A framework for implementing self-organising task-oriented multi-sensor networks

Christian Scheering and Alois Knoll

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

ABSTRACT

Advanced manipulation skills that enable cooperating robots in a work cell to recognise, handle and assemble arbitrarily placed objects require sensory information on both the environment and the assembly process. Using standard approaches it becomes difficult to coordinate sensor usage and data fusion under a given task when the number of sensors becomes large or is changed during run-time. We present a solution to both coordination and fusion based on the multi (sensor) agent paradigm: each agent implements a sensory skill, a negotiation protocol and a physical communication interface to all other agents. Within this sensor-network teams cooperate on a common task. They are formed dynamically after a negotiation phase following a specific task formulation. Our framework consists of a formal specification of the requirements that the sensory skill of an agent has to meet and a comprehensive library of (C++)-objects encapsulating all of the negotiation protocol and communications. This separation makes it very easy to implement individual sensory skills. We show how the abstract concepts of this approach and the metaphor of "negotiation" work in a real-world network: several uncalibrated cameras are used to guide a manipulator towards a target. We also show how agent-teams may easily (self-)reconfigure during task execution in the case of unexpected events. The framework is distributed free of charge and can be obtained over the Internet at `http://magic.uni-bielefeld.de`.

Keywords: multi-sensor network, distributed sensing, sensor-fusion, multi-agents, uncalibrated visual servoing, contract net

1. INTRODUCTION

Multisensor networks (MSN) with sensors in different locations and of different types for recording different parameters of an environment are becoming increasingly popular because they have a number of advantages over single sensor solutions:

- one sensor may compensate the deficiencies of another one in terms of accuracy, sensitivity to environmental conditions, noise rejection (usually in the same measurement domain but possibly with different physical sensors, e.g. range image generation with cameras and sonars);
- the failure of a single sensor does not result in the failure of the whole system; if properly designed, the system performance degrades gracefully, not catastrophically;
- a greater number of measurements taken from different locations may provide a more complete picture of the environment.

Controlling a heterogeneous network of interacting, potentially failing sensors is a challenging multi-faceted task. As of this writing only a limited number of approaches to designing such systems in a systematic way have been published in literature; only a subset of them have been implemented in real world applications. Specific issues of research include reconfiguration in homogeneous networks of sensors for multi-vehicle control,[1,2] sensor selection,[3] network architecture and communication between sensors.[4]

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,[5] an application of this modelling approach can be found in.[6] This is, 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 MSNs run in different setups, for evaluating the system performance based on sensor models, for monitoring or for debugging. Recently, a more advanced scheme was published,[7] which enables the designer to experiment with, simulate, debug and to continuously monitor the performance of different versions of a sensor system. All of these approaches and methodologies target static MSN, 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 sensor placement) was not considered. In our view, however, this is a very important issue for two reasons:

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

1. If there is a multitude of hardware sensors available, as for example the cameras in our assembly cell, [8] then the sensors can be utilised for performing more than one class of tasks, e.g. optical detection, recognition, tracking of objects using monocular, binocular, trinocular techniques with articulated or fixed sensors.

2. Given both an appropriate modelling of the sensors, the environment and, consequently, the results to be expected, it is possible to specify tasks with only very few parameters and leave it to the system to configure itself so as to obtain the best result in terms of accuracy, speed, etc.

From both the software and the system engineering point of view it is desirable to split the design of a system as complex as a MSN into different layers of abstraction.

a) 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.

b) 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.

c) 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.

d) A *communication protocol* associated with a physical communication system through which the logical sensors may exchange raw data with each other or with a mediator external to the network.

e) A *cooperation protocol* specifying which logical sensor may interact with which other one under a certain task. It implicitly defines the architecture of the MSN (i.e. it is derived from the architecture).

Layers a) and b) define the private properties of the sensor while d) and e) are provided by the embedding system. Layer c) is the interface connecting them. The physical sensor together with the software modules for processing its data and implementing the cooperation and communication protocols is called a *sensor agent*. It is important to mention 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.

We therefore consider it important to develop experimental software platforms that enable researchers to efficiently combine many physical sensors and fusion methods into an operational system that can handle diverse tasks with only a minimum of input about the sensing strategy, sensor team forming and other administrative information. To this end we introduce the notion of Contracting Multisensor Networks (CMN), which stands for our methodology for easily designing multisensor systems for a wide range of applications (see our early theoretical work [9]). It is our goal to make the CMN work with only a minimum of knowledge about the individual sensor and to endow it with enough intelligence (and henceforth computing power) to perform all the necessary configuration and parameter adaptation at run-time. In other words: to specify a task for the sensor network only four parameters need be passed to the CMN:

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 while the result must possess in terms of object/environmental parameters (e.g. length, orientation, distance, etc.);

Deadline: maximum time until task completion.

Currently, these task definitions are flat, i.e. they cannot be nested. It seems to be promising, however, to allow such definitions to be made recursively. If the aforementioned specifications are provided in a coherent form using a common formalism, e.g. a synchronous functional language [10] 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:

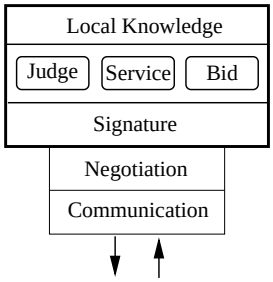


Figure 1. Structure of Agents in *MagiC*.

- Groups of sensors with different physical measurement principles can be combined into one MSN.
- 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.
- Logical sensor specifications can be easily written (templates are provided).
- Task specifications are simple and human-readable.

The software system we describe in the following sections implements a complete CMN development environment. It is the result of a long term research effort, it is fully operational and it can be obtained freely for academic use at `http://magic.uni-bielefeld.de`.

2. MAGIC – *M*ULTI *A*GENT *G*ENERATION *IN C++*

The *MagiC*-library builds on C++ as its supporting language. This decision was taken 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 an agency* [11] in which *autonomy, social ability, reactiveness* and *pro-reactiveness* 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 basic idea of *MagiC* is to provide a C++-class with all communication and cooperation capabilities necessary for implementing CMNs. 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 in the introduction. The most important features of *MagiC* agents are

- Agents are reactive, proactive or both;
- No distinction is made between local or remote agents (virtual agent space);
- Agents may call other (teams of) agents to provide services to the outside world, these services may be elementary or very complex;
- More than one sensory task may be handled at a time when agents are available.

Cooperation between agents is "contracted by negotiation" using the well-known contract-net protocol [12](CNP). An agent offering and distributing tasks to be executed is called a *manager*, and an agent capable of executing the task is called a *contractor*. Agents can take on these roles dynamically depending on their own private state. In *MagiC* it is possible to have more than one contractor for a task or a group of agents (the latter is called a *team*).

2.1. Agent objects

Fig. 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 (offers) of other agents in the network. If their signature matches with that of the task, then the agent's bid-method `make_bid` is called automatically. Within this bid the agent is able to self-assess the quality of its own bid and compare 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 yet available). If the agent designer did not provide the agent with its own version of `make_bid`, a default bid is made if the signature matches, indicating that this agent can in principle perform the task. Proactive agents may also take on the role of the manager. For a manager there are four ways of selecting contractors:

- first-come, first-serve: this is the default;
- requested quality: only those contractors bid for the task that meet a certain result quality criterion (existence, time, precision, etc.); if more than one replies, *MagiC* automatically selects the best by comparing the quality;
- received quality: the manager does not request a certain quality; the one replying with the highest quality is selected;
- selection method: if the agent designer defines a `select_contractors`-method, all incoming bids are evaluated by this method.

Due to the fact that C++ cannot transfer objects between processes (as is possible e.g. in Java or Modula-3) a special base class `TBaseObject` was defined which supplies this feature for every class derived from it.

2.2. Sensory Tasks

The execution of tasks follows simple rules. The manager constructs a *task description* containing the actual task parameter, expiration time, intended quality and task signature. Each description is derived from the `task_c` base class. By default, the new description class definition must contain a SIGNATURE statement. Agents whose SIGNATURE statement matches that of the manager are potential contractors. This statement is used during task announcement to determine the task signature automatically. A new instance is created by calling the C++-constructor:

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

This implemented type of task specification differs slightly from the one described in the introduction in that the action-specification in *MagiC* is realised by the SIGNATURE-statement contained in the task description. After announcing the task by broadcasting the description, the manager waits for contractors, selects the best contractor agent(s) according to its selection methods and waits for a result or a failure. This is all done in just two lines of code by calling the agent's inherited method `announce`:

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

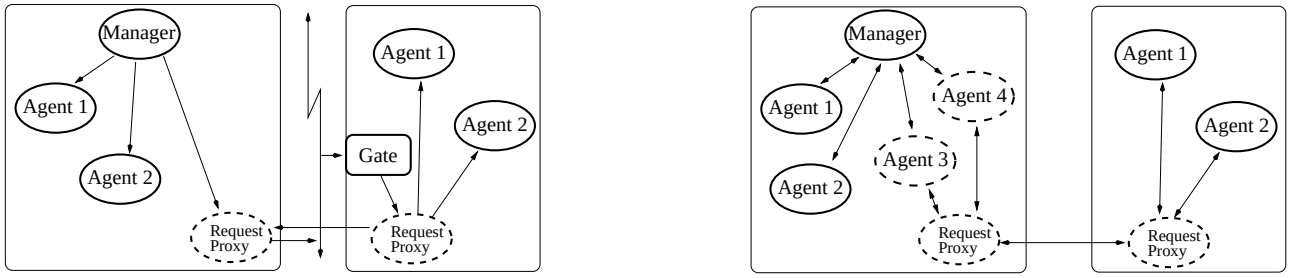
in which `result` is a pointer to a result list and `task` the appropriate description.

2.3. Teams

Apart from the possibility finding appropriate contractors with single task announcements, *MagiC* provides the designer of an agency with the facility of requesting *teams* of agents to which a task is announced iteratively. This is very useful for tasks that require a set of contractors. One advantage of a team is that a task may automatically be split into subtasks without announcing these subtasks individually. Another advantage is that all contractors are 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. In *MagiC* a team offers additional facilities for dealing with single member failures. After receiving the results from all team members or in case of an unexpected event (such as an agent is not answering at all or too slowly) the virtual method `reconfigure_team` is called in which the agent designer can 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.

2.4. Virtual Agent Space

Technically speaking each *MagiC* agent is modelled as a *thread* inside a process. By employing the proxy principle, [13] all agents regardless of their physical location (in the same process, on the same computer or on a remote computer in a network) are addressed in the same manner simply by announcing a task. This generates the illusion of a *virtual agent space* containing all agents. Fig. 2 shows the interaction between agents hosted inside two processes on different computers. Beside the manager



(a) Initialising proxies (b) Virtual local agents

Figure 2. Virtual agent space using proxy-agents.

agent there are two other local agents $Agent_1$ and $Agent_2$ in the same process. On 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. 2(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. 2(b)). The proxy in the announcing process appears as a contractor to the manager and the local agents of the remote process are contractors to their local proxy. This gives the manager (rather: the designer) the aforementioned illusion of "seeing" every contractor in its local process. It is also possible to create so-called *concurrent services*, e.g. to maintain a shared single resource as a frame-grabber device or for the above mentioned gate agents.

3. SAMPLE TASK: MULTI-SENSOR VISUAL GUIDANCE

In this section we show how to visually guide a 6 dof manipulator to a desired target with a set of redundant arbitrarily positioned uncalibrated cameras and how easily we modelled this as a CMN using *MagiC*.

3.1. Principle

The task is to position a 6 dof manipulator holding an object over a desired target (Fig. 3) by utilising several uncalibrated cameras. The key idea of our visual control approach is to assume a parallel camera model for the image forming process, to define an image-based position error in j different views and to construct a simple linear equation from the parallel projection camera-model. The latter is called the *fusion equation* and is used for a resulting Cartesian path correction movement. The parameters of this equation are estimated with a linear Kalman filter (KF) using measurements obtained by the cameras in different locations.

(a) (b)

Figure 3. Initial (a) and final position (b) of the manipulator and its projected trajectory as seen from one camera.

3.1.1. Fusion equation

The parallel projection $\boldsymbol{P}^j$ (see[14]) 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 plane is

$$\boldsymbol{f^j} = \begin{pmatrix} r_{11}^j & r_{12}^j & r_{13}^j & t_1^j \\ r_{21}^j & r_{22}^j & r_{23}^j & t_2^j \end{pmatrix} \cdot \boldsymbol{m}^w = (\boldsymbol{R}^j\ \boldsymbol{t}^j) \cdot \boldsymbol{m}^w = \boldsymbol{P}^j \cdot \boldsymbol{m}^w \tag{1}$$

Assuming that both the target and the manipulator can 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 $\boldsymbol{g}$ results from defining an appropriate error-displacement vector $\Delta\boldsymbol{d_e}$ which is to be minimised:

$$\Delta\boldsymbol{d_e} = \boldsymbol{m} - \boldsymbol{g} \rightarrow 0 \tag{2}$$

For the corresponding displacement feature $\Delta\boldsymbol{f}_e^j$ in the j^{th} camera using Equation (1) a simple linear relationship follows:

$$\begin{aligned} \Delta\boldsymbol{f}_e^j &= \boldsymbol{f}_m^j - \boldsymbol{f}_g^j \\ &= \boldsymbol{P}^j \cdot \boldsymbol{m}^w - \boldsymbol{P}^j \cdot \boldsymbol{g}^w \\ &= \boldsymbol{R}^j \cdot \boldsymbol{m} + \boldsymbol{t}^j - \boldsymbol{R}^j \cdot \boldsymbol{g} - \boldsymbol{t}^j \\ &= \boldsymbol{R}^j \cdot \Delta\boldsymbol{d_e} \end{aligned} \tag{3}$$

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:

$$\boldsymbol{d}_e = \sum_{i=1}^{3} \xi_i \boldsymbol{d}_i, \text{ with } \xi_{1,2,3} \in R \tag{4}$$

Using $\boldsymbol{R}^j$, the projected version of Equation (4) is:

$$\boldsymbol{f}_e^j = \boldsymbol{R}^j \cdot \boldsymbol{d}_e = \boldsymbol{R}^j \cdot \sum_{i=1}^{3} \xi_i \boldsymbol{d}_i = \sum_{i=1}^{3} \xi_i \cdot \boldsymbol{R}^j \cdot \boldsymbol{d}_i = \sum_{i=1}^{3} \xi_i \boldsymbol{f}_i^j \tag{5}$$

Hence under this assumption $\boldsymbol{f}_e$ is a linear-combination of the projected base using the *same* $\boldsymbol{\xi}$ as in 3D space. Calculating an appropriate set of scalars ξ_1, ξ_2, ξ_3 in the image space and inserting them into Equation (4) leads directly to the desired displacement-vector in the Cartesian 3D space. Unfortunately, eq. (5) is under determined. Therefore at least two views are

* Since only displacements are considered, the Δ is omitted in the sequel.

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:

$$\underbrace{\begin{pmatrix} \boldsymbol{f}_e^1 \\ \boldsymbol{f}_e^2 \\ \vdots \\ \boldsymbol{f}_e^j \end{pmatrix}}_{\boldsymbol{z}} = \underbrace{\begin{pmatrix} \boldsymbol{f}_1^1 & \boldsymbol{f}_2^1 & \boldsymbol{f}_3^1 \\ \boldsymbol{f}_1^2 & \boldsymbol{f}_2^2 & \boldsymbol{f}_3^2 \\ \vdots & \vdots & \vdots \\ \boldsymbol{f}_1^j & \boldsymbol{f}_2^j & \boldsymbol{f}_3^j \end{pmatrix}}_{\boldsymbol{H}} \cdot \underbrace{\begin{pmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \end{pmatrix}}_{\boldsymbol{\xi}} \tag{6}$$

Eq. (6) is the aforementioned *fusion equation*, which plays the central role in our approach. Only three parameters need to be estimated (independent of the number of cameras) and only three initial test movements are necessary for this purpose. $\boldsymbol{H}$ is determined simply by measuring the projection of each test move in all the images.

3.1.2. Solving the fusion equation

We use a linear discrete Kalman filter to solve for the parameters of eq. (6). Assuming zero-mean, white-noise $\mathbf{v}$ and $\mathbf{w}$, the plant and measurement equation are:

$$\begin{aligned} \xi(k+1) &= \xi(k) + \mathbf{v}, & \mathbf{v} &\sim N(0, \mathbf{Q}) \\ \mathbf{z}(k) &= H(k) \cdot \xi(k) + \mathbf{w}, & \mathbf{w} &\sim N(0, \mathbf{R}) \end{aligned} \tag{7}$$

The incremental prediction and update solutions can be found in the literature.[15] In our approach the whole system dynamics is included in the system noise $\mathbf{v}$. We have chosen pure diagonal matrices for $\mathbf{Q}$, $\mathbf{R}$ and the initial state covariance $\mathbf{P}_{(0|0)}$ 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 in a first step the projection $\boldsymbol{f}_i^j$ of the manipulator during the three Cartesian test moves are obtained. With the measured position residuals an initial down-scaled correction movement $\boldsymbol{d}_c = s \cdot \boldsymbol{d}_e, \in (0,1]$ is then calculated. After each movement a new $\boldsymbol{\xi}$ is estimated. This is iterated until the target is reached (dynamic look-and-move). Results from simulations and real experiments showing the accuracy of the proposed method can be found in our previous work.[16]

3.2. Modelling the task in a CMN

Our method of visually guiding a manipulator with a set of redundant uncalibrated cameras fits nicely into the *MagiC* framework and can be easily modelled as a CMN. From each camera only the position-residual $\boldsymbol{f}_e^j$ between the goal and the manipulator needs to be known in eq. (6). Therefore we can model each camera as a *track agent* which measures the local positions and sends them back as a result of a task announcement made by a "top-manager" which in turn solves the parameters of the fusion-equation and controls the robot. Each track-agent decides to make a bid depending on whether it sees both the target and the manipulator (whether those are in the image is currently determined by a human operator but will be automated soon). On the manager side a `reconfigure`-method enables the manager in case of failure to reconfigure his team autonomously (for the full class specification see the appendix).

4. REAL-WORLD EXPERIMENTS

As mentioned in the introduction, the objective of a CMN with a redundant set of sensors is not only to increase the accuracy but also the robustness and fault-tolerance when compared with single sensor solutions. We performed three experiments to show that the above described CMN modelled with *MagiC* is self-organising with respect to the selection of tracking agents but that it also tolerates the complete failure of some contractors. The agency-setup for all experiments in this section is shown in Fig. 4. It is comparatively simple in that it contains one pure manager agent **R** announcing tracking tasks and up to four different track agents $\mathbf{TA}_{1,2,3,4}$, each in its own process. During the experiment only three frame-grabbers were available. However, each grabber has two input channels allowing two cameras to be connected. Therefore, the track agents are forced to share access to them by announcing grabbing-tasks to up to four grab-agents $\mathbf{GA}_{1,2,3,4}$. These grab-agents run on computers hosting the frame-grabbers and are modelled as dynamic team agents. Despite the necessity to share at least one grabber between two requesting track agents, the illusion of exclusive access to a certain channel of the grabber is maintained. Tracking the robot is achieved through simple template matching. Both target selection and template initialisation are provided interactively by a human operator during the execution of the `make_bid` methods of the track agents.

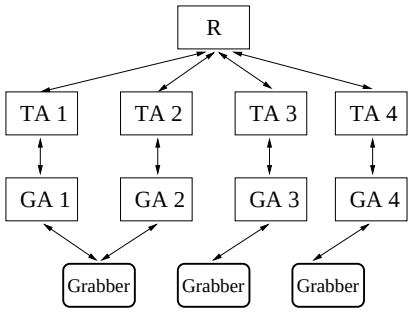


Figure 4. Modelled agency for the experimental setup.

4.1. Redundant visual guidance

The first task in the experiment was to position the robot carrying a yellow wooden cube above a desired target (also a cube) using as many contractors as possible. Fig. 5 shows the final camera images of the experiment, the projected trajectory after completion of the task and the final search window for the template-matching. The target to reach was the position above a "tower" consisting of two other cubes. After 29 iterations the target was reached successfully. Due to measurement errors near the white computer monitor (middle left in Fig. 5 (a)) the performed trajectory was not a straight line (Fig. 5 (a) and (c)). Another reason for slightly perturbed trajectories is the assumption of a linear relationship between 2D and 3D space in eq. (3).

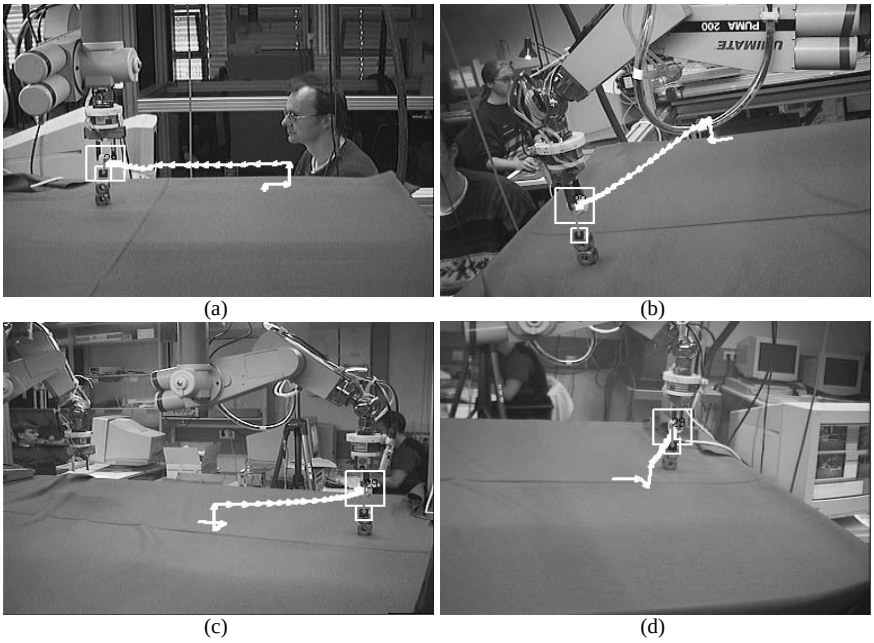


Figure 5. First experimental task: final robot positions, projected trajectories including initial test moves, size of template matching window. Images are seen by the four cameras used in this experiment.

As with all linear approximations (e.g. the image Jacobian[17]) this assumption holds only at the position where the relationship was established. The farther away the robot moves from this position, the higher the errors. Due to the dynamic look-and-move scheme used here these errors are corrected perfectly.

4.2. Contractor failure

The second experiment uses the same setup but with failing agents during task execution. Figs. 6 (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. As a

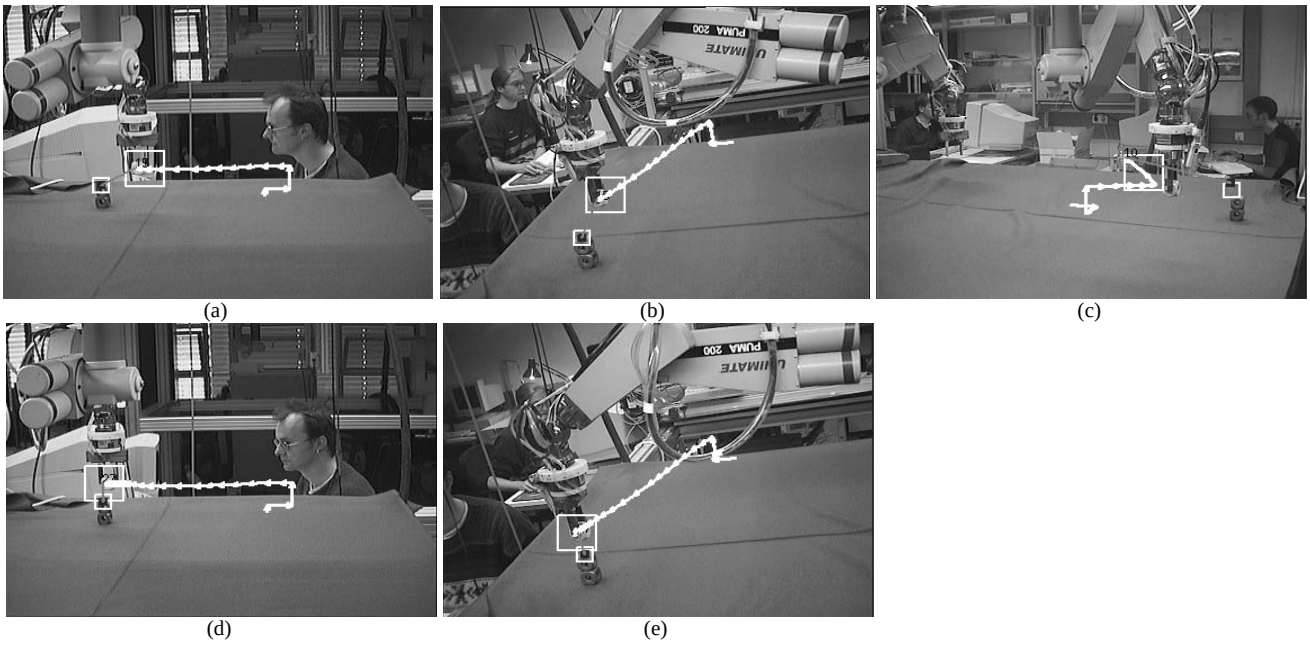


Figure 6. Second experimental task: the target is still reached despite several failures.

consequence the failing agent, whose tracking window is shown in Fig. 6(c) loses contact to the robot and tracks something else. This kind of failure (which may be detected with methods described in our previous work [18]) forced the agent in this experiment to terminate itself. Nevertheless, there exist another two active agents which guide the robot successfully further to the target (Figs. 6(d) and (e)).

4.3. Reconfiguration

The third experiment finally shows the CMN's capability for reorganisation. During task execution only one contractor survives, which results in the CMN reannouncing the task automatically. Figs. 7(a) to (c) show the situation after reannouncing the task. The image in Fig. 7(a) comes from the agent that was a team member and shows its path travelled so far (including the new test moves). Figs. 7(d) to (f) show the resulting trajectories of the robot after reaching the target. However, the new automatic test moves perturbed the robot's position from where the failure occurred. This leads to a change in the trajectory followed by the robot in order to reach the desired target. For instance, the robot must now move upward (Fig. 7(e)). Along the way this illustrates the capability of our method to control the robot in three dimensions.

5. CONCLUSIONS

We presented the *MagiC* framework for modelling self-organising CMN. We showed a small portion of its operational power by some simple examples of visual manipulator guidance and its behaviour under unexpected events like partial sensor-agent failures. Future work will concentrate on two issues: building more applications using *MagiC* and enhancing the system by integrating more sophisticated monitoring and debugging facilities.

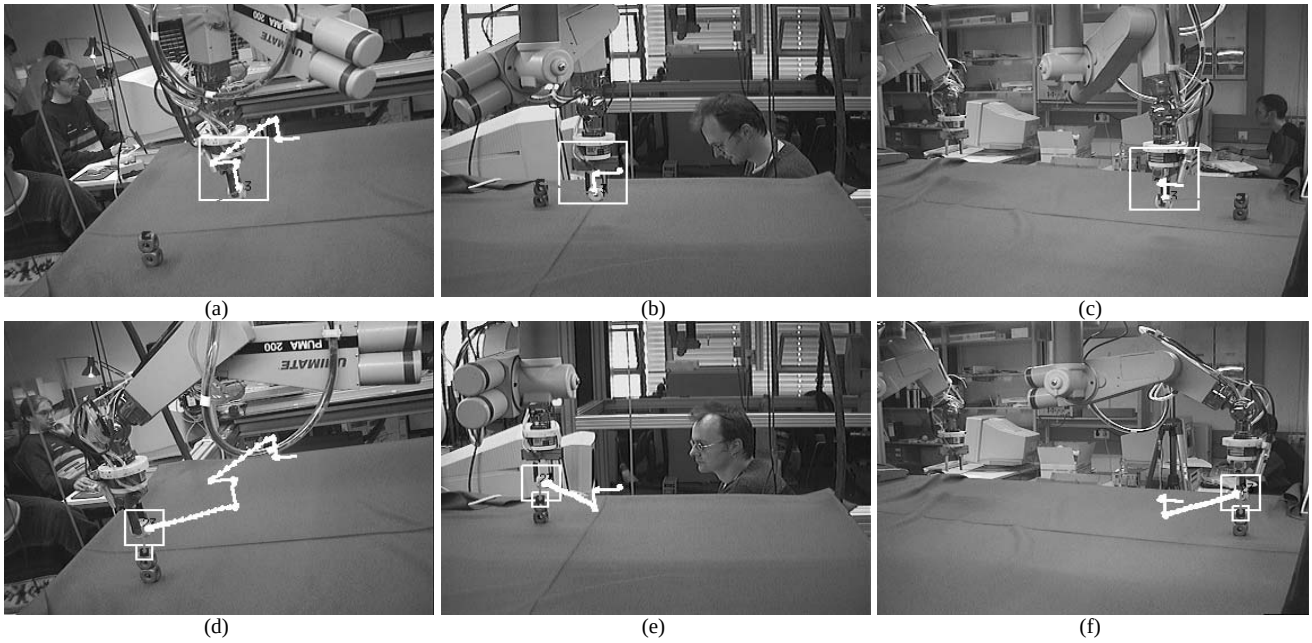


Figure 7. Third experimental task: automatic reconfiguration.

APPENDIX A. EXAMPLE CLASS DEFINITIONS

```
class tpoints : public TBaseObject {
public:
    TBaseLong ox, oy, mx, my;
    TBaseLong subteam_id;
    tpoints() {ox = oy = mx = my = 0L, subteam_id = -1L;}
};

class track_request : public TBaseObject {
public:
    TBaseLong init;
    track_request() {init = 1L;}
};

class track_agent : public team_agent_c {
private:
    void      init(void);
    bid_c    *make_bid(TBaseObject *parameter, quality_c *required);
    int       find_target_and_manipulator(tpoints *, track_request *);
public:
    virtual TBaseObject *service(TBaseObject *);

    SIGNATURE(track_agent, track_request, tpoints);
};

class track_task : public team_task_c {
public:
    track_task(track_request *request, quality_c *required, double expiration) {
        timeout   = expiration;
        parameter = request;
        quality   = required;
    };
```

```
    SIGNATURE(track_agent, track_request, tpoints);
};

class track_team : public team_c {
private:
    void         init(void);
    virtual void reconfigure_team(team_task_c *act_task);
public:
    void perform_a_target_approach(void);
};

bid_c *track_agent::make_bid(TBaseObject *parameter, quality_c *required) {
    bid_c *bid = new bid_c;

    track_request *request = (track_request *)parameter;

    // required quality ignored!
    if (!find_target_and_manipulator(result, request))
        bid->quality = 0.0;
    else
        bid->quality = 1.0;

    return bid;
};

TBaseObject *track_agent::service(TBaseObject *parameter) {
    track_request *request = (track_request *)parameter;
    tpoints       *result;

    result->subteam_id = team_id.sub_team_id; // inherited from team_agent_c
    if (request->init != 0) init();
    if (!find_target_and_manipulator(result, request))
        exit(-1);

    return result;
};

void track_team::reconfigure_team(team_task_c *act_task) {
    track_task *task = (track_task *)act_task;

    if (!task->teams_ok) {                // inherited from team_task_c
        while (noofmembers < 2) {         // inherited from team_c
            release_team(task);           // release of all team members working on task
            task->request->init = 1L;
            request_team(task);
            task->request->init = 0L;
            perform_a_target_approach(); // again!
        }
    }
}

// simplified main of a track_agent
int main(int argc, char *argv[]) {
    track_agent trakki;

    trakki.init();    // init and connect to a grab-agent
    trakki.standby(); // wait for announcements
};
```

```
// simplified main of manager
int main(int argc, char *argv[]) {
    track_team     manager;
    track_request  request;
    track_task    *task   ;

    task   = new track_task(request, NULL, 5.0);
    manager.init();
    manager.request_team(task);
    manager.announce    (task);
    manager.release_team(task);
};
```

ACKNOWLEDGMENTS

The work presented in this paper has been funded by the German Research Foundation (DFG) in the collaborative research afford SFB 360, project D4.

REFERENCES

1. N. Carver, V. Lesser, and Q. Long, "Distributed sensor interpretation: Modeling agent interpretations in dresun," tech. rep., University of Massachusetts, Amherst, 1993.
2. 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.
3. 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.
4. S. Iyengar, M. Sharma, and R. Kashyap, "Information routing and reliability issues in distributed sensor networks," *IEEE Trans. on Signal Processing* **40**, December 1992.
5. T. Henderson, W. Fai, and C. Hansen, "MKS: A multisensor kernel system," *IEEE Trans. on Syst., Man, and Cybernetics* **SMC-14, No. 5**, 1984.
6. 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.
7. M. Dekhil and T. C. Henderson, "Instrumented sensor system architecture," *International Journal of Robotics Research* **17**, pp. 402–417, April 1998.
8. A. Knoll, B. Hildebrandt, and J. Zhang, "Instructing cooperating assembly robots through situated dialogues in natural language," in *Proc. Int. Conf. on Robotics and Automation (ICRA-97)*, 1997.
9. J. Meinkoehn and A. Knoll, "A model of non-hierarchical control in distributed sensor-networks," in *Proceedings of SPIE/OE Technology 1992, Sensor Fusion V*, vol. 1828, The International Society for Optical Engineering, (Boston), November 1992.
10. 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.
11. M. Wooldridge and N. R. Jennings, "Intelligent agents: theory and practice," *Knowledge Engineering Review* **10**(2), pp. 115–152, 1995.
12. R. G. Smith, *Distributed Problem Solving*, UMI Research Press, 1981.
13. M. Shapiro, "Structure and encapsulation in distributed systems: The proxy principle," in *6th International Conference on Distributed Computer Systems*, May 1986.
14. D. Harris, *Computer graphics and applications*, Chapman and Hall, 1984.
15. Y. Bar-Shalom and X. Li, *Estimation and Tracking*, Artech House, 1993.
16. C.Scheering and B.Kersting, "Uncalibrated hand-eye coordination with a redundant camera system," in *Proc. Int. Conf. on Robotics and Automation (ICRA-98)*, 1998.
17. 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.
18. C. Scheering and A. Knoll, "Local failure detection in a redundant camera system for visual manipulator guidance," in *Proc. EuroFusion98*, October 1998. to appear.