Distributed Contract Networks of Sensor Agents with Adaptive Reconfiguration: Modeling, Simulation, Implementation and Experiments

Alois C. Knoll

Faculty of Technology, University of Bielefeld, D-33501 Bielefeld, Germany

ABSTRACT

As both sensors and communication technologies are becoming more powerful at decreasing costs, sensor networks are constantly growing in complexity, i.e. in the information they generate and in the need for coordinating network resources. In this paper we comparatively explore the performance of multi-level hierarchical networks and flat networks, i.e. arrays of sensor agents with specific competences but without predefined communication or control structures. We develop a model of the performance of these types of networks for analysing the throughput of sensing tasks depending on a variety of network and sensor parameters. This analytical model serves as the basis for numerical simulation runs, which reveal the specific advantages of the different network structures. Motivated by these results, we develop an adaptive control scheme, which enables the network to organise itself at run-time so as to achieve the highest result precision in minimum time with minimum communication overhead, maximum parallelism and maximum fault tolerance. This control scheme is a computer-operational version of the contract network, originally proposed for problem solving in Distributed Artificial Intelligence, with agents teaming up dynamically after a negotiation phase following a task specification. The metaphor of contracting also holds the potential to integrate agents that control actuators, e.g. cooperating robots, by assigning appropriate competences to them. Hence the design of sensorimotor systems, whose behaviour is adaptively controlled in a situation-dependent way by a team of different sensors, is a straightforward extension. Moreover, it is also easy to extend the network by adding agents that do not have a sensing component and can only perform sensor data processing tasks. We introduce a complete software framework for implementing these kinds of networks. It consists of a formal specification of the requirements that the sensory component of an agent has to meet and a comprehensive library of C++ objects encapsulating all of the negotiation protocol and communications routines. In the experimental section of the paper we show how several uncalibrated cameras are used to estimate the trajectory of a manipulator in order to guide the manipulator towards a target in real time. We also illustrate how agent-teams may easily reconfigure during task execution in the case of unexpected events. We then go on to show how complex vision tasks can be broken down into tasks that identical agents may work on in parallel. Finally we show how fusion for vision at a symbolic, i.e. object description level, may contribute to solve the hard problem of recognising occluded objects from different view points. We conclude the paper by sumarising our results and sketching future research, in particular the automatic mutual reprogramming of agents using a specifically developed ontology.

Keywords: multi-agent contract network, sensor-fusion, fault tolerance, task adaptation, uncalibrated visual servoing, distributed vision systems

1 INTRODUCTION

The rapid growth of computing power, communication bandwidth and reliability at low costs, along with advances in sensor technologies and data fusion methodology [1–6] provide a sound basis for the design of large scale, spatially distributed,

E-mail: knoll@techfak.uni-bielefeld.de

adaptive and reconfigurable networks of sensors for a diverse range of tasks. The implementation of such systems would have been hardly conceivable just a few years ago. Despite the massive work that has been spent on both distributed and multi-agent systems (e.g. for Internet applications and in the field of distributed AI), there are still only very few systems available that are specifically tailored to the needs of distributed sensor systems. Such a system would be characterised by the following key properties:

- **Simplicity and Scalability:** The interface between the physical (non-standard) sensor and the software system should be easy to implement and the sensor operation should be transparent, i.e. easy to monitor. The interface should be unaffected by the size of the system, i.e. the total number of sensors in the network. The system performance should scale up in a predictible manner when sensor are added.

- **Fault Tolerance:** If an individual sensor is detected as being temporarily or permanently defective, it should be automatically phased out of the normal system operation and, if possible, replaced with one or more alternative sensors. In this case the system's performance should degrade disgracefully, it should never come to a complete stop.

- **Task Adaptation:** Complex sensors with sophisticated preprocessing facilities may produce a variety of qualitatively different results, e.g. a "camera agent" may provide gray-scale images but it may also recognise faces if so instructed. Moreover, the combination of such agents in a multi-sensor system may result in completely new services, e.g. the generation of range images through three or more camera agents. The multi-agent system should make it easy to implement and to give access to access all of these services through flexible task specifications (not necessarily completely defined by the system's designer but changing over time). It is also desirable that agent (or teams of agents) automatically adapt to environmental changes.

- **Self-Organisation:** The ability of the system to form agent teams and the tuning of parameters controlling the task processing should be formulated on an abstract level during the design process of the system. At run-time, team formation should be completely automatic, i.e. only dependent on the tasks to be performed (and on the state of the environment). No intervention on the part of the programmer/operator should be necessary.

- **Heterogeneity and Integration:** The multi agent system should allow for the combination of sensors of different principles of operation. It should also be possible to include actuators so as to build combined sensor/actuator systems that close the sensorimotor control loop, particularly desirable are multi-sensor/multi-actuator systems. Obviously this requires communication modes that guarantee real-time interaction between sensors and actuators.

- **Standardisation:** Established standards, e.g. for distributed communication and operating systems, should be adhered to as closely as possible.

Only a limited number of approaches to provide frameworks for designing reconfigurable multi-agent systems along the above lines for sensor coordination and sensor/actuator integration have been published in literature and only a fraction of them have been implemented in real world applications. Issues of research reported in recent literature include reconfiguration in homogeneous networks of sensors for multi-vehicle control([7] and, to some degree, [8]); sensor selection(e.g. [9]); network architecture and communication between sensors [10]; sensor configuration under certain tasks(e.g. [11]) and the fundamental question of fusion of information from individual sensors ([12,4,13]). When designing a multisensor system, it is very useful

to abstract from the physical sensor. This abstraction was the focus of early work by Henderson [14], an application of this modelling approach can be found in [15]. This was, however, only a framework for structuring the design process (by compiling specifications into a target language); there were hardly any software tools available for guiding the design process, for making multi-sensor networks run in different setups, for evaluating the system performance based on sensor models, for monitoring or for debugging. In [16], a more advanced scheme was published, which enables the designer to experiment with, simulate, debug and to continuously monitor the performance of different versions of a sensor system. We also refer the reader to [17], which summarises much of the work on fusion in decentralised sensor networks that was started by Durrant-Whyte in the early nineties [18] aiming at improving the navigation of mobile robots. These approaches and methodologies target mostly static sensor networks, i.e. they do not provide means for task-adaptive automatic configuration at run-time, in particular the processing of different classes of tasks requiring completely different strategies for sensing (and/or sensor placement).

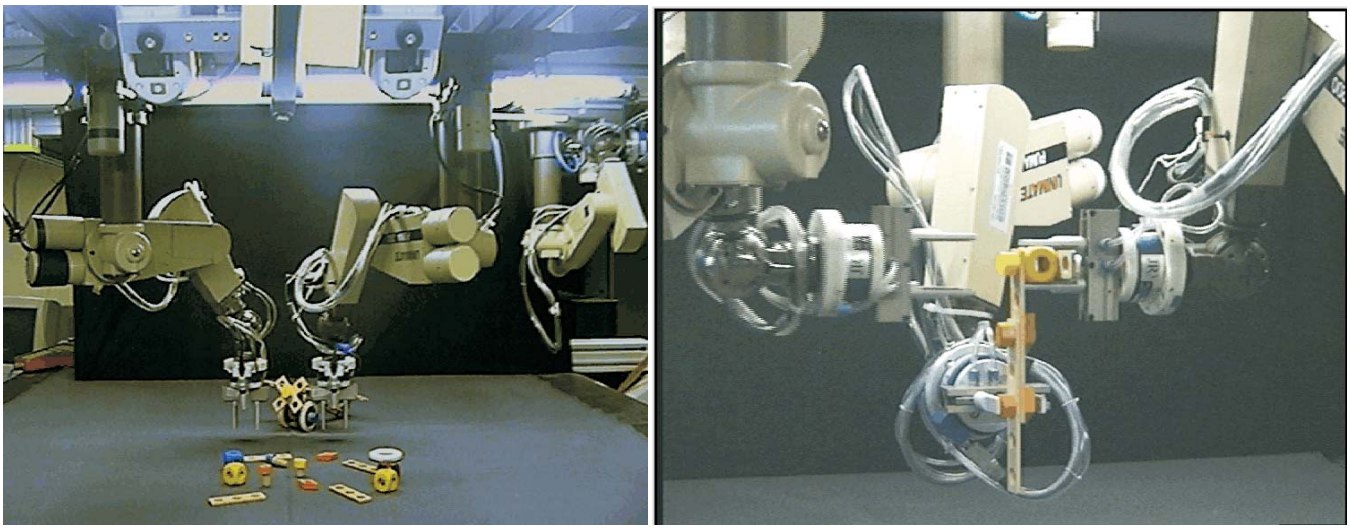


Figure 1. Complex multi-sensor/multi-robot setup

Our work in the field of adaptive and reconfigurable sensor/actuator networks started with a theoretical interest in comparing different network architectures with respect to task throughput. We were also interested in designing a computer operational implementation of the contract network for organising the competition and cooperation between sensor agents based on the metaphor of contract negotiation between experts as outlined in [19,20]. As part of a large scale research initative aiming at designing a "situated artificial communicator" (SAC), we developed an assembly cell integrating a large number of sensors and actuators (see [21] for an early description of this cell). The SAC is a robot system capable of building complex toys (wooden "airplanes" from elements like nuts, bolts, slats, wheels, etc.) solely based on multimodal interaction with a non-expert human instructor. The building elements are placed on a table in arbitrary position and "stacking order" without any fixtures; they must be recognised by a sensor system, and they must also be tracked to maintain permanent reference between utterances of the instructor and the internal world model. Part of the assembly cell is shown in Fig. 1. It is equipped with up to six robots, one of which mounted on a translational slide. The multi-sensor system of the cell consists of six articulated cameras for active vision, a stereo camera rig, miniature cameras on the robot hands with optional laser-plane projectors and force/torque sensors in the robots' wrists. This diverse range of complex sensors, the multitude of (sub-)tasks to be performed at different time-scales (e.g. sensor placement; detection, recognition, localisation, tracking of objects using monocular, binocular, trinocular

techniques with articulated or fixed sensors; 3D scene reconstruction, visual servoing, ...), the need to integrate actuators into the sensing process (for placing sensors), the unpredictable environment, the need to interface standard third-party software systems, and the great number of interconnected processors make this setup an ideal testbed to evaluate distributed fault-tolerant and task-adaptive sensor networks. Results obtained in this complex setting may easily be transferred to standard scenarios like decentralised tracking of aircraft, etc. We shall return to this setup in the experimental sections of this paper. In the next three sections, the network structures we investigated, the performance characteristics we obtained through extensive simulations and the software framework we developed for implementing adaptive reconfigurable networks in a straightforward manner will be introduced.

2 NETWORK ARCHITECTURE

) The basic entity of the network is the "sensor agent", which combines the physical sensor, preprocessing functionalities, services offered to other agents, local memory and communication facilities into one "black box". We differentiate between agents that include a sensing component (physical sensor) together with the communication component and agents that provide only this communication component. In other words: coordination agents, if present in the system, provide services like task coordination or processing power to the former. This definition of sensor agent classes includes the other interesting mappings between the physical sensor and its processors: (i) the $1 \rightarrow K$ relation, where one physical sensor provides information for K more or less specialised agents. An example would be a team of agents that cooperate on the segmentation of image regions (e.g. [13]), and (ii) the $K \rightarrow 1$ relation, which is the classical hierarchical network in which K sensor agents are controlled by one superior agent. Clearly, a mix of them is also possible, this would result in a $K \rightarrow M$ relation, where K agents at a lower level interact with M agents at a higher level of a hierarchy. We assume that the network receives a sensing task from an *external mandator*. Depending on the competences assigned to them, the sensor agents then agree to form a team to work on the task, either sequentially or in parallel.

This setting suggests a comparison with a human team of experts and it is therefore helpful to briefly recall models from organisation theory. A huge amount of work has been done in this field, particularly for dealing with uncertainty introduced when only partial information is accessible to every node (see, for example, [**?**]). Basically, there are two main issues to be dealt with when organising teams of interacting agents: (i) the structure of the team and (ii) the control mechanism for coordinating the agents. Criteria for defining (i) and (ii) are both the problem complexity (e.g. the arrival rate of tasks, the amount of knowledge necessary for resolving the problem and for coordinating the a priori knowledge and the resources) and the uncertainty (about the precision of the acquired data, about the behaviour of the sensor agents and about the behaviour of the environment).

2.1 Team structure

The team structure is specified by defining the capabilities of the team members and by assigning them competences and responsibilities. An obvious choice for these assignments is the simple hierarchy in Fig. 2(a), where the agents at the lower level are all specialising in unique classes of tasks. In example from human organisations, the mandator M would be an executive officer who wants a report to be printed and bound and who controls every step in this process by successively having it typed (A_1), copied (A_2), and so on. In the context of our sensor agents this implies that certain agents may specialise in particular tasks such as the sensing of the physical data; others work on different problems (e.g. preprocessing data from

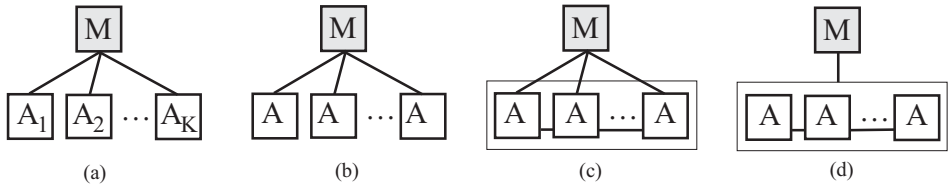


Figure 2. Different forms of two-level hierarchies. Explanations see text.

different channels, establishing communication paths or coordinating subordinated agents). A different form of hierarchy is shown in Fig. 2(b), where a team of non-specialised agents works on and returns complete solutions to the upper level agent (e.g. a pool of typists are given tasks by the upper level agent according to their current workload). This would correspond to several instantiations of the same agent type (with equal competences and duties) that can handle tasks depending on their current load. In an *extended hierarchy* (such as proposed in [10]) there may be more than one level of coordinating agents. Agents at the lower level may communicate with each other (e.g. for load compensation), but they are still identifiable by the upper level agents, Fig. 2(c). Specialised agents may coexist with non-specialised agents in one network. However, if there are non-specialised agents on the same level, then there is a potential for these agents to coordinate themselves by interchanging information directly without any arbitration by a superior agent, see Fig. 2(d). This is the concept of lateral structures. Both hierarchical and lateral structures may coexist in one network; subtrees are structured laterally and organise their cooperation within their layer of the subtree autonomously after receiving a certain task from their superior agent (or the external mandator M). In such organisations there is no coordinating authority, and agents may be transient members of different teams.

2.2 Control mechanisms

The control mechanism defines how and when sensor agents communicate (interact). From an interaction, a transfer of control may result, which may be preceded by a selection process. The mechanism for coordinating communication between the agents may be either static, i.e. communication channels and hence groups of sensors for working on a certain task are fixed (e.g. [12]), or it may be dynamic. The latter means that cooperation between sensor agents is agreed upon for a limited period of time and vanishes after completion of the task. During the selection process, an exchange of information with a number of different agents may take place. The decision for or against the cooperation with a potential partner may then be taken after evaluating the latter's offer in terms of promised result quality, e.g. time of completion and measurement precision. An example of a static control mechanism is the *conservative selection strategy*: An agent that initiates a cooperation for a certain class of tasks for the first time looks for suitable partners and (possibly randomly) selects one of them. When the same task (or class of tasks) is to be performed again, the agent selects the same partner(s). After some time, all classes of tasks have caused each agent to "know" each partner for every task class and the partnerships for cooperation are fixed. With a dynamic strategy, current partnerships do not affect future relations. The selection process is repeated each time a cooperation becomes necessary, and the momentary state of potential partners (e.g. workload) may be taken into account. Obviously, the selection strategy may have a drastic effect on the performance of the network. Sophisticated dynamic strategies entail more communication overhead, but they lend themselves to be used in lateral networks in which agents are little specialised. Moreover, the effects of sensor failure are less severe, and the addition or removal of agents does not necessitate a complete reinitialisation of the network. In hierarchical networks (or in lateral networks with highly specialisd agents) where there is only a limited choice of partners, static strategies normally have a clear advantage.

The basis for our comparative simulations of team formation, fault tolerance and reconfiguration were the two structures shown in Fig. 3. In the case of a "flat" base structure (corresponding to Fig. 2(d)), it is assumed that the network forms a grid of K nodes (i.e. sensor agents that may sensibly cooperate on solving the task, e.g. because they share a common field of view), see Fig. 3, left. A sensor agent is composed of a coordination component cc responsible for establishing links and transferring both control information and sensor data to other sensor agents together with an optional sensing component sc in charge of both data acquisition and data (pre-)processing. The sensing component and coordination component of a sensor agent communicate by means of a local data base, which stores information related to the tasks the sensor agent is allocated to. This notion of (overlapping) teams forms the basis of our work we introduce in section 4 because apart from the lateral structure with full reachability it contains all hierarchical structures as a subset of possible specialisations (through the assignment of limited capabilities/competences to each individual agent). For the hierarchical organisation, two additional layers of sensor

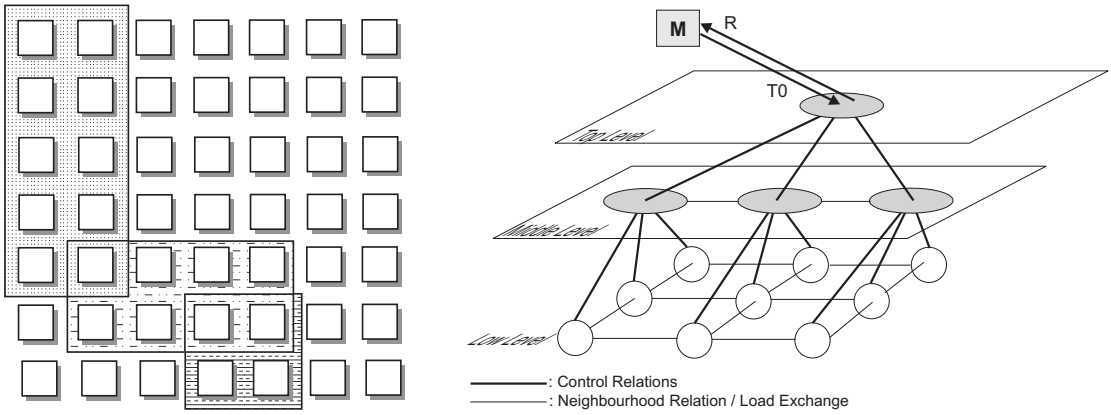


Figure 3. Flat and hierarchical structure of sensor agents. The shaded areas in the left subfigure indicate different teams cooperating on individual tasks for a limited period of time.

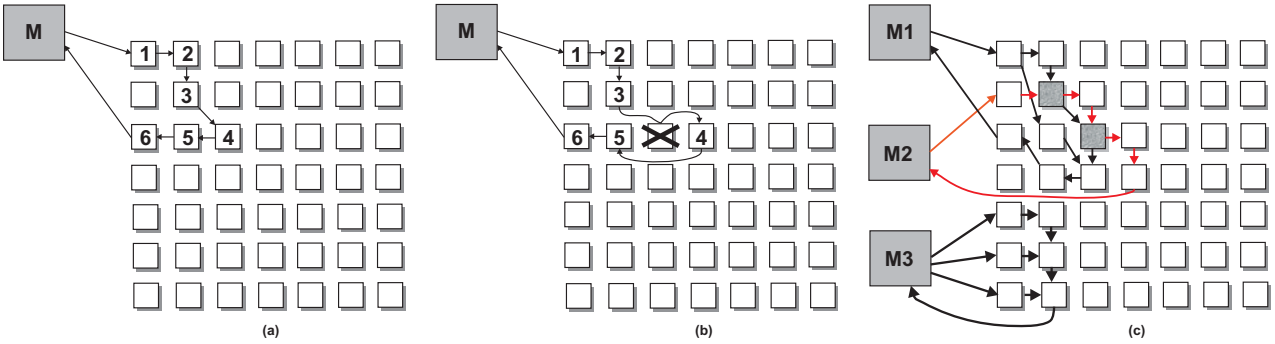


Figure 4. Different types of team formation in the flat structure of Fig. 3.

agents (manager-agents) were introduced with the original sensor grid of K agents constituting the lowest level. Each sensor agent at the middle level coordinates one row of the quadratic sensor agent grid. The middle level agents are coordinated by a single manager-agent at the top level, which also communicates with the mandator M (receiving a task T_0 and returning a result R). Both the middle-level and the top-level agents implement only cc. Fig. 4 shows three examples of team formation in

the flat structure: (i) in the left subfigure (a), the formation is completely sequential, i.e. an external mandator picks a suitable contractor from the set of potential contractors, which, after completion of its own subtask, forwards the remaining subtaks(s) and the preliminary results to agent 2. The latter forwards the "left-overs" to agent 3, freeing itself for new tasks, until agent 6 returns the completed result to the mandator. (ii) The situation in subfigure (b) is similar except for the original agent 4 failing during task execution and being replaced with an alternative agent (with identical or similar profile). (iii) Subfigure (c) illustrates the general case: tasks are fed into the network from many mandators and spread out over the network to be worked on in parallel and to share agents (shaded in (c)).

3 MODELING AND SIMULATION OF NETWORK PERFORMANCE

In this section we briefly introduce our analytical model used as the basis for extensive simulation runs on a commercial transaction-based simulation system, in which we compared the hierarchical and the flat structures discussed above. The sensor component sc of a sensor agent is represented as an $M/M/1$-queue, i.e. a service center with exponentially distributed inter-arrival times of new tasks and an exponentially distributed service time. The coordination component cc is represented as an $M/D/1$-queue, i.e. with exponentially distributed inter-arrival times and a constant service time. For the purposes of our

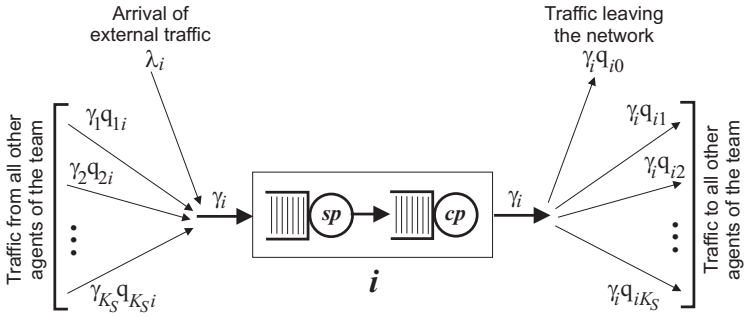


Figure 5. Traffic balance at network agent i.

simulation, a task arriving at an agent is first processed by its sensor component sc and then by its coordination component cc. In particular, it is assumed that an external mandator was already located for each newly arriving task. The rate of newly arriving tasks λ_i at a sensor agent i with $i = 1, \ldots, K$ is termed *external arrival rate* and is assumed to be identical for all agents. Thus, the total external arrival rate is given by $\lambda = \lambda_i K$. A task processed by a sensor agent i is routed to a sensor agent j of a team consisting of $K_S \leq K$ agents (which are competent to work on the task) with probability q_{ij}, where $i, j = 1, \ldots, K_S$. We assume that the agent does not contribute to its own input traffic, i.e. it does not act as its own mandator, and hence $q_{ii} = 0$. The task exits the network when it was successfully completed with probability

$$q_{i0} = 1 - \sum_{j=1}^{K_S} q_{ij}, \text{ with } i = 1, \ldots, K \tag{1}$$

The probabilities q_{ij} are called the network routing probabilities. The tasks arriving at agent i from other agents j (because of contracting) are a fraction of the total rate of tasks γ_j leaving sensor agent j with $j = 1, \ldots, K_S$. The rate of traffic flowing into agent i is called the internal arrival rate of agent i and is given by

$$\gamma_{i_{int}} = \sum_{j=1}^{K_S} \gamma_j q_{ji} \tag{2}$$

Due to the flow balance assumption in a steady state system, tasks must leave a sensor agent at the same rate at which they arrive there. A fraction q_{ij} of the set of tasks arriving at agent i are directed from sensor agent i to sensor agent j with the rate $\gamma_i q_{ij}$. Furthermore, a fraction q_{ji} of tasks is directed from sensor agent j to sensor agent i with the rate $\gamma_j q_{ji}$. Consequently, the total traffic rate γ_i at a sensor agent i is given by the network traffic equations (see Fig. 5):

$$\gamma_i = \lambda_i + \sum_{j=1}^{K_S} \gamma_j q_{ji} \tag{3}$$

where $1, \ldots, K$. In our simulation, the external arrival of tasks is assumed to be a stationary Poisson process. However, the internal arrival rate is not necessarily such a process: in the case of a dynamic selection strategy (such as selection by smallest workload), arrival rates depend on system history. Moreover, as will be explained below, the probability q_{ji} of a task arriving from sensor agent j at sensor agent i is a function of the number of sensor agents which have already processed this task. The coordination component of an agent decides by means of an evaluation function whether a task processed by the sensor component can be successfully completed. Up to now no further assumptions on the nature of sensor data evaluation were made. However, a probability distribution needs to be defined for the number of agents working on a task before it is completed. For reasons of simulation simplicity, the process of K_S agents transferring a given task and accepting it for completion or rejecting it is viewed as a Bernoulli experiment (other distributions can easily be introduced into the simulation). After each transfer, the task is accepted by new agents with probability b and rejected with probability $1 - b$. We assume that there is a final agent that communicates the result to the mandator. The probability of the k^{th} agent accepting the task for completion is then given by

$$P(k) = (1-b)^{k-1} b \tag{4}$$

The corresponding geometric probability distribution is given by

$$F(k) = 1 - (1-b)^k \tag{5}$$

This function determines the probability q_{i0} of a task exiting the network as successfully completed by sensor agent i after having been worked on by k agents including i with $k \geq 1$ and $E[k] = 1/b$. Additionally, q_{i0} is set to 1, should the chosen processing deadline requeired by the mandator (e.g. because of real-time scheduling constraints) have expired at the time a task arrives. Based on these assumptions, hierarchical and flat structures were compared in performance.

Before turning to the simulation results we list some of the important simulation parameters. The main simulation *output parameter* of interest (and hence the measure of the comparative performance of the network structures used) is the percentage V of tasks successfully completed before a given deadline (other output parameters are the mean time a task remains in the network before completion and the average population of taks). The following variables were among the simulation input parameters, which were introduced to determine the behaviour of the modelled organisation expressed as the value of V:

- The network size K defining the number of sensor agents in the network. With the hierarchical structure, K is the number of agents on the lowest level with the top level having one agent. K is a square number, and every agent at the intermediate level controls $\sqrt{K}$ agents (e.g. $K = 9$ in Fig. 3);
- The relative processing deadline d within which tasks should be completed (in time units);
- The probability f of a sensor agent failing at a specific point in time;

- The repair delay r after which failed sensor agents return into the system (in time units);
- The completion probability b determining the mean number of nodes required to successfully complete a task (low level nodes with hierarchies);
- The sensing component service time sp and the coordination component service time cp.

Time units are relative to the average service time of the agent, i.e. $cp + sp$. Further input parameters, whose influence on the throughput V we studied, were the *selection strategy* (workload dependent vs. static), the average communication delay m between agents and the update time interval Δ it takes before mandating agents are informed about the individual workload of potential contractors.

The selection strategy employed in both network structures is selection by smallest workload (if more than one sensor agent is ready to accept the task).* As a measure of difficulty of the task, the coefficient b was varied, a decrease in b resulting in an increase in $E[k]$, the mean number of sensor agents necessary to successfully complete a task. The node failure probability f and repair delay r as well as the network size K are viewed as measures of complexity. Additionally, the coordination component service time cp was varied with respect to the sensor component service time sp to represent an increasing complexity in reaching coordination decisions.

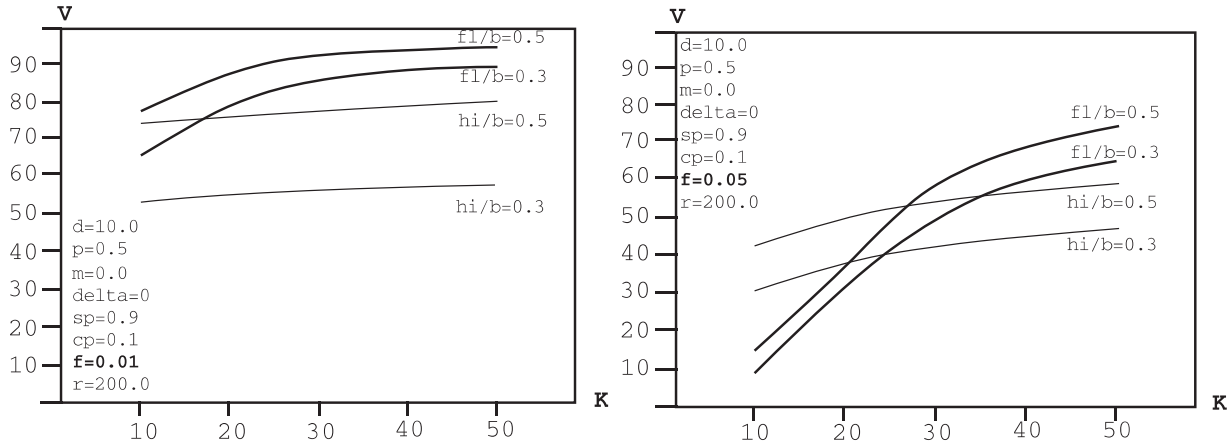


Figure 6. Left – Effect of network size K on throughput V. Two parameters: network structure and completion probability b. Low failure probability f and long repair delay r. **Right** – Same parameters but high failure probability f and long repair delay r.

3.1 Simulation results

We will now present some of our simulation results that show the performance of the network for the most interesting cases, i.e. when reconfiguration is necessary because of the failure of individual nodes. Fig. 6 illustrates the effect of increasing the network size K and varying the failure probability f, which was 0.01 and 0.05, respectively. In addition, the performance of flat (`fl`) and hierarchical (`hi`) organisations are shown for different degrees of task completion probabilities b, i.e the reciprocals of task complexity. In the case of low failure probability f (Fig. 6, left), the superiority of `fl` over `hi` is evident: even with

*We also ran extensive simulations using alternative strategies, which were published elsewhere [22].

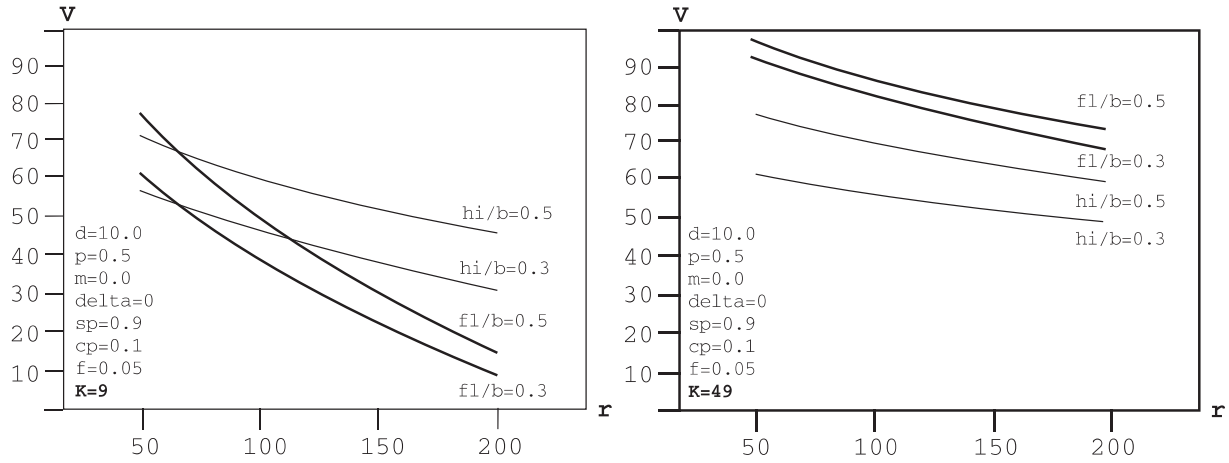


Figure 7. Left – Effect of the repair delay r on V with varying completion probability b. High failure probability f and small network size $K = 9$. **Right** – Same parameters but network size $K = 49$.

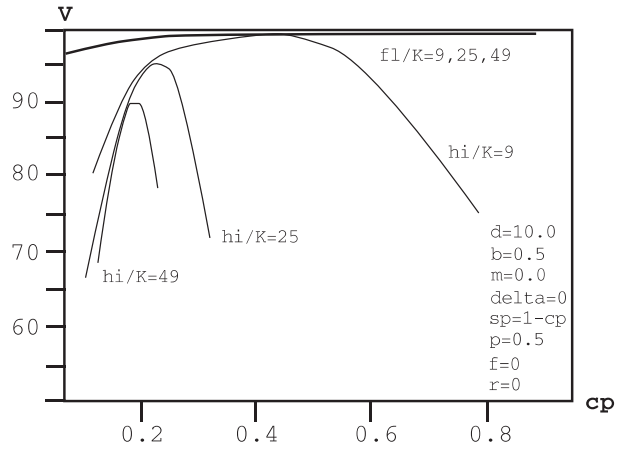


Figure 8. Throughput V as a function of the coordination component service time cp. cp is measured as a fraction of the total service time.

small network sizes, `fl` provides a higher percentage V of tasks completed successfully within the deadline d than `hi`. This advantage increases with growing network size K because the set of potential contractors that a mandator may select from becomes larger in the case of `fl`. This is also true of the case of high failure probability. Here, too, a larger network size will increase the likelihood to find a working contractor, reducing the average amount of time needed for repeated futile negotiation phases. Moreover, in a larger network an increase in complexity (b getting smaller) results in less degraded performance for `fl` when compared to `hi`. With b decreased from 0.5 to 0.33 (and hence $E[k]$ increased from 2 to 3), `fl` suffers from a degradation of about 5%, whereas `hi`'s performance degrades by approximately 20% under identical conditions. We note that with small network sizes, `hi` shows a better performance than `fl`. However, as K increases, a break-even point is reached, at which `fl`'s performance exceeds that of `hi`. With completion probablity decreasing, this break-even point moves to decreasing network sizes K.

In Fig. 7 the organisation performance is plotted as a function of the repair delay r with high failure probability f fixed

at 0.05. The repair delay corresponds, for example, to the time it takes to re-focus a sensor if the current focus turns out to be inadequate after taking a measurement. Here, the probability f defines how frequently this happens. Initially, with a small network ($K = 9$) and short repair delay, `fl` is at an advantage over `hi`. With increasing repair delay, `fl` performance degrades significantly below `hi` performance. In the case of a large network ($K = 49$; Fig. 7, right), the situation is reversed. Here, even with long repair delays r, the lateral organisation outperforms the vertical organisation. This is due to the fact that in our model the repair delay only affects the sensing component of an agent (communication facilities normally need not be re-focused) and hence the intermediate and top level agents are not affected by this delay. With increasing K, the ratio between the nodes is not affected and the complete sensor agents becomes smaller, and hence this advantage vanishes. It is also clearly visible that the difference in organisation performance increases with completion probability getting smaller. This sensitivity of `hi` to increased network size is explained by the bottleneck effect affecting upper-level managing agents.

Another measure of complexity is the amount of time required by a coordination component to reach a coordination decision (fraction of cp of the constant total service time $sp + cp$). Its influence was investigated for a variety of parameter settings, and some of the results are displayed in Fig. 8. The network size does not affect `fl` performance, but critically affects `hi` performance. Here, another interesting feature of `hi` performance was encountered: As cp increases (and sp correspondingly decreases), `fl` performance remains relatively stable, rising from nearly 100% to a full 100% of successfully completed tasks. This is largely due to the fact that sp is modelled as exponentially distributed and cp follows a uniform distribution. This setting is relevant for networks that consist of a large number of coordinators (that do not have a sensing component). However, besides being sensitive to increased network size due to bottleneck effects, `hi`'s performance rises sharply with increased service time cp. The performance peak almost reaches `fl` performance, sharply falling off as cp becomes larger: the bottleneck characteristic of managing agents is directly amplified by increasing cp. The decrease to the left of the optimum is not so obvious. A detailed study reveals, however, that with cp service times decreasing, tasks may flow increasingly faster through the hierarchy, leading to saturation effects in the subsets of low-level agents coordinated by the middle level managers (this behaviour would change if we chose a different selection strategy). The results displayed in Fig. 8 suggest that, in contrast to the robustness of `fl` performance, a hierarchical organisation is sensitive to the relation of sp to cp service times. Thus, if a hierachical organisation is chosen, these parameters must be carefully tuned for maximum throughput.

4 GENERIC IMPLEMENTATION OF ADAPTIVE CONTRACT NETWORKS

The results we obtained from our simulations motivated us to embark on a long-term research effort to develop and implement the generic software framework *MagiC* (**M**ulti **a**gent **g**eneration **i**n **C**++), which provides all the communication and protocol building blocks to rapidly design large networks of interacting agents (both `fl`, `hi` and mixed). From both the software and the system engineering point of view, it is desirable to split the design of a system as complex as an Adaptive Contract Network (ACN) into different layers of abstraction:

1. A *logical sensor specification* abstracts from the physical implementation of the sensor and, if provided in the form of an abstract data type, may directly provide the software interface of a logical sensor. It may also form the basis for implementing the software processes necessary to transform requests received via this interface into sensor actions and to transform the resulting sensor signals into the appropriate data domain.

2. A *fusion method specification* which defines the input and result parameters of the fusion method(s) available in the system for performing certain tasks. It may be part of the logical sensor specification.

3. A *task specification* defining the task to be performed and the expected result parameters (data types) that are expected upon task completion in terms of the object and/or environment model.

4. A *communication protocol* associated with a physical communication system through which the logical sensors may exchange raw data with each other or with an external mandator.

5. A *cooperation protocol* specifying which logical sensor may interact with which other one under a certain task (class). It implicitly defines the architecture of the ACN.

Layers 1 and 2 define the private properties of the sensor while 4 and 5 are provided by an embedding system. Layer 3 is the interface connecting them. It is important to note here that the sensor agent must provide a more or less elaborate facility for *self-assessment*, i.e. it must be able to estimate both the quality of the result it may deliver and the time it will need to compute the result in a given task context.

It was our goal to make available to programmers an integrated platform that provides procedures to efficiently combine many physical sensors and fusion methods into an operational system. This system should handle diverse tasks with only a minimum of input about the sensing strategy, team formation and other administrative information. Moreover, the ACN should work with only a minimum of knowledge on the programmer's side about the individual sensor. To this end, the sensing agent must be endowed with enough "intelligence" to perform all the necessary configuration and parameter adaptation at run-time. To specify a task for the sensor network only four parameters need to be passed to the ACN:

- **Action:** definition of what is to be done by the sensor. It must match with at least one of the actions specified in one of the sensor agents;
- **Parameter:** definition of the data to be worked on, primarily the required input and the desired result (if any);
- **Quality:** required quality (precision) with respect to some crisp or fuzzy measure that the result must possess in terms of object/environmental parameters (e.g. velocity, length, orientation, distance, etc.);
- **Deadline:** maximum time until task completion (or task being returned as – temporarily – not performable).

In all our implementations the protocol/communication overhead resulting from these specifications never exceeded a few hundred bytes. Even with slower networks, this is no obstacle to real-time performance. Currently, the task definitions are flat, i.e. they cannot be nested. It seems to be promising, however, to also allow recursive definitions (or to be formulated as scripts, see section 6.1). If these specifications are provided in a coherent form using a common formalism, e.g. a synchronous functional language [23] or C++ classes, and if the aforementioned protocols are implemented in both the communication system and the sensor agents, then systems with the following features may be built:

- Groups of sensors with different physical measurement principles can be combined into one ACN.
- A sensor may be added to the sensor network *at run time* with very little action required on the system operator's part. If, conversely, a unit fails, it may just as easily be phased out of the network (ideally, the failure is recognised automatically an the unit is simply not considered as being able to take part in any further contracting negotiations).
- Logical sensor specifications can be easily written (templates are provided).

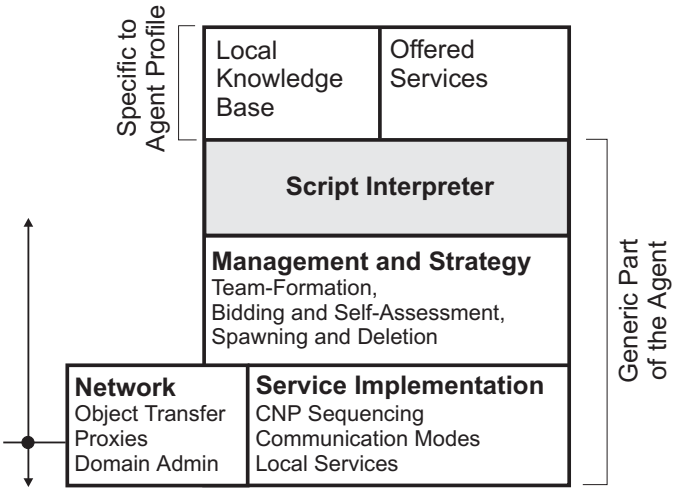


Figure 9. Structure of Agents in *MagiC*. The script interpreter is optional (see sec. 6.1).

- Task specifications are simple and human-readable.

The software system *MagiC* we describe in the sequel implements a complete ACN development environment. It is fully operational on both Solaris and Linux platforms. *MagiC* is built on C++ as its supporting language, not least because nearly every interface to sensor/actuator-hardware (e.g. robots, force-torque sensors, cameras, frame-grabbers, etc.) is available in C/C++. *MagiC* also benefits from the fact of C++ being an object-oriented programming language with features like abstract data types, inheritance, virtual methods. Moreover, there is a strong conceptual link: using a *weak notion of agency* [24,25], in which *autonomy, social ability, reactiveness* and *pro-activeness* are relevant characteristics of an agent, one may think of C++ objects as being *agents* except for having appropriate social abilities like communication and cooperation. The idea of agent design in *MagiC* is to provide a base class with certain communication and cooperation capabilities. The task of designing agents is thus reduced to filling in (predefined) virtual methods that realise the interface between the agent's local knowledge base of its own methods and data types according to the abstract layers mentioned above. The most important features of *MagiC* agents are

- Agents are reactive, proactive, or both;
- No distinction is made between local or remote agents running on diverse platforms (virtual agent space);
- Agents may call other (teams of) agents to provide services to the mandator, i.e. these services may be elementary or very complex;
- Many tasks may be handled parallel, only limited by the number of agents available (instantiated).

Agents can take on the roles of mandators and contractors dynamically depending on their own private state.

4.1 Agent objects in *MagiC*

Fig. 4.1 shows the structure of *agent objects* in *MagiC*. Each agent is derived from the base class `agent_c` and inherits methods for communication and negotiation enabling the agent to bid for a task, to perform a task and to offer tasks. Apart from these

capabilities, each agent must at least define a `service`-method and a signature of this service containing the class name of the agent, the parameter class name the agent can deal with and the result class name. Both parameter and results must be derived from the `TBaseObject` class:

```
class my_new_agent : public agent_c {
private:
    <local members>
public:
    virtual bid_c             *make_bid  (TBaseObject *parameter, quality_c *required);
    virtual list<TBaseObject> *service   (TBaseObject *parameter);
    virtual void               select_contractors(list<bid_c>);
    SIGNATURE(my_new_agent, my_input, my_result);
};
```

Purely reactive agents wait in the background for task announcements of other agents in the network. If their signature matches with that of the task, the agent's bid-method `make_bid` is called automatically. Within this bid the agent assesses the quality of its own bid and compares it with the required one. The agent can also decide *not* to bid for the task if its local state indicates that it is not able to perform the task (e.g. because some resources are not available yet). If the agent does not provide the agent its own version of `make_bid`, a default bid is made on a the signature match. This behaviour simply states the fact that this agent can in principle perform the task. Proactive agents may also take on the role of a mandator. There are three ways of selecting contractors: (i) first-come, first-serve: this is the default; (ii) quality-criterion: only contractors bid for the task that fulfill the criterion (time, existence, precision, etc.); if more than one replies *MagiC* automatically selects the best by comparing the quality; (iii) selection method: if the user defines a `select_contractors`-method, all incoming bids are evaluated by this method. Due to the fact that C++ does not allow a transfer of objects between processes (as it is possible in Java or Modula-3) a special base class `TBaseObject` was defined which supplies this feature for every class derived from it.

4.2 Tasks

The execution of tasks follows simple rules. The mandator constructs a task description containing the actual task parameter, the expiration time, the intended quality and the task signature. Each description has to be derived from the `task_c` base class. By default, the new description class definition must contain a "`signature`" statement equal to that exported by the prospective contractors. It is used during task announcement to determine the tasks signature automatically. A new instance is created by calling the constructor:

```
task = new my_new_task(parameter, quality, deadline);
```

The action-specification in *MagiC* is realised by the `signature`-statement contained given in the task description. Possible contractors are agents with the same `signature`. After announcing the task by broadcasting the description, the mandator waits for contractors, selects according to its selection method the best agent(s) and waits for a result or a failure. This is all done by calling the agents inherited method `announce`:

```
my_new_agent manager;
result = manager.announce(task);
```

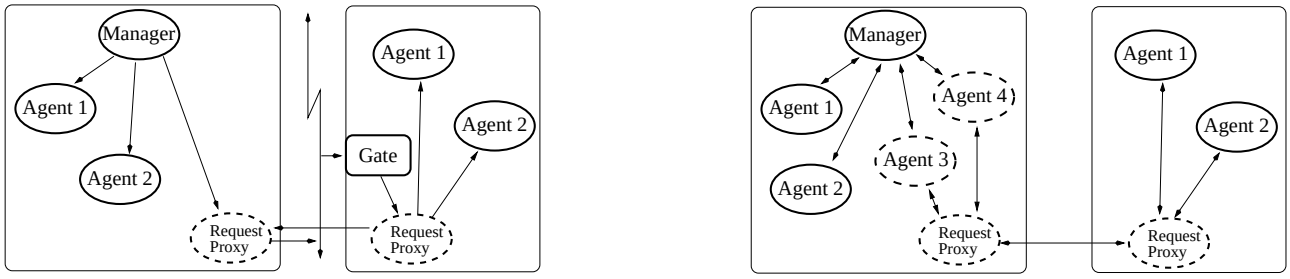
in which `result` is a pointer to a result list and `task` the appropriate description.

4.3 Teams

Apart from single task announcements to find appropriate contractors, *MagiC* provides the designer of an agency with the facility of requesting *teams* of agents to which a task is announced iteratively up to the release of the team. The idea behind the teams (see sec. 2) is that there are certain tasks (e.g. for survey operations) for which it is useful to find a set of contractors through negotiation and not to release them after a single task. One advantage of a team is that a task may automatically be split into individual parts according to the number of team members with just one announcement instead of announcing several sub-tasks. In *MagiC* all contractors may be requested *exclusively* which means that their internal state is only affected by announcements of one manager (e.g. for a simple tracking-agent the template of the object to be tracked is valid up to a re-initialisation of the manager). After announcing a team task, a unique team member ID is automatically assigned to each contractor. After receiving the results from all team members or in case of an unexpected event (such as the agent is not answering at all or to slowly) the virtual method `reconfigure_team` is called in which the agent designer may exactly define how to react (for instance to release a particular agent or to announce the task again). This enables a team to *reconfigure itself* during task execution.

4.4 Virtual Agent Space

Technically, each *MagiC* agent is modelled as a thread inside a process. By employing the proxy-principle [26], all agents are addressed in the same manner simply by announcing a task, regardless of their location (in the same process, on the same computer or on a computer anywhere in the network). The overall effect is the illusion of a *virtual agent space* of all agents. Fig. 10 shows the interaction between agents hosted in two processes on different computers. Beside the $Manager$



(a) Initialising proxies

(b) Virtual local agents

Figure 10. Virtual agent space using proxy-agents.

agent there are two other local agents $Agent_1$ and $Agent_1$ in the same process. Due to the task announcement a local but invisible *request proxy* is created which propagates the announcement over the network. This announcement is received by *gate-agents* which are located in every process containing *MagiC* agents, Fig. 10(a). These gate-agents create a local request proxy which in turn connects immediately to the announcing proxy (this enables the process to create new proxies for each incoming announcement). After this connection is established, the whole negotiation between the processes is handled via the proxies (Fig. 10(b)). The proxy in the announcing process works as contractor to the manager and the local agents of the remote process are contractors to their local proxy. This gives the manager the illusion of "seeing" every contractor in its local process.

5 APPLICATION I: DECENTRALISED SENSING AND FUSION ON THE DATA LEVEL

In this section we present an example of a real-world implementation of fusion on the data level, i.e. on slightly preprocessed images of uncalibrated colour cameras. Using *MagiC* these methods together with the necessary data flow were very straightfor-

ward to realise as encapsulated agents, and task adaptation as well as fault-tolerance through reconfiguration did not necessitate any further implementation efforts.

The general task is *visual servoing with many cameras*: a 6-dof manipulator (one of the PUMA 260 manipulators in our setup shown in Fig. 1) is to be positioned over a desired target (see Fig. 11). A set of four *arbitrarily positioned uncalibrated* colour cameras are used (note that other than financial resources, there is no principal limit to the number of cameras). We start by assuming a parallel-camera model for the image formation process. Defining an image-based position error in j different views and employing the parallel projection camera-model leads to a simple linear equation for a resulting Cartesian correction movement called the *fusion equation*. The parameters in turn are estimated with a linear Kalman filter (KF) using measurements obtained by the different cameras.

(a) (b)

Figure 11. Initial (a) and final position (b) of the manipulator and its projected trajectory as seen from one camera.

5.1 Solving the Fusion Equation

The parallel projection $\boldsymbol{P}^j$ (see [27]) used to generate the feature $\boldsymbol{f^j}$ of a 3D world point $\boldsymbol{m}$ in *homogeneous* coordinates $\boldsymbol{m}^w = (m_x, m_y, m_z, 1)^T = (\boldsymbol{m}, 1)^T$ onto the j^{th} camera image plane is

$$\boldsymbol{f^j} = \begin{pmatrix} r_{11}^j & r_{12}^j & r_{13}^j & t_1^j \\ r_{21}^j & r_{22}^j & r_{23}^j & t_2^j \end{pmatrix} \cdot \boldsymbol{m}^w = (\boldsymbol{R}^j \; \boldsymbol{t}^j) \cdot \boldsymbol{m}^w = \boldsymbol{P}^j \cdot \boldsymbol{m}^w \tag{6}$$

Assuming that both the target and the manipulator may be represented as points in Cartesian 3D space, a simple error function for a linear point-to-point movement of a manipulator at $\boldsymbol{m}$ to a goal $\boldsymbol{g}$ may be defined as the error-displacement vector $\Delta \boldsymbol{d_e}$ to be minimised:

$$\Delta \boldsymbol{d_e} = \boldsymbol{m} - \boldsymbol{g} \rightarrow 0. \tag{7}$$

For the corresponding displacement feature $\Delta \boldsymbol{f}_e^j$ in the j^{th} camera, using eq. (6) this results in a simple linear relationship:

$$\begin{aligned} \Delta \boldsymbol{f}_e^j &= \boldsymbol{f}_m^j - \boldsymbol{f}_g^j \\ &= \boldsymbol{P}^j \cdot \boldsymbol{m}^w - \boldsymbol{P}^j \cdot \boldsymbol{g}^w \\ &= \boldsymbol{R}^j \cdot \boldsymbol{m} + \boldsymbol{t}^j - \boldsymbol{R}^j \cdot \boldsymbol{g} - \boldsymbol{t}^j \\ &= \boldsymbol{R}^j \cdot \Delta \boldsymbol{d_e} \end{aligned} \tag{8}$$

Given a base of three orthogonal displacement vectors[†] $\{\boldsymbol{d}_1, \boldsymbol{d}_2, \boldsymbol{d}_3\}$ the error-displacement vector $\boldsymbol{d_e}$ can be calculated by their linear combination:

$$\boldsymbol{d}_e = \sum_{i=1}^{3} \xi_i \boldsymbol{d}_i, \text{ with } \xi_{1,2,3} \in R \tag{9}$$

where $\xi_{1,2,3}$ are linear factors defining the incremental movement vector in the initial base frame $\boldsymbol{d_{1,2,3}}$. Using $\boldsymbol{R}^j$, the projected version of Equation (9) is:

$$\boldsymbol{f}_e^j = \boldsymbol{R}^j \cdot \boldsymbol{d}_e = \boldsymbol{R}^j \cdot \sum_{i=1}^{3} \xi_i \boldsymbol{d}_i = \sum_{i=1}^{3} \xi_i \cdot \boldsymbol{R}^j \cdot \boldsymbol{d}_i = \sum_{i=1}^{3} \xi_i \boldsymbol{f}_i^j \tag{10}$$

Hence, $\boldsymbol{f}_e$ is a linear-combination of the projected base using the *same* $\boldsymbol{\xi} = (\xi_1, \xi_2, \xi_3)^T$ as in 3D. Calculating an appropriate set of scalars $\xi_{1,2,3}$ in the image space and inserting them into Equation (9) leads directly to the desired displacement-vector in the Cartesian 3D space. Equation (10) is under-determined. Therefore, at least two views are necessary yielding an over-determined system. Assuming a redundant multi-camera system with j different cameras, all views can be integrated simply by solving the following over-determined system:

$$\underbrace{\begin{pmatrix} \boldsymbol{f}_e^1 \\ \boldsymbol{f}_e^2 \\ \vdots \\ \boldsymbol{f}_e^j \end{pmatrix}}_{\boldsymbol{z}} = \underbrace{\begin{pmatrix} \boldsymbol{f}_1^1 & \boldsymbol{f}_2^1 & \boldsymbol{f}_3^1 \\ \boldsymbol{f}_1^2 & \boldsymbol{f}_2^2 & \boldsymbol{f}_3^2 \\ \vdots & \vdots & \vdots \\ \boldsymbol{f}_1^j & \boldsymbol{f}_2^j & \boldsymbol{f}_3^j \end{pmatrix}}_{\boldsymbol{H}} \cdot \underbrace{\begin{pmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \end{pmatrix}}_{\boldsymbol{\xi}} \tag{11}$$

Eq. (11) plays the central role and is called the *fusion equation*. Only three parameters have to be estimated (independently of the number of cameras) and hence only three initial test movements are necessary. $\boldsymbol{H}$ is simply determined by measuring the projection of each test move in all the images. We use a linear discrete Kalman filter to estimate the parameters of Equation (11). Assuming zero-mean, white-noise for $\mathbf{v}$ and $\mathbf{w}$, the plant and measurement equation are:

$$\begin{aligned} \xi(k+1) &= \xi(k) + \mathbf{v}, & \mathbf{v} &\sim N(0, \mathbf{Q}) \\ \mathbf{z}(k) &= H(k) \cdot \xi(k) + \mathbf{w}, & \mathbf{w} &\sim N(0, \mathbf{R}) \end{aligned} \tag{12}$$

where the system noise $\mathbf{v}$ represents uncertainty in the manipulator position, and the observation noise $\mathbf{w}$ results from camera and preprocessing fluctuations. The incremental prediction and update solutions can be found in [28]. We have chosen pure diagonal matrices for $\mathbf{Q}$, $\mathbf{R}$ and the initial state covariance $\mathbf{P}_{(\mathbf{0}|\mathbf{0})}$ with the diagonal elements $\sigma^2_{P_{(0|0)}} = 0.1, \sigma^2_Q = 0.01$ and $\sigma^2_R = 5.0$. The initial state-estimate is set to $\boldsymbol{\xi}_{(0|0)} = (1,1,1)^T$. For a point-to-point movement to a selected target the projection $\boldsymbol{f}_i^j$ of the manipulator during the three Cartesian test moves are obtained first. With the measured position-residuals an initial down-scaled correction movement $\boldsymbol{d}_c = s \cdot \boldsymbol{d}_e, \in (0,1]$ is calculated. After each movement a new $\boldsymbol{\xi}$ is estimated. This is iterated as long as the target is not reached ("dynamic look-and-move"). Results from simulations and real experiments showing the accuracy of the proposed method and how it improves if more cameras are added can be found in [29].

5.1.1 Modelling as an ACN

The method of visually guiding a manipulator with a set of redundant cameras as described above fits nicely into the *MagiC* framework and can be easily modelled as an ACN. From each camera only the position-residual $\boldsymbol{f}_e^j$ between the goal and the manipulator needs to be obtained for solving eq. (11). Therefore, we can assign each camera to a *track-agent* which is able

[†] Because only displacements are considered, the Δ is omitted in the sequel.

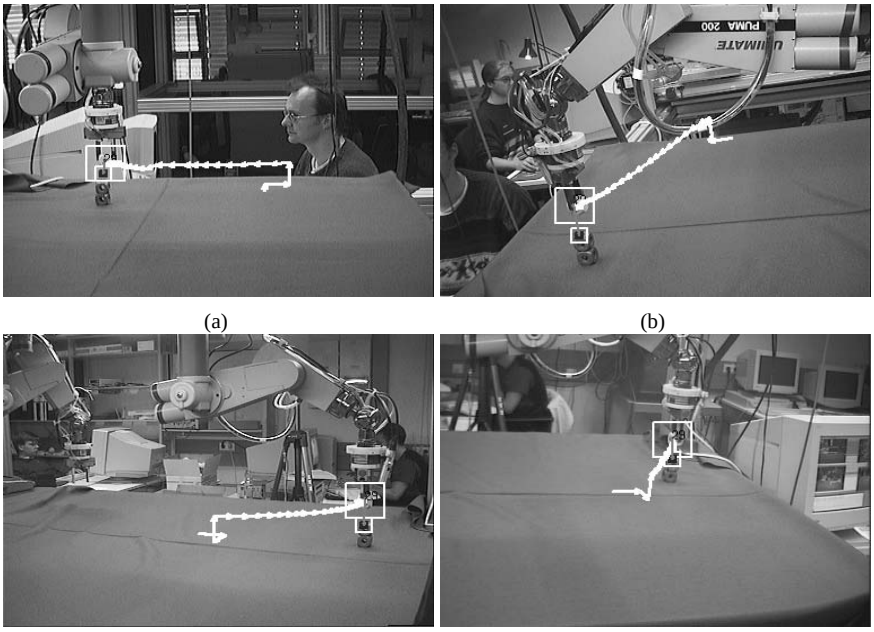


Figure 12. Redundant visual servoing. End positions and projected trajectories as seen simultaneously from four different camera locations.

to measure the local positions and sends them back as a result of a task announcement made by a top-level-manager which in turn solves for the parameters of the fusion-equation and controls the robot. Each track-agent decides to make a bid depending whether it sees both the target and the manipulator.

We carried out three experiments (redundant visual guidance without failures, with contractor failure, with contractor failure and automatic reconfiguration) to show that the ACN implemented with *MagiC* is self-organising with respect to the selection of tracking agents, but that it also tolerates the complete failure of contractors. The agency-setup for all experiments consisted in one pure manager agent M announcing tracking tasks and up to four different tracking-agents, each in its own process. During the experiment, only three frame-grabbers were available forcing the tracking-agents to share access by announcing grabbing-tasks to up to four grab-agents. The latter ran on the computers hosting the frame-grabber and are modelled as "infinite team" agents due to the necessity to share the grabbers between requesting tracking-agents – giving them the illusion of exclusive access to a certain channel of the grabber. The basic vision method for recognising the robot position is simple template matching.

5.2 Redundant visual guidance

The goal of the first experiment was to guide the robot carrying a yellow wooden cube above a desired target (a "tower" of two other cubes) using all potential contractors. Fig. 12 shows the final images taken from the different viewpoints including the projected trajectory performed by the robot and the last search window for the template-matching. After 29 iterations (each resulting in a differential move command to the robot) the target was successfully reached. The reason for the slightly non-linear trajectory is the assumption of a linear relationship between 2D and 3D space in eq.8. As with all linear approximations (e.g. the image Jacobian [30]) this assumption holds only above the position where the linearisation was defined. The farther

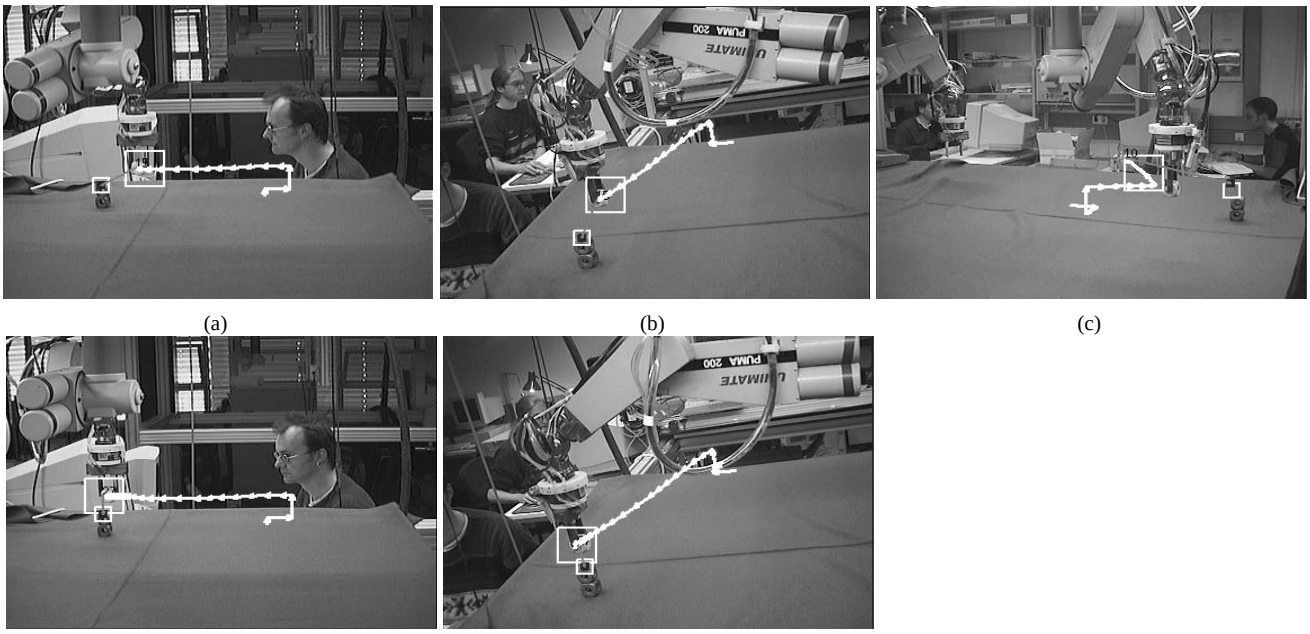


Figure 13. Redundant visual servoing, second situation. The target is still reached despite several failures.

away the robot moves from this position, the higher the errors becomes. Nevertheless, due to the dynamic-look-and-move scheme used here, these errors are permanently corrected, which results in a smooth curved trajectory.

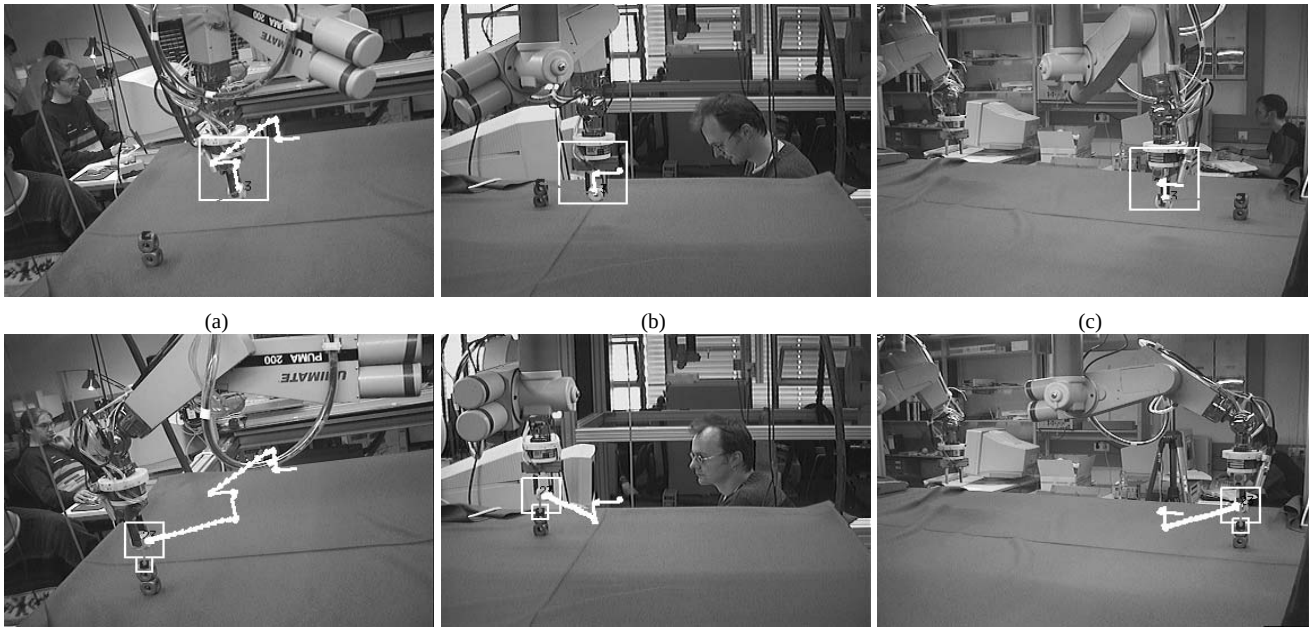
5.3 Contractor failure

The second experiment uses the same setup but with failing agents during task execution. Fig. 13 (a) to (c) show the situation after the failure of one track-agent while the robot is guided towards the target. The failure was induced manually by interrupting the agents' process. Furthermore, the agent shown in Fig. 13(c) has lost contact with the robot and tracks something else. This kind of failure (which can be detected with methods described in [31]) forced the agent in this experiment to terminate itself. Nevertheless there still exist two active agents which guide the robot successfully to the target (Fig. 13 (d) and (e)).

5.4 Automatic Reconfiguration

The third experiment shows the ACN's capability to reorganise itself. During task execution all contractors with the exception of one fail. This causes the ACN to re-announce the task automatically. Figs. 14(a) to (c) shows the situation when the robot makes new test moves[‡] following the re-announcement of the taks. Fig. 14(a) is the view of the agent that was a team member from the beginning, showing its path travelled so far including the new test moves. Figs. 14(d) to (f) show the trajectories of the robot after reaching the target. The new test moves perturbed the robot's position at the location where the complete failure occurred. Again, this inadvertent breakdown gives an example of the capability of our method to dynamically control the robot in three dimensions using more than one two-dimensional projection of its motion in 3D-space (e.g. in Fig. 14(e), the robot has to move upward).

[‡]New test moves would not be necessary if it is plausible that the old calibration parameters are stil valid. We chose this behaviour to make it more difficult for the system to return to the original trajectory.



(a) (b) (c)

(d) (e) (f)

Figure 14. Redundant visual servoing, third situation. Trajectories resulting from automatic reconfiguration.

6 APPLICATION II: DECENTRALISED SENSING AND FUSION ON A SYMBOLIC LEVEL

As shown in the previous section, the basic contract network architecture of *MagiC* is well suited to rapidly designing implementations of classical data fusion methods including applications that require real-time performance, e.g. reactive robot control. While the network structure is highly flexible to enable fault tolerance through adaptive run-time reconfiguration, the ability of the network to adapt to structurally new tasks (e.g. due to object domain changes) is relatively limited. If at all possible, it can be accomplished only through team formation. The reason is that the task class is fixed for each agent (except for parameter changes). For high-level systems that fuse information on a symbolic level[§], it is mandatory that agents be capable of communicating such symbolic information to the other agents. Moreover, they should offer a maximum of flexibility in terms of task classes, object classes, methods, parameter sets, etc. to the agent community.

The system for decentralised vision and fusion presented in this section was developed with the goal of endowing the agents with maximum "intelligence" in mind. It is built on top of *MagiC*, and it offers the potential for three-fold flexibility: (a) through autonomous network (re-)configuration, (b) through a high-level communication/command language for agent interaction (human readable and writeable, but powerful enough to describe complex facts and tasks) and (c) through the ability to completely *program* each agent's behaviour in a powerful script language (with scripts generated and transmitted *by other agents*). It turned out in our practical work with this distributed vision/fusion system (called DiVA, for **Di**stributed **V**ision **A**rchitecture) that (i) it is very simple to use in complex setups (ii) it is very easy to adapt to different tasks and environmental conditions (iii) it is very straightforward to extend by adding new processing modules at run-time.

In general, recent research has indicated that modeling vision systems as societies of autonomous agents is a promising and powerful approach: Boissier and Demazeau proposed the MAVI [32] system, a multi-agent system for visual integration,

[§]Examples are systems consisting of a number of object recognisers that integrate recognition results from different camera view points or systems that reason about optimal sensor placement depending on environmental conditions.

which is based on the ASIC [33] multi-agent control architecture. This architecture is subdivided into different processing layers. this, however, is too complex for most vision applications and makes it difficult to handle the system. Following the purposive vision paradigms, Bianchi and Rillo [34] have proposed a multi-agent vision system employing a behaviour-based decomposition into specific tasks, while Yanai and Deguchi [35] have developed an object recognition system for integrating different vision strategies. Contrary to the MAVI system, these approaches share a more rigid architecture, which reduces the applicability to different environmental conditions and requirements.

In the rest of this section we shall explain in more detail the system architecture, the communication language, and the interaction strategy of DiVA. Towards the end of the section we present experimental results obtained with this system, both for the recognition of occluded objects and for the symbolic fusion aiming at improving object recognition.

6.1 DiVA Agent Architectures

Since computer vision algorithms are frequently time-consuming, DiVA provides two different classes of agents: *master agents* and *slave agents*, with different internal architectures. The former accomplish all of the planning and interpretation as well as many of the required image processing tasks. The latter are responsible for assisting master agents in performing those time-consuming tasks, where slave agents generally work in teams controlled by corresponding master agents to compute sub-results. The number of both classes of agents vary dynamically depending on the specific observed scene and the type of vision/fusion task to be performed.

6.1.1 Master Agent

Master agents perform complex planning and interpretation tasks. They are composed of five modules (Fig. 15):

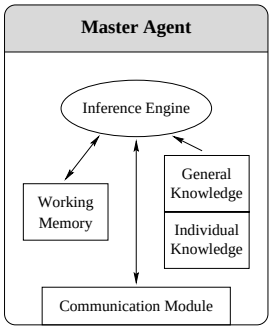


Figure 15. Architecture of a master agent.

1. *Communication module:* It contains methods for sending and receiving messages as well as functions for wrapping various data types.

2. *General knowledge:* General knowledge refers to basic planning strategies and the grammar of the communication language. Most of the knowledge is stored in rules and facts that are applicable to different situations to determine the behavior of the agents.

3. *Individual knowledge:* Knowledge pertaining to individual (fine) planning strategies, processing functions and the (subset of) the communication language. Note that it is not required that all agents share the same ontology.

4. *Inference engine:* Based on general and individual knowledge the inference engine does all the planning and interpretation of messages. The main task of the inference engine is to generate program scripts appropriate to solve requested tasks of other agents.

5. *Working memory:* Used by the inference engine for storing various information, like state, sub-results and knowledge about the dynamic environment.

The internal sequence for working on a task is as follows. (i) The agent analyses the task including the instruction, the destination specifications and additional constraints (it is not required, though, that the agent understand all of the source's specifications). (ii) According to the description of the task the agent automatically generates a complex *Clips*[¶]-style program script, that is composed of all required processing functions as well as further requests to other agents. An example for such a script is shown in Table 2. (iii) The program script is executed under control of the agent, to enable the agent to perform an exception handling and to react to unexpected situations.

6.1.2 Slave Agent

Although vision systems can be designed solely by using master agents, the slave agent class is provided as a an additional tool to facilitate the implementation of parallel processing methods to speed up computation. Slave agents are completely controlled by master agents, i.e. the master decides autonomously how many slave agents it wishes to instatiate and which particular processing must be performed. The communication between a master and its slaves is reduced to the absolute minimum. Therefore, slave agents need only a simple architecture containing a communication module, processing functions and rudimentary mechanisms for interpreting messages.

6.2 Communication Language

The syntaxes of communication languages used in the field of distributed computer vision systems tend to be cryptic and yet too simple to express complex facts and tasks; this is true especially for the parts of the communication languages which are relevant for the application itself. Therefore, along the lines of the language KQML [37], we have developed a new flexible communication language called DiVA/CL to meet the following basic requirements: (i) to facilitate the construction of flexible and self-organising vision systems, (ii) to be extensible (but even in its core form it should express complex facts and tasks), (iii) to be simple to understand, i.e. human readable and writable, and (iv) to ensure that efficient mechanisms for interpreting messages are easy to design. In contrast to other communication languages, which define messages composed of function names and parameters, DiVA/CL can be utilised to specify a task by an abstract description, which is independent of the underlying implementation. Therefore, agents can be added and replaced without having any knowledge of the internal structures of the existing agents. DiVA/CL employs message types making the intention of an message explicit:

```
<message> ::= <message-type> <message-content>
```

The defined message types are similar to the ones used in other communication languages [32,34] in that they implement speech acts, but they differ in their semantics:

[¶]C-Language Integrated Production System, see [36]

1. **request** is used to ask for the assistance of other agents. Generally, this message type indicates that the agent can not perform a particular task on its own.

2. **answer** messages are replies to a request or script message, which can be both a result or an error message.

3. **inform** is used for passing additional information to other agents, which is not necessarily needed for performing particular tasks. This type is also used for indicating the presence or absence of agents.

4. **script** can be used for getting direct access to the capabilities of an agent avoiding the interpretation mechanisms. Since the master agents generate program scripts in order to perform requested tasks dynamically, programs can be passed directly.

Furthermore, the message content is subdivided into a message text and additional message data:

```
<message-content> ::= <message-text> <message-data>
```

where the `<message-text>`-slot contains the message text as a string and the `<message-data>`-slot is a list storing different data types, e.g. images and edges. The text string itself can be both a message text or a *Clips*-style program script. Messages are built from a formal grammar. For reasons of efficiency, this grammar allows only one goal for each message. A goal is not a specific entity but a global plan, which can be restricted by additional constraints. The latter can be complex logical expressions containing variables. Example: A simple object recognition task is to extract all ledges as well as all red objects shown in an image taken in our robot setup (Fig. 1). The image is taken by a camera whose server is called 'clinton'. This task can easily be specified using the following message text:

```
(extract ?dest ?src)

(is-a    ?dest object)
(or (has-name ?dest ledge) (has-color ?dest red))

(is-a    ?src  image)
(has-source ?src camera) (has-server ?src clinton)
```

There are two important points to note here. (i) The interpretation of such messages can be realized in a very efficient manner using the pattern-matching facilities of expert system tools; (ii) entities can be identified not only by their unambiguous names but also by their features and attributes. Although such querying requests are very useful and important to vision applications, they are generally not supported by other agent-based vision systems.

6.3 Communication Network

The agents of the society interact with each other following *MagiC*'s implementation of the contract net protocol. However, because slave agents are implemented to assist master agents in performing time-consuming tasks, they often communicate and interact with their corresponding master agents using a subset of the protocol. An example for a (transient) communication flow is sketched in Fig. 16. There are three different types of connections among master agents and their corresponding slave agents, which often occur during a communication. The first type is a single master agent, processing all functions on its own. The second type is composed of a master agent that divides time-consuming tasks into sub-tasks to be solved by its associated

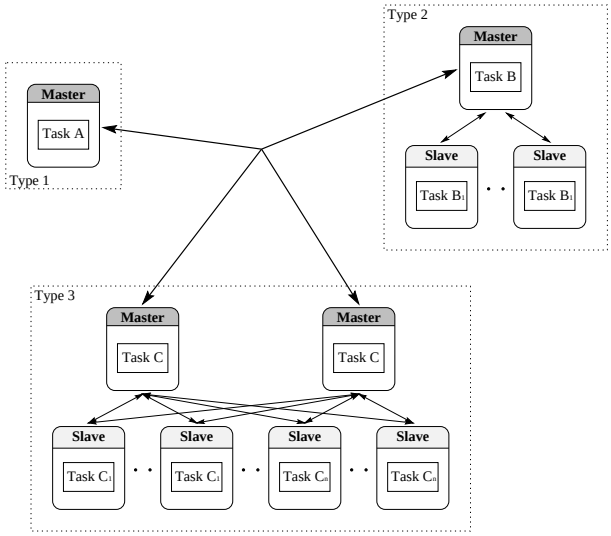


Figure 16. Example of a DiVA communication network.

slaves. The third type consists of two (or more) master agents for the same tasks. Again, they generate sub-tasks performable by their slaves, where the slaves can be dynamically shared between the master agents. Note that the hierarchical structure between master and slave agents is not pre-defined in a static sense; it is purely a result of dynamic communication processes resulting from the scene as taken by the cameras. Following our general philosophy and to ensure a high degree of flexibility, pre-defined hierarchical structures, although possible, should be avoided because this would make it difficult to add, remove or exchange agents.[‖]

6.4 Interaction Strategies

Another aspect affecting the flexibility of the society is the interaction strategy among master agents. Each master agent accomplishes missions received from an external mandator according to its own knowledge and goals. The interaction between the masters leads to a self-organisation of the agent society. This self-organisation process is goal-driven and proceeds as follows:

1. If a master agent (or a mandator) requests a task, all master agents of the society decide whether or not they can accomplish the given task. As mentioned before, this is done by analyzing the instruction, the destination specifications, and the additional constraints.

2. All master agents that are responsible for the particular task make a bid. According to these bids the master agent that has requested the task selects the agents that should be awarded the contract.

3. Then, the selected agents *automatically generate* appropriate program scripts according to the task specifications.

[‖] Such an example is the agent architecture proposed in [34]; here, the modification of an agent or of a behavior may affect other agents, which entails the modification of these agents, too.

While the first two items are standard contract protocol steps, the third one is specific to DiVA (see the following subsection for an example). It generates some overhead, but this way completely new functionalities can be phased in at run-time simply by adding appropriate agents or by sending new scripts. Precise knowledge about the internal structure of existing agents is not necessary if the instructions are well-formed expressions in DiVA/CL.

6.5 Sample vision applications

As a testbed for DiVA, we have transformed an object recognition system based on the fuzzy invariant indexing technique (FII) for object recognition [38] into a society of agents. FII lends itself to the recognition of all (man-made) objects with clearly distinguishable boundaries like lines, ellipses, squares, etc.** Its main advantages are a very high recognition rate and very high robustness with respect to perspective distortions, i.e. variations of the camera viewpoint. The agents have been implemented using standard *MagiC*-classes, and the knowledge as well as planning strategies of the agents have been modeled using the expert system tool *Clips 6.10* [36]. The agent society consists of six different agent types, four master agents and two slave agents, each one corresponding to a particular vision task:

1. *Master communicator agent:* The master communicator agent (Fig. 17, left)provides a graphical user interface, to monitor the agent society. It allows the user to specify different vision tasks and visualizes the results.
2. *Master/slave image processing agents:* Performs all of the high-level communication with the society and incorporates strategies for splitting up particular image processing tasks to be accomplished by a dynamic team of slave agents.
3. *Feature extraction agent:* The feature extraction agent is responsible for feature specific tasks, including extraction of edge points from images and fitting of geometric primitives like lines and ellipses.
4. *Master/slave object recognition agents:* The object recognition agents perform a recognition process based on FII, including the grouping of geometric primitives, invariant calculation, hypothesis generation and verification.

All types of agents were implemented for both Linux and Solaris platforms. Consider the simple example in section 6.2, where the system is to extract all ledges as well as all red objects shown in an image taken in our robot scenario. The system takes on a transient system structure as depicted in Fig. 17, right. The corresponding trace of the message passing is shown in Tab. 1. As indicated, the master object recognition agent is the only agent capable of recognizing objects in images. Although the object recognition agent has no knowledge about accessing cameras, the agent is awarded the contract. In order to solve the recognition task the agent needs geometric primitives (especially lines and ellipses) to be extracted from the image. Since the source specification does not match this requirement, the agent requests to extract the geometric primitives from the unknown source specifications (2). Next, a feature extraction agent is awarded the contract. Again, this agent needs the assistance of the agent society to detect the required edge points from the unknown source (3). This sub-task is solved by the master image processing agent. The agent grabs an image from the specified camera and asks its slaves to apply an edge operator (4, 5). Note that the communication between master and slave agents is very simple. Using the resulting edge image (6) the feature extraction agent extracts the particular geometric primitives and passes them to the master object recognition agent (7). Now the

**It is not limited to our toy element domain; on the contrary, it can be easily adapted to domains with structurally different objects. We are using this domain mainly for reasons of "benchmarking" against alternative approaches, because of its potential to construct arbitrarily complex scenes and because it lends itself perfectly to the combination with our robots (i.e. for sensor/actuator integration.

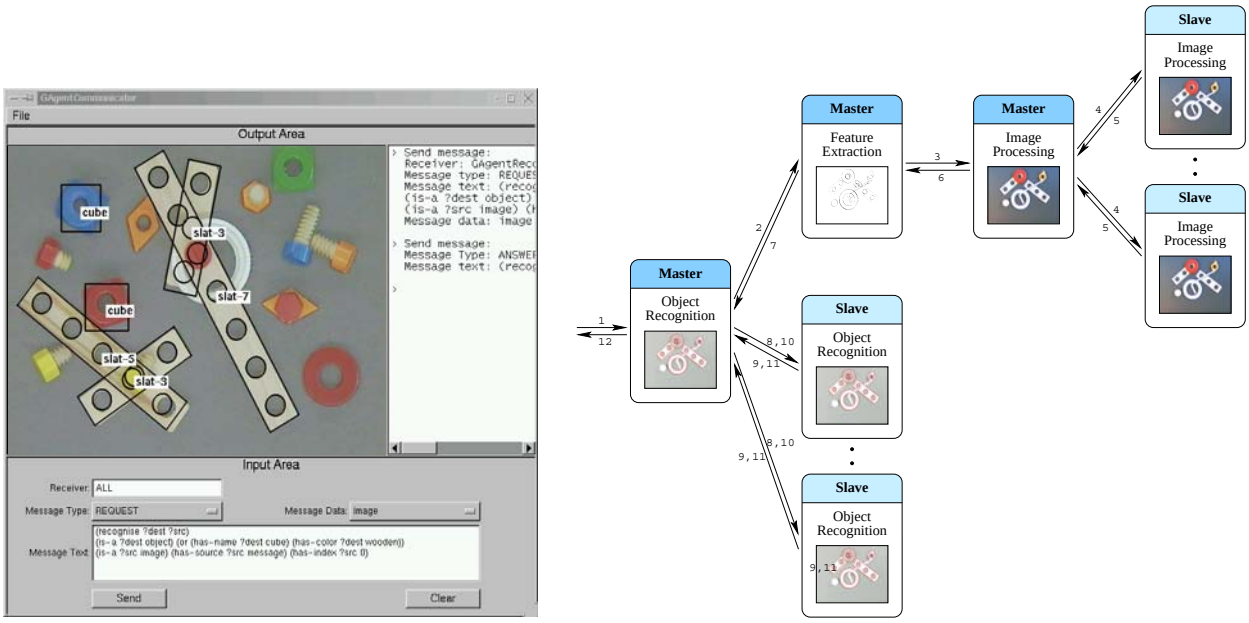


Figure 17. Left – Graphical user interface provided by the communicator agent. **Right** – Team structure resulting from self-organisation.

```
 1: REQUEST: (extract ?dest ?src)
             (is-a ?dest object)(or (has-name ?dest ledge)(has-color ?dest red))
             (is-a ?src image)(has-source ?src camera)(has-server ?src clinton)
 2: REQUEST: (extract ?dest ?src)
             (is-a ?dest feature)(or (has-type ?dest line)(has-type ?dest ellipse))
             (is-a ?src image)(has-source ?src camera)(has-server ?src clinton)
 3: REQUEST: (extract ?dest ?src)(is-a ?dest image)(has-type ?dest edge)
             (is-a ?src image)(has-source ?src camera)(has-server ?src clinton)
 4: REQUEST: (apply-canny)
 5: ANSWER:  (apply-canny)
 6: ANSWER:  (extract edge-image clinton-1)
 7: ANSWER:  (extract lines UNKNOWN-2)(extract ellipses UNKNOWN-2)
 8: REQUEST: (generate-hypotheses)
 9: ANSWER:  (generate-hypotheses)
10: REQUEST: (verify-single-hypotheses)
11: ANSWER:  (verify-single-hypotheses)
12: ANSWER:  (extract rim UNKNOWN-2)(extract ledge-3 UNKNOWN-2)
             (extract ledge-7 UNKNOWN-2)
```

Table 1. Trace of message passing.

master object recognition agent recognises the specified objects. This is done with the assistance of the slave object recognition agents, which perform the hypotheses generation as well as the verification of the hypotheses (`8-11`). Finally, the recognition task is accomplished (`12`).

An example of a simple program script, which has been automatically generated by the feature extraction agent during this recognition process, is shown in Table 2. It is subdivided into three different sections: in lines 01...04 the feature extraction agent requests an edge image from the agent society, in lines 05...10 all feature extraction functions are performed, i.e. the fitting of straight lines and ellipses, and finally, in lines 11...13, an answer is generated.

```
01: (bind ?var-gen7 (make-instance instance-gen9 of GMessage (type
     REQUEST)))
02: (send ?var-gen7 put-text "(extract ?dest ?src)(is-a ?dest image)
    (has-type ?dest edge)(is-a ?src image)(has-source ?src message)
    (has-index ?src 0)")
03: (send ?var-gen7 put-data (send [input-message] get-data))
04: (send-message (instance-name-to-symbol ?var-gen7))

05: (bind ?var-gen10 (send ?var-gen7 get-data 0))
06: (bind ?var-gen3 (extract-edge-points ?var-gen10 10))
07: (delete-data ?var-gen10)
08: (bind ?var-gen5 (extract-conics ?var-gen3 ellipse))
09: (bind ?var-gen2 (extract-lines ?var-gen3))
10: (delete-data ?var-gen3)

11: (send [output-message] put-type ANSWER)
12: (send [output-message] add-text "(extract lines UNKNOWN-2)(extract
    ellipses UNKNOWN-2)")
13: (send [output-message] add-data ?var-gen2 ?var-gen5)
```

Table 2. Automatically generated *Clips*-style program script

The recognition result of the task is shown in Fig. 18, where Fig. 18(a) is the original image taken by the specified camera, Fig. 18(b) shows the edge image provided by the image processing agents, Fig. 18(c) are the fitted features computed by the feature extraction agent, and Fig. 18(d) shows the final recognition result. The system recognise the objects that match the given object specifications, namely five 3-hole-ledges and two red rims. All other objects present in the image are ignored. The system fails to detect one red rim, a problem resulting from inaccurate feature extraction as can be seen in Fig. 18c. Note that these results are from a single top-view image with heavy mutual occlusion. Obviously, positive and negative false recognition results cannot always be completely avoided. To reduce the likelihood of this happening, we are currently developing an approach to integrate multi-viewpoint fusion into DiVA; section 6.5.2 gives a first example of the results obtained with this approach. Before briefly describing the potential of that approach, we present another example of object recognition from different viepoints with great perspective distortion.

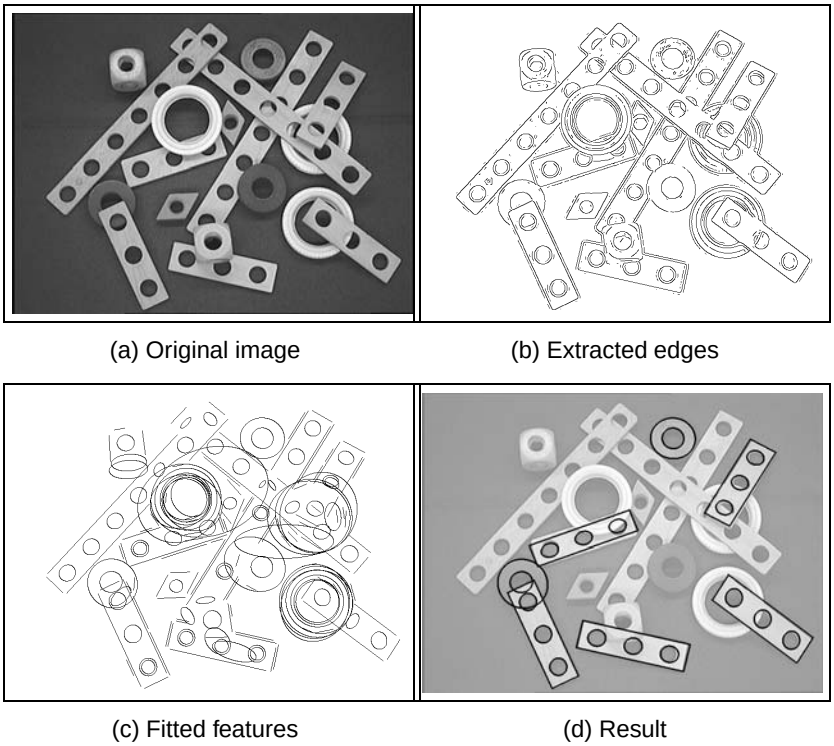


Figure 18. Recognition result of a test scene.

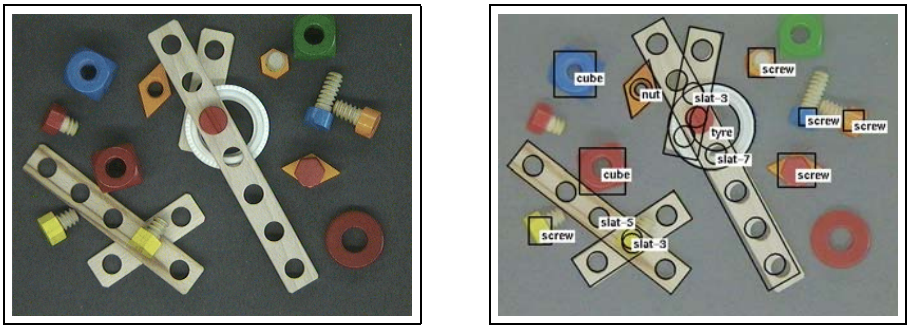
6.5.1 Results for Test Scenes Containing Partially Occluded Objects

As mentioned above, it is our goal to enable DiVA beyond being an easy-to-use scene and task adaptive distributed vision system (i) to fuse information from (dynamically changing[††]) sensor locations and (ii) to not only select the most useful information sources but to control the view points. The latter will be realised as a "trading negation" between agents to resolve the conflict between the time needed for moving the sensor and the expected information gain. In other words: to draw the maximum profit from information being available from different viewpoints in terms of the "fusion gain" (precision and completeness of the recognised enviroment) the agent society must be able to analyse sensor readings independently of their locations. One of the most important presupposition in the context of a distributed vision network is the ability to recognise objects independently of the viewing perspective. Figs. 19, 20, and 21 show that DiVA's agent society, thanks to its perspective-invariant recognition techniques, handles this task very robustly. The top-view images were taken at an angle of 90 degrees, the front-view at 45 degrees and the hand-camera images at about 75 degrees. Despite the high variations in object orientation (cf. the bolts), in viewing angle and in mutual occlusion, DiVA recognises nearly all objects with no interaction of a human operator required.

6.5.2 Occluded Objects and multiple viewpoint fusion.

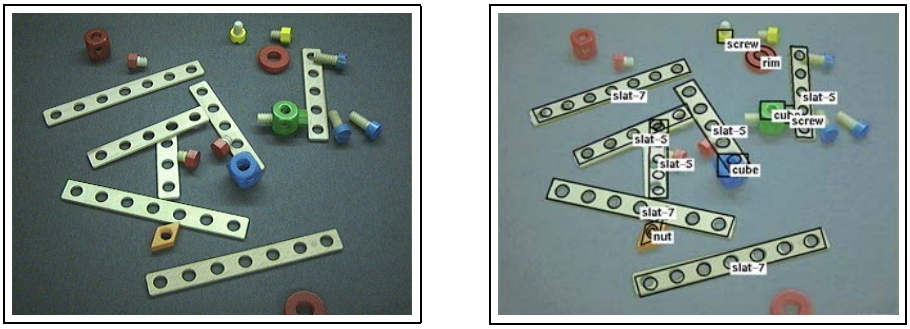
Fig. 22 shows one of our first results obtained for multiple-view fusion. In an iterated hypothesis-test process, the agent society agrees on the correct solution of the task to find all slats, independently of their number of holes. The two images are the sole source of information, the lower one is a magnified partial view taken under a slightly different angle with the hand

[††] Due to changes of sensor parameters like focus, aperture, zoom, orientation (as in the case of standard heads for active vision) or due to moving the sensor physically in a new location and thereby also changing its orientation (as is the case with our robot cameras mounted near the end effector.



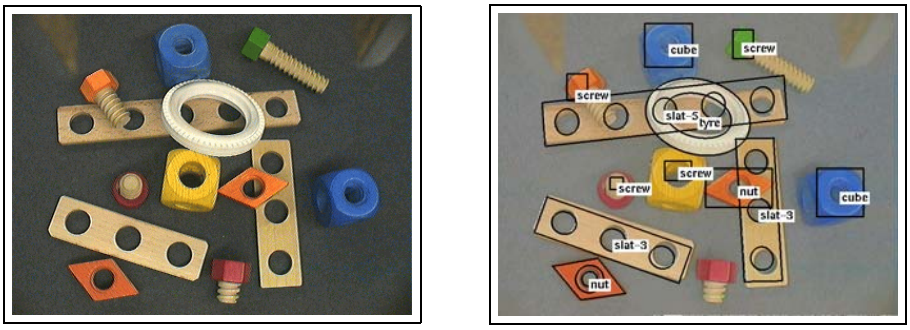
(a) Original image (b) Recognition result

Figure 19. DIVA result: occluded objects taken with a top-view camera.



(a) Original image (b) Recognition result

Figure 20. DIVA result: occluded objects taken with a front-view camera.



(a) Original image (b) Recognition result

Figure 21. DIVA result: occluded objects taken with a hand camera.

camera. Here, again, the negotiations between the agents solved the problem completely autonomously, up to now based on relatively simple heuristics, which will be extended to more sophisticated evaluation functions in the future, e.g. for assessing the informational utility of a certain view and the optimum fusion sequence (for the latter see e.g. [39]).

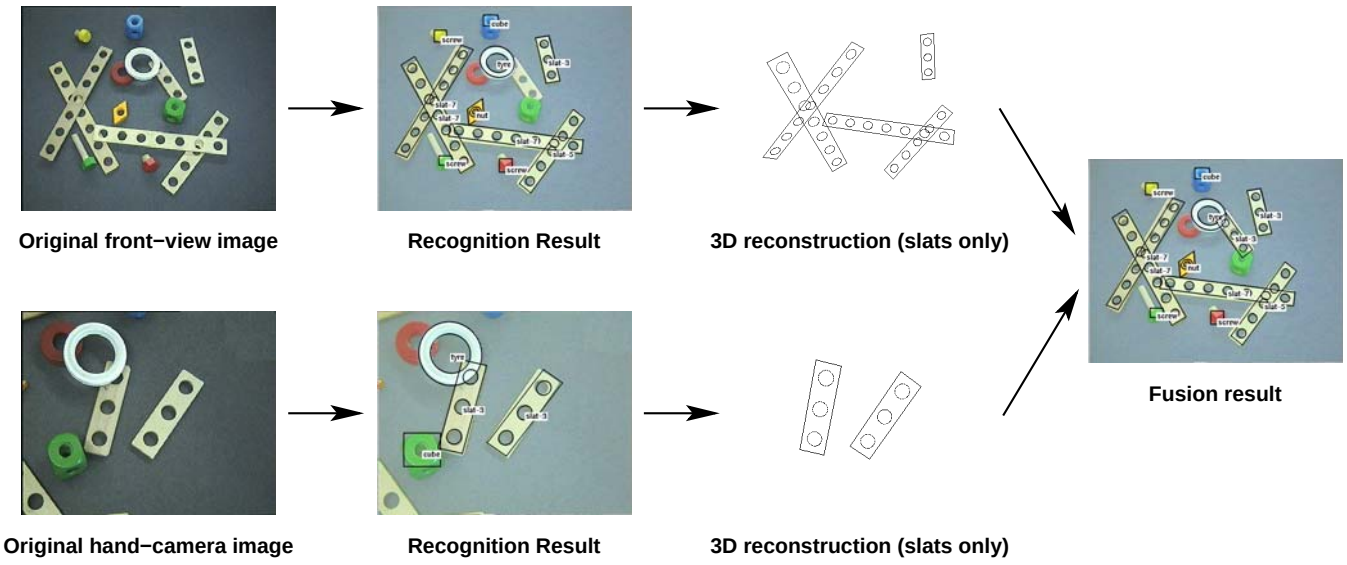


Figure 22. Results obtained from multiple viewpoints: robust recognition of all required objects through fusion on a symbolic object description level.

6.5.3 Conclusions

In this paper we outlined some aspects pertaining to the construction of distributed adaptive reconfigurable (sensor) agent networks. We proposed a model for self-organisation through the contract net protocol, which enables the formation of hierarchic and anarchic teams of interconnected agents. Through simulations it was shown that, with respect to throughput and fault tolerance, there is a general tendency of lateral networks (such as the proposed contract-net based agent societies) to have an advantage of the extended forms of hierarchies we investigated when network sizes increase, but at the cost of a slightly higher communication overhead. The comprehensive software system *MagiC* makes it very easy for programmers to rapidly implement adaptive networks of contracting agents; only the agent-specific procedures and heuristics have to be filled into predefined templates by the programmer. All the management necessary for handling the contracting phases etc. is provided by *MagiC*. It has been used in a variety of applications, and it turned out that, besides being a cross-platform run-time environment, it is also an excellent integration tool from a software engineering point of view. With *MagiC* as its software base, we developed DiVA, a distributed computer vision and fusion system, which is both task and environment adaptive through self-organisation of an agent society divided into two different agent classes. DiVA's control strategy is completely decentralised, and agents communicate in a flexible communication language. DiVA's recognition results are convincing, in particular with respect to occlusion, and its architecture will serve as a sound basis for the integration of further sensor types. Our current research focuses on enhancing the flexibilty and fusion capabilites of DiVA, on developing methods for optimal placements of sensors, the integration of different competitive vision methods and the automatic mutual programming of agents through the script language.

REFERENCES

1. E. Waltz and J. Llinas, *Multisensor Data Fusion*, Artech House, 1990.
2. D. Hall, *Mathematical Techniques in Multisensor Data Fusion*, Artech House, 1992.
3. F. Sadjadi, "Selected papers on sensor and data fusion," in *SPIE Milestone Series*, vol. MS 124, SPIE - Society of Photo-Optical Instrumentation Engineers, 1996.
4. R. Brooks and S. Iyengar, *Multi-Sensor Fusion: Fundamentals and Applications with Software*, Prentice-Hall, 1997.
5. V. Cantoni, V. D. Gesu, and A. Setti, eds., *Human and Machine Perception: Information Fusion*, Plenum Press, 1997.
6. I. Goodman, R. Mahler, and H. Nguyen, "Mathematics of data fusion," in *Theory and Decision Library. Series B, Mathematical and Statistical Methods*, vol. 37, Kluwer Academic, 1997.
7. N. Carver, V. Lesser, and Q. Long, "Distributed sensor interpretation: Modeling agent interpretations in DRESUN," tech. rep., University of Massachusetts, Amherst, 1993.
8. C. Tomlin, G. J. Pappas, and S. Sastry, "Conflict resolution for air traffic managment: a study in multi-agent hybrid systems," in *IEEE Conference on Decision and Control*, (San Diego), 1997.
9. C. Giraud and B. Jouvencel, "Sensor selection in a fusion process: a fuzzy approach," in *First IEEE Conf. on Multisensor Fusion and Integration for Intelligent Systems, MFI-94*, IEEE Press, 1994.
10. S. Iyengar, M. Sharma, and R. Kashyap, "Information routing and reliability issues in distributed sensor networks," *IEEE Trans. on Signal Processing* **40**, December 1992.
11. D. Erdmann, "Understanding action and sensing by designing action-based sensors," *Int. Journal of Robotics Research* **14, No. 5**, Oct. 1995.
12. G. Hager and M. Mintz, "Task-directed sensor data fusion and sensor planning," *Int. J. of Robotics Research* **10**(4), 1991.
13. Y. Demazeau, "From feature extraction to integration of visual modules using agent systems," in *Proc. Workshop on Solving Complex Problems with Multi-Agent Systems*, Univ. of Bielefeld, Feb. 1994.
14. T. Henderson, W. Fai, and C. Hansen, "MKS: A multisensor kernel system," *IEEE Trans. on Syst., Man, and Cybernetics* **SMC-14, No. 5**, 1984.
15. C. Fröhlich, F. Freyberger, G. Karl, and G. Schmidt, "Multisensor system for an autonomous robot vehicle," in *Int. Workshop Inform. Process. in Autonomous Mobile Robots*, G. Schmidt, ed., Springer-Verlag, 1991.
16. M. Dekhil and T. C. Henderson, "Instrumented sensor system architecture," *International Journal of Robotics Research* **17**, pp. 402–417, April 1998.
17. A. Mutambara, *Decentralized Estimation and Control for Multisensor Systems*, CRC Press, 1998.
18. B. Rao, H. Durrant-Whyte, and A. Sheen, "A fully decetralized multi-sensor system for tracking and surveilance," *Int. J. Robotics Research* **12**(1), 1991.
19. R. Smith and R. Davis, "Frameworks for cooperation in a distributed problem solver," *IEEE Trans. on Systems, Man and Cybernetics* **SMC-11**, 1981.
20. R. Davis and R. Smith, "Negotiation as a metaphor for distributed problem solving," *Artificial Intelligence* **20**(1), 1983.
21. A. Knoll, B. Hildebrandt, and J. Zhang, "Instructing cooperating assembly robots through situated dialogs in natural language," in *Proc. IEEE Conference on Robotics and Automation, Albuquerque, New Mexico*, 1997.
22. A. Knoll, "Die Erzeugung von Entfernungsbildern in der Robotik," Technische Universität Berlin, Jan. 1993. (in German).

23. E. Marchand, E. Rutten, H. Marchand, and F. Chaumette, “Specifying and verifying active vision-based robotic systems with the signal environment,” *International Journal of Robotics Research* **17**, pp. 418–432, April 1998.
24. M. Wooldridge and N. R. Jennings, “Agent theories, architectures and laguages: a survey,” in *Intelligent Agents*, M. Wooldridge and N. R. Jennings, eds., vol. 890 of *Lecture Notes in AI*, pp. 1–39, Springer–Verlag, 1995.
25. M. Wooldridge and N. R. Jennings, “Intelligent agents: theory and practice,” *Knowledge Engineering Review* **10**(2), pp. 115–152, 1995.
26. M. Shapiro, “Structure and encapsulation in distributed systems: The proxy principle,” in *6th International Conference on Distributed Computer Systems*, May 1986.
27. D. Harris, *Computer graphics and applications*, Chapman and Hall, 1984.
28. Y. Bar-Shalom and X. Li, *Estimation and Tracking*, Artech House, 1993.
29. C. Scheering and B. Kersting, “Uncalibrated hand-eye coordination with a redundant camera system,” in *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA-98)*, 1998.
30. A. C. Sanderson, L. E. Weiss, and C. P. Neumann, “Dynamic sensor-based control of robots with visual feedback,” *IEEE Trans. Robot. Automat.* **RA-3**, pp. 404–417, Oct. 1987.
31. C. Scheering and A. Knoll, “Local failure detection in a redundant camera system for visual manipulator guidance,” in *Proc. EuroFusion98*, October 1998.
32. O. Boissier and Y. Demazeau, “MAVI: a multi-agent system for visual integration,” in *Proc. IEEE Conference on Multi-sensor Fusion and Integration for Intelligent Systems, Las Vegas, Nevada, USA*, pp. 731–738, 1994.
33. O. Boissier and Y. Demazeau, “ASIC: An architecture for social and individual control and its application to computer vision,” in *Proc. European Workshop on Modeling Autonomous Agents in a Multi-Agent World*, pp. 107–118, 1994.
34. R. Bianchi and A. Rillo, “A purposive computer vision system: a multi-agent approach,” in *II IEEE Workshop on Cybernetic Vision 1996, Proc. IEEE Computer Society*, pp. 225–230, 1996.
35. K. Yanai and K. Deguchi, “An architecture of object recognition system for various images based on multi-agent,” in *Proc. International Conference on Pattern Recognition, Brisbane, Australia*, pp. 278–281, 1998.
36. Artificial Intelligence Section, Lyndon B. Johnson Space Center, *CLIPS Reference Guide, Volume I, Basic Programming Guide*, 1998.
37. http://www.cs.umbc.edu/kqml.
38. T. Graf, A. Knoll, and A. Wolfram, “Recognition of partially occluded objects through fuzzy invariant indexing,” in *Proc. IEEE International Conference on Fuzzy Systems, Anchorage, Alaska, USA*, pp. 1566–1571, 1998.
39. B. Dasarathy and S. Townsend, “FUSE – fusion utility sequence estimator,” in *Proc. 2nd Int. Conf. on Information Fusion*, Int. Society of Information Fusion, (Sunnyvale, CA), 1999.