which it was propagated.
- When2Propagate: If the jth object, i.e. $objects(j)$, crosses L_i. Then copy $objects(j)$ to the set $propagate_objects$, sent it to camera $i+1$ and set $objects(j).propagate = 1$.
- SendProcessedDetections: If $objects(j).propagate = 1$, then sent newly associated detection to the neighboring camera by copying $objects(j).det$ to $fuse_out.det$.
- AddObject: Adds the received objects contained in the set $propagate_objects$ to the local set $objects$.
- SuccesfullObjectMigration: In case $objects(j).ID$ equals a received ID, it indicates that a neighboring node detects the object. As such set $objects(j).takeover = 1$.
- ReceiveFuse: Copies the received detections of $fuse_in.det$ to the set r_det.
- ProbabilityUpdate: Similar to the one defined in Individual Tracking extended with the criteria that in order to delete an object $objects(j).propagate$ and $.takeover$ need to be both 0 or 1.

In order to show the flexibility of this architecture with respect to an on-line adaptation of the "LocalTracker", we will use the same data-set as Individual Tracking. However, at 15 seconds we will re-define some functions to enable Distributed Tracking. Figure 11 presents the results of this test. Again the estimated tracks are represented by the solid line and the symbol represents the camera of the corresponding track. However, in case an object migrated from camera 1 to 2 its symbol changes from $\circ$ to $*$. Similarly the symbol of objects which migrates from camera 2 to 3 changes from $*$ to $\diamond$

Figure 11 shows that objects that are tracked at 15 $[s]$ or later and cross(ed) L_1 and/or L_2 are sent to the neighboring camera. As such correct migration of object through the network of cameras is shown.

Finally Central Tracking is implemented on this application to show that different nodes can have a different "LocalTracker". The basic idea is that a central-node performs the object tracking based on all detections of all cameras. To implement the Central Tracker we add an extra central-processor-node to the system. All functions of the "LocalTracker" in a camera are

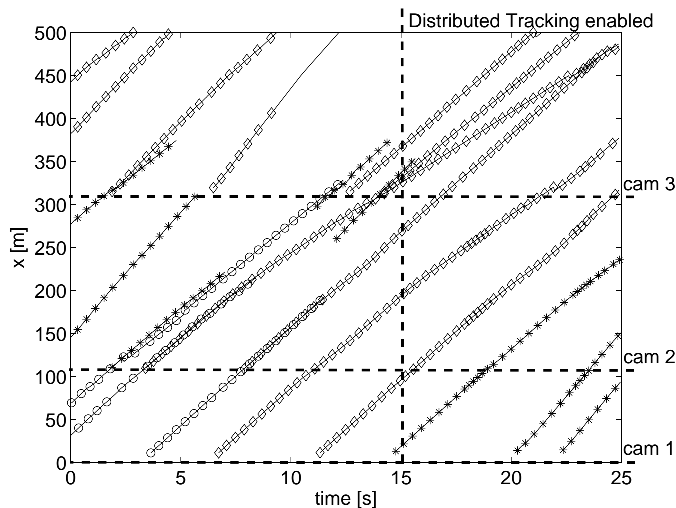

Fig. 11. The object-tracks of Distributed Tracking (○ = cam 1, $*$ = cam 2, ◇ = cam 3).

empty except "SendRawDetections", which is defined to sent all its local detections to the central-processor. The "LocalTracker" of the central-processor receives all these detections. As such the algorithm of the "LocalTracker" in the central-processor is similar to the one defined in Individual Tracking added with the non-empty function:

- ReceiveFuse: Copies the received detections of $fuse_in.det$ to the set r_det.

Figure 12 presents the result of Central Tracking in which the same data is used as in Individual Tracking. Due to the fact that only one node is able to track objects, i.e. the central processor, no symbols are required because all tracks correspond to the central processor.

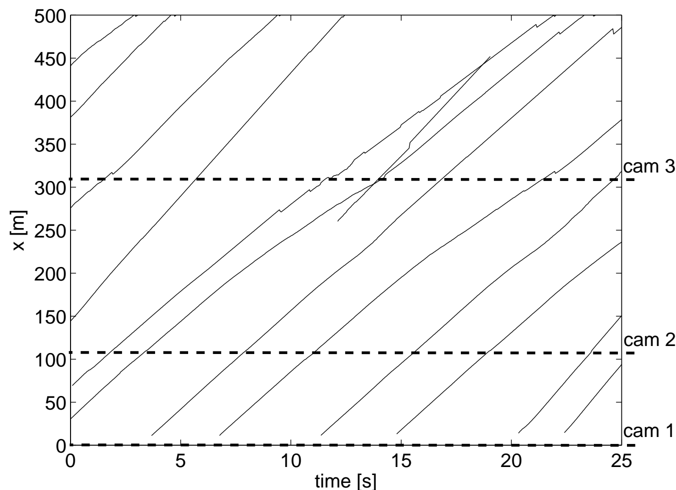

Fig. 12. The object-tracks of the Central Tracking.

6. CONCLUSIONS

This paper presents a modular software-architecture suitable for object tracking and migration. Object tracking deals with local tracking of an object in a single node based on local and received information. Object migration deals with the fact that objects physically move through the network of nodes. Therefore its estimated track should migrate through the network of nodes. This general architecture gives a designer tools to compare different algorithms and implement different trackers in the system with less effort. Moreover, the modularity also allows tracking applications in self-organizing networks. This is because the algorithm can be adapted on-line, depending on its situation, resources and newly available tracking methods. To show the architecture's flexibility and adaptability three different tracking methods were implemented and demonstrated on a real-life application.

REFERENCES

Akyildiz, I., Su, W., Sankarasubramaniam, Y., and Cayirci, E. (2002). Wireless sensor networks: a survey. *Computer Networks*, 38, 393–422.

Bar-Shalom, Y. and Li, R. (1995). *Multitarget-Multisensor Tracking: Principles and Techniques*. YBS.

Blackman, S. and Popoli, R. (1999). *Design and Analysis of Modern Tracking Systems*. Artech House, Norwood, MA.

Chong, C., Mori, S., Barker, W., and Chang, K. (2000). Architectures and algorithms for track association and fusion. *IEEE Aerospace and Electronic Systems Magazine*, 15(1), 5–13.

Durant-Whyte, H., Rao, B., and Hu, H. (1990). Towards a fully decentralized architecture for multi-sensor data fusion. In *1990 IEEE Int. Conf. on Robotics and Automation*, 1331–1336. Cincinnati, Ohio, USA.

Kalman, R. (1960). A new approach to linear filtering and prediction problems. *Transaction of the ASME Journal of Basic Engineering*, 82(D), 35–42.

Karlsson, R. and Gustafsson, F. (2001). Monte Carlo data association for multiple target tracking. In *IEE International Seminar on Target Tracking: Algorithms and Applications*.

Loy, G., Fletcher, L., and Zelinski, A. (2002). An Adaptive Fusion Architecture for Target Tracking. In *5th International Conference on Automatic Face and Gesture Recognition*. Washington DC, USA.

Mallick, M., Pao, L., and Chang, K. (2004). Multiple Hypotheses Tracking Based Distributed Fusion Using Decorrelated Pseudo Measurement Sequence. In *2004 American Control Conference*. Massachusetts, Boston, USA.

Poore, A. and Gadaleta, S. (2006). Some assignment problems arising from multiple target tracking. *Mathematical and Computer Modelling*, 43, 1074–1091.

Roth, D., Koller-Maier, E., Rowe, D., Moeslund, T., and Gool, L. (2008). Event-Based Tracking Evaluation Metric. In *IEEE Workshop on Motion and Video Computing (WMVC)*. Copper Mountain, Colorado, USA.

Songhwai, O., Sastry, S., and Schenato, L. (2005). A Hierarchical Multiple-Target Tracking Algorithm for Sensor Networks. In *Proc. of the 2005 IEEE International Conference on Robotics and Automation*. Barcelona, Spain.

Songhwai, O., Stuart, R., and Sastry, S. (2004). Markov chain Monte Carlo data association for general multi-target tracking. In *Proc. of the 2004 IEEE Conference on Decision and Control*. Paradise Island, Bahamas.

Ward, D., Lehmann, E., and Williamson, R. (2003). Particle Filtering Algorithms for Tracking an Acoustic Source in an Reverberant Environment. *IEEE Transactions on Speech and Audio Processing*, 11(6), 826–836.