International Journal on Artificial Intelligence Tools
Vol. 0, No. 0 (2000) 000—000


Publisher

# A MULTI-AGENT SYSTEM ARCHITECTURE FOR DISTRIBUTED COMPUTER VISION

THORSTEN GRAF and ALOIS KNOLL
*University of Bielefeld, Faculty of Technology*
*P.O.Box 10 01 31, D-33501 Bielefeld, Germany*
*E-mail: {graf, knoll}@techfak.uni-bielefeld.de*

Received (received date)
Revised (revised date)

We present a multi-agent system architecture dedicated to model computer vision systems, which provides the vision system with a great degree of flexibility. The basic idea of this architecture is to model a vision system as a society of autonomous agents, where each agent is responsible for specific vision tasks, the control strategy of a vision system is decentralized, and agents communicate using a flexible but easy understandable communication language. This directly leads to self-organizing vision systems, which accomplish vision tasks by goal-driven communication processes. We describe in detail the basic concepts of the proposed multi-agent system approach including the agent architectures, the communication language and network, as well as the interaction strategies. As a testbed for the proposed architecture we have modeled an object recognition system as an assembly of agents which organize themselves according to a given recognition task by employing communication.

*Keywords*: multi-agent architecture, computer vision, object recognition

## 1. Introduction

The capability of observing the world based on visual information is an essential requirement in robotic applications with increasing importance since tasks handled within robotic scenarios are getting more complex and fewer restrictions are imposed to the environmental conditions.

Therefore, modern complex robotic applications impose several requirements to a vision system and its architecture: Firstly, it must be simple to use the vision system in complex setups; secondly, the vision system must be able to adapt dynamically to different (possibly changing) tasks and environmental conditions; thirdly, the architecture must provide the ability to handle different competitive information; and lastly, the addition of new processing modules must be possible without rebuilding the complete system. Conventional vision systems generally fol-

low system architectures which make it difficult to comply with these requirements.

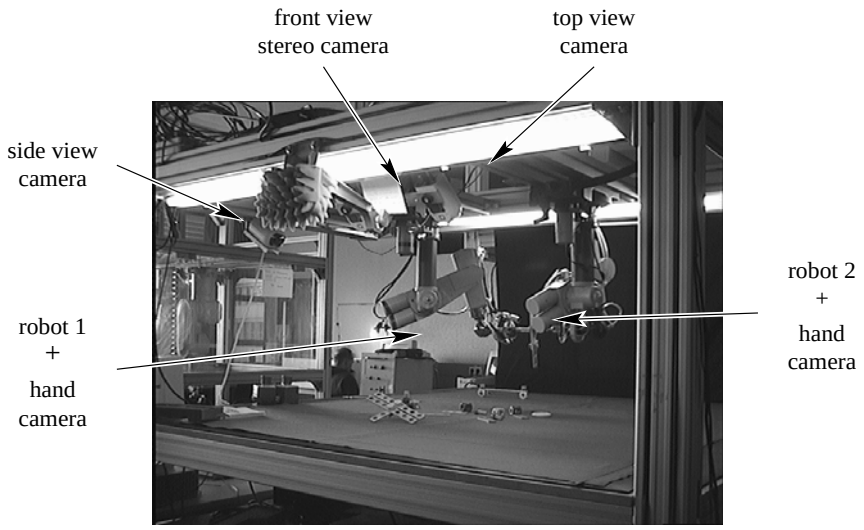



Fig. 1. Complex robotic setup

An example for such a robotic setup is shown in Fig. 1, which is part of the research project SFB 360 "Situated Artificial Communicators" supported by the German Research Foundation (DFG). The goal is to develop an intelligent assembly cell which is controlled and assisted by a human instructor using natural spoken language. The particular assembly task of the system is to build up complex toy objects, such as airplanes, using a construction kit (see Knoll *et al*[1] for additional information about the aim of the project). To accomplish a construction task, the assembly cell is equipped with several cameras to solve the various sub-tasks that may arise during the assembly process, like the recognition and localization of objects, 3D scene reconstruction, and visual servoing.

Recent research has indicated that modeling vision systems as societies of autonomous agents is a promising and powerful approach to tackle such objectives: Boissier and Demazeau have proposed the MAVI[2] system, a multi-agent system for visual integration, which is based on the ASIC[3] multi-agent control architecture. This architecture is subdivided into different processing layers which is too complex for most vision applications and make it difficult to handle the system. Following the purposive vision paradigms Bianchi and Rillo[4] have inspired a multi-agent vision system employing a behavior based decomposition in specific tasks, while Yanai and Deguchi[5] 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.

We therefore propose a multi-agent system architecture which provides the vision system with a great degree of flexibility. This architecture simplifies the generation of complex self-organizing vision systems. In Sect. 2 we explain in detail the basic concepts of the architecture including the structure of agents (Sect. 2.1), the communication language (Sect. 2.2) and network (Sect. 2.3), and the interaction strategy (Sect. 2.4). In Sect. 3 we demonstrate an implementation of the architecture in a distributed object recognition system, which organizes itself according to given recognition tasks. Finally, in Sect. 4 our conclusions and directions of possible future research are presented.

## 2. Multi-Agent System Architecture

In the proposed multi-agent system architecture a computer vision system is modeled as a society of agents, each one responsible for specific vision tasks. Since many of the computer vision algorithms are very time-consuming the architecture provides two different classes of agents: master agents and slave agents. The former accomplish all of the planning and interpretation as well as many of the required image processing tasks. The later are responsible to assist master agents in performing time-consuming tasks only, where the slave agents generally work in teams controlled by corresponding master agents to provide the distributed calculation of sub-results.

In the society the agents are connected to each other using a contract net, whose topology is that of a completely connected network. The agents generally have to communicate in order to achieve their goals. Since the capability to communicate is an essential property of agents, the structure of the communication language has a great impact on the flexibility and applicability of the whole system. Therefore, we incorporate a communication language, which on the one hand can be simply employed, i.e. human readable and writable, and which on the other hand is flexible enough to describe also complex facts and tasks.

In the following we explain in detail the multi-agent system architecture including the structures of master and slave agents, the communication language and network, and the interaction strategy.

### 2.1. *Agent Architectures*

Since master and slave agents are responsible for different purposes within a computer vision system they differ in their architectures as well.

#### 2.1.1. *Master Agent*

Master agents are modeled as autonomous agents, which generally must perform complex planning and interpretation tasks. To enable master agents to accomplish these tasks adequately, they share the agent architecture sketched in Fig. 2. As shown, the architecture is composed of five different modules:

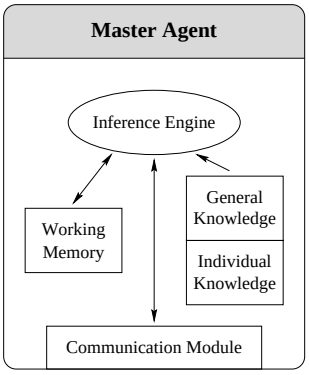



Fig. 2. Architecture of a master agent

1. *Communication module:*
   The communication module is responsible for connecting to other agents. It contains methods for sending and receiving messages as well as functions for wrapping various data types, which are required to solve the particular tasks to which the agent belongs.

2. *General knowledge:*
   This data base is used for storing general knowledge, like basic planning strategies and the grammar of the communication language. Most of the knowledge is stored in rules and facts that are applicable in different situations and determines the general behavior of master agents.

3. *Individual knowledge:*
   The individual knowledge base contains all knowledge concerning the particular agent, like specific planning strategies, processing functions and the provided vocabulary. Note, that it is not required that all agents share the same vocabulary. Similar to the general knowledge base the individual knowledge is stored in rules and in facts and determines the individual behavior of each agent.

4. *Inference engine:*
   Based on general and individual knowledge the inference engine accomplishes all planning and all interpretation tasks of the agent. The most extensive work that must be performed by the inference engine is to generate program scripts appropriate to solve requested tasks of other agents.

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

In order to perform a particular vision task, the responsible agent proceeds as follows: Firstly, the agent analyzes the given task including the instruction, the destination specifications and additional constraints. However, it is generally not required that the agent understand all of the source specifications. According to the description of the task the agent automatically generates a complex *Clips*[6]-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 the experimental results in Tab. 3. Lastly, 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.

#### 2.1.2. *Slave Agent*

Although a computer vision system can be modeled by employing just master agents, the multi-agent system architecture provides an additional slave agent class. Since many of the image processing algorithms are very time-consuming, the slave agent class is provided as a an additional tool that facilitates the implementation of parallel processing methods to speed up computation.

The particular purpose of slave agents is simply to assist master agents in performing time-consuming tasks, where the slave agents often can communicate with their corresponding masters only. Slave agents are completely controlled by master agents, i.e. the master decides how many slave agents he wants to use as well as which particular processing function must be performed. The communication between a master and its slaves is reduced to the absolute minimum. Generally, a master agent determines only the processing functions and additional parameters.

Therefore, slave agents need only a simple architecture containing a communication module, processing functions and rudimentary mechanisms for interpreting messages.

### 2.2. *Communication Language*

As already mentioned, the communication language has a great impact on the flexibility and applicability of the whole system. However, the syntax of communication languages used in the computer vision domain generally tends to be cryptic and too simple to express complex facts and tasks; this is true especially for that part of the communication language which is relevant for the application itself.

Therefore, we have developed a more flexible communication language considering the following basic requirements:

- the communication language must permit the construction of flexible and self-organizing vision systems,
- it must be able to express complex facts and tasks,
- it must be simple to understand, i.e. human readable and writable, and
- efficient mechanisms for interpreting messages must be provided.

In contrast to other communication languages, which just permits messages composed of function names and parameters, the developed one can be utilized to specify a task by an abstract description which is independent of the underlying implementation. Therefore, agents can be simply added and replaced without having any knowledge of the internal structures of the existing agents.

The developed communication language employs message types making the intention of an message explicit:

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

The allowed message types are similar to the ones used in other communication languages[2,4] in that they implement speech acts but differ in some important respects:

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

2. *answer:*
   An `answer` message is a reply to a request or script message, which can be both a result or an error message.

3. *inform:*
   The message type `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:*
   The message type `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 capable for storing different data types like images and edges. The text string itself can be both a message text or a *Clips*[6]-style program script.

An excerpt of the formal grammar used for specifying a message text is shown in Tab. 1. For reasons of efficiency this grammar allows only one goal for each message, where a goal is not a specific entity but rather some kind of global plan, which can be restricted by additional conditions. These conditions can be complex logical expressions containing variables as well.

Table 1. Formal grammar of messages

```
<message-text>   ::= (<goal> <variable>*) <goal-condition>*

<goal>           ::= convert | extract | introduce | ...
<goal-condition> ::= <attr-condition> | <not-condition> |
                     <and-condition>  | <or-condition>

<attr-condition> ::= <is-a-attr> | <has-type-attr> | ...
<not-condition>  ::= (not <goal-condition>)
<and-condition>  ::= (and <goal-condition>+)
<or-condition>   ::= (or <goal-condition>+)

<is-a-attr>      ::= (is-a <variable> <object-class>)

<object-class>   ::= feature | image | ...
```

A simple example for an object recognition task, which possibly must be solved during an assembly process in the robotic application shown in Fig. 1, is to extract all ledges as well as all known red objects shown in an image taken from a camera, whose server is called 'penelope'. This task can easily 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 penelope)
```

There are two important points to note here: Firstly, the interpretation of such messages can be realized in a very efficient manner using the pattern-matching facilities of expert system tools; and secondly, 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.

### 2.3. *Communication Network*

The agents of the society are connected with each other through a decentralized contract net, which topology is that of a completely connected network. However, since slave agents are implemented to assist master agents in performing time-consuming tasks, they often communicate and interact with their corresponding master agents only.

An example for a communication network is schematically sketched in Fig. 3. The example demonstrates three different types of connections among master agents and their corresponding slave agents, which often occur during a communication process. The first type consists of a single master agent, which is able to perform all required processing functions on its own, while the second type is composed of a master agent which divides time-consuming sub-tasks in order to be solved by its corresponding slaves. The third type consists of two (or more) master agents, which are responsible for the same tasks. Again, they divide sub-tasks in order to be accomplished by their slaves, where the slaves can be dynamically shared between the master agents.

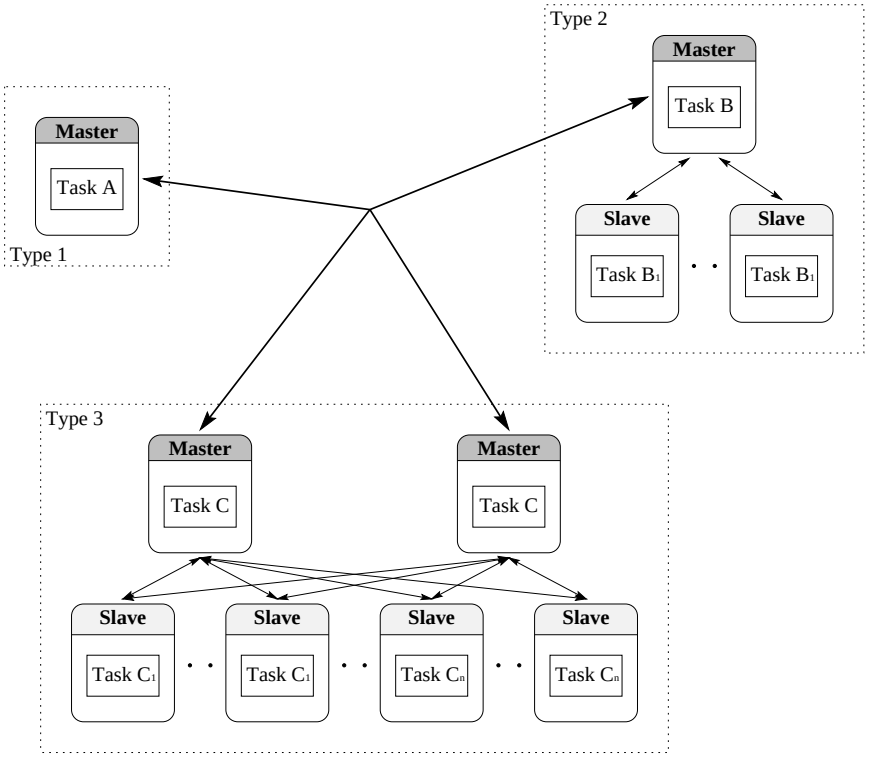



Fig. 3. Example for a communication network

It is important to note here, that the hierarchical structure between master and slave agents is not pre-defined in a statical way, but is rather a result of communication processes. This structure may dynamically change, if additional agents are instantiated or if existing agents are deleted at run-time. Moreover, it is also possible that two different types of master agents share the same slave agents because some sub-tasks are the same or very similar.

Moreover, to provide a high degree of flexibility, pre-defined hierarchical structures should be avoided wherever it is possible, since any imposed hierarchical structure will generally make it difficult to add, remove or exchange agents. For example in the agent architecture proposed by Biachni and Rillo[4] the modification of an agent or of a behavior may affect other agents, which entails the modification of these agents too.

### 2.4. *Interaction Strategies*

Another important aspect concerning the flexibility of the agent society is the interaction strategy among master agents. In our multi-agent system architecture the control mechanisms of computer vision systems are decentralized, i.e. each master agent accomplishes requested tasks according to its own knowledge and goals, where generally no further control mechanisms, like hierarchical structuring, should be imposed.

Therefore, the agents have to interact with each other in order to solve a given vision task. The interaction is performed by a communication process that leads to a self-organization of the agent society. This self-organization process is goal-driven and proceeds as follows:

- If a master agent requests a vision task, all master agents of the society decide if they can accomplish the given task. As mentioned before, this is done by analyzing the instruction, the destination specifications, and the additional constraints. Generally, the source specifications of the task are neglected.

- All master agents that are responsible for the particular task make a bid. According to these bids the master agent, which has requested the task, selects the agents that should award the contract.

- Then, the selected agents generate appropriate program scripts according to the task specifications. If the source of the task is unknown the selected agents generates further requests to gain a required sub-result determined from the unknown source.

There are some important points to note here: A drawback of this interaction strategy is that it can be established just at run-time if the vision system can accomplish a particular vision task. Furthermore, the interaction strategy produces some overhead because all master agents try to react to a request. Nevertheless, this overhead can be neglected since most of the vision algorithms are generally very time-consuming compared with the overhead.

The advantage of this approach is the flexibility of the resulting vision system: agents can be added and deleted at run-time without causing any problems. Furthermore, new functionality can be provided by simply adding appropriate agents without having precise knowledge of the internal structure of existing agents.

## 3. Experimental Results

As a testbed for the proposed multi-agent system architecture we have transformed an object recognition system based on the fuzzy invariant technique[7,8] into a society of agents. The agents have been implemented in C++ using the multi-agent generation tool *MagiC*[9]. This tool provides classes for building different types of agents and mechanisms for encapsulating all of the negotiation protocols and communication. The knowledge as well as planning strategies of the agents have been modeled using the expert system tool *Clips 6.10*[6].

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 provides a graphical user interface, to gain access to the agent society. It allows the user to specify different vision tasks and visualizes the results.

2. *Master/slave image processing agents:*
   Since most of the image processing algorithms, like convolution and edge detection, are very time consuming, we build both classes of agents, a master image processing agent as well as a slave image processing agent. The master agent performs all of the high-level communication with the society and incorporates strategies for splitting up particular image processing tasks in order 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 the fuzzy invariant indexing technique [7]. This process consists of: grouping of geometric primitives, invariant calculation, hypothesis generation and verification.

   Similar to the image processing agents the master agent establishes all of the communication and planning strategies while the slaves are responsible for the execution of particular recognition tasks.

We have run a number of agents of each type on different platforms including Linux-PCs and Sun-Solaris-Workstations. Although we have not imposed any pre-defined hierarchical structure on the system, the agent society is capable of solving complex vision tasks.

For example, if we request the task given in Sect. 2.2, the system takes on a transient system structure as sketched in Fig. 4. The corresponding trace of

message passing is shown in Tab. 2. 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) extracted in an 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 (see Fig. 3a) 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 master object recognition agent is able to recognize 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).

Table 2. Trace of message passing

```
 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 penelope)

 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 penelope)

 3: REQUEST: (extract ?dest ?src)(is-a ?dest image)(has-type ?dest edge)
             (is-a ?src image)(has-source ?src camera)(has-server ?src penelope)

 4: REQUEST: (apply-canny)

 5: ANSWER:  (apply-canny)

 6: ANSWER:  (extract edge-image penelope-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)
```

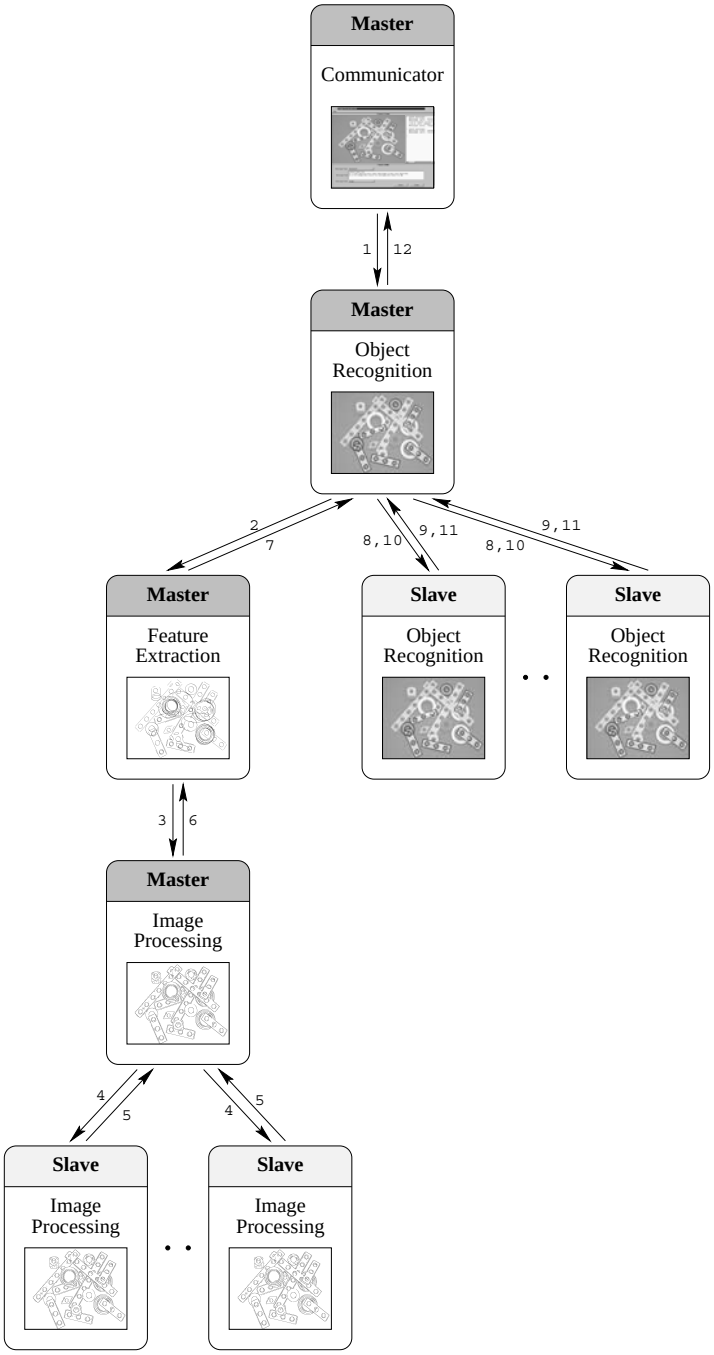



Fig. 4. Self-organized system structure

An example of a simple program script, which is automatically generated by the feature extraction agent during this recognition process, is shown in Tab. 3. 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.

Table 3. Automatically generated *Clips*-style program script

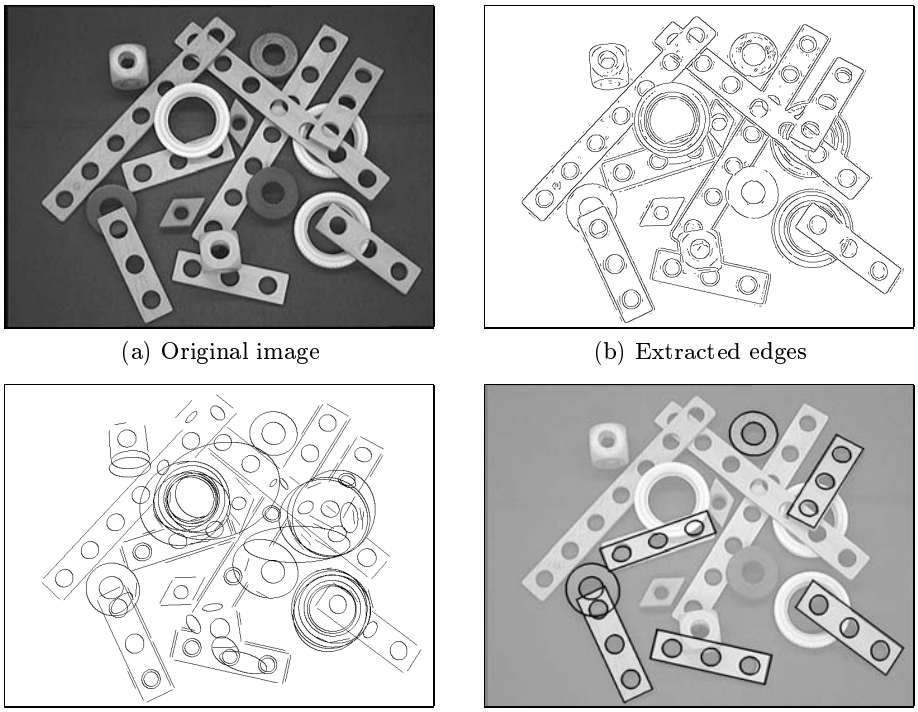
```
01: (bind ?var-gen7 (make-instance instance-gen9 of GMessage (type
     REQUEST)))
02: (send ?var-gen7 put-text "(extract ?dest ?src)(is-a ?dest image)
    (has-type ?dest edge)(is-a ?src image)(has-source ?src message)
    (has-index ?src 0)")
03: (send ?var-gen7 put-data (send [input-message] get-data))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

The recognition result of the task is shown in Fig. 5, where Fig. 5a is the original image taken by the specified camera, Fig. 5b shows the edge image provided by the image processing agents, Fig. 5c are the fitted features computed by the feature extraction agent, and Fig. 5d shows the final recognition result. As can be seen, the multi-agent system architecture is able to recognize 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. Unfortunately, the system fails to detect one red rim, where this deficiency is not a problem of the agent architecture but rather of an inaccurate feature extraction as can be seen in Fig. 5c.

The addition or deletion of agents at run-time causes no problems of system stability (assuming that all agents necessary to accomplish a task are available). These behaviors are an important factor for both the autonomy as well as the flexibility of the system, i.e. to improve the recognition capability of the vision system new agents can be added.



(a) Original image (b) Extracted edges

(c) Fitted features (d) Result

Fig. 5. Recognition result for a test scene

## 4. Conclusions and Future Research

We have presented a multi-agent system architecture dedicated to build flexible self-organizing computer vision systems, which solve given vision tasks by goal-driven communication processes. The architecture is based on the idea to model a vision system as a society of autonomous agents, where each agent is responsible for specific vision tasks, the control strategy of a vision system is decentralized, and agents communicate using a flexible but easy understandable communication language. As demonstrated, this architecture has several distinguishing features such as flexibility, modularity, autonomy and openness.

Future research will concentrate on several aspects that may further enhance the flexibilty and applicability of a vision system. Two main goals of this investigation are the integration of different competitive recognition methods to provide redundancy and to enhance the recognition performance as well as the integration of different learning strategies.

**Acknowledgment**

T. Graf's contribution to this work was in part funded by the Deutsche Forschungsgemeinschaft within the postgraduate research unit "Aufgabenorientierte Kommunikation" (task-oriented communication).

**References**

[1] A. Knoll and B. Hildebrandt, and J. Zhang, *Instructing cooperating assembly robots through situated dialogs in natural language*, Proc. IEEE Conference on Robotics and Automation, Albuquerque, New Mexico (1997)

[2] O. Boissier and Y. Demazeau, *Mavi: a multi-agent system for visual integration*, Proc. IEEE Conference on Multisensor Fusion and Integration for Intelligent Systems, Las Vegas, Nevada, USA (1994) 731–738

[3] O. Boissier and Y. Demazeau, *Asic: An architecture for social and individual control and its application to computer vision*, Proc. European Workshop on Modeling Autonomous Agents in a Multi-Agent World (1994) 107–118

[4] R. Bianchi and A. Rillo, *A purposive computer vision system: a multi-agent approach*, Workshop on Cybernetic Vision 1996, Proc. IEEE Computer Society (1997) 225–230

[5] K. Yanai and K. Deguchi. *An architecture of object recognition systems for various images based on multi-agent*, Proc. International conference on Pattern Recognition, Brisbane, Australia (1998) 278–281

[6] Artificial Intelligence Section, *CLIPS Reference Guide Volume I Basic Programming Guide*, Lyndon B. Johnson Space Center (1998)

[7] T. Graf, A. Knoll, and A. Wolfram, *Recognition of partially occluded objects through fuzzy invariant indexing*, Proc. IEEE International Conference on Fuzzy Systems, Anchorage, Alaska, USA (1998) 1566–1571

[8] T. Graf, A. Knoll, and A. Wolfram, *Fuzzy invariant indexing: a general indexing scheme for occluded object recognition*, Proc. International Conference on Signal Processing, Beijing, China (1998) 908–911

[9] C. Scheering and A. Knoll. *Framework for implementing self-organized task-oriented multisensor guidance*, Proc. International Symposium on Intelligence Systems and Advanced Manufacturing, Boston, USA (1998)