Vision Planner for an Intelligent Multisensory Vision System

X. Y. Jiang, H. Bunke
Institut für Informatik und angewandte Mathematik
Universität Bern, Länggassstrasse 51, 3012 Bern, Switzerland

Abstract

In this paper we present a multisensory vision system that is intended to support the vision requirements of an intelligent robot system. Contrary to many other vision systems, our system has two significant new features. First, it contains multiple sensors, object representations, image analysis and interpretation methods in order to solve a number of different vision tasks. Secondly, it comprises a vision planner. Upon a task-level vision request from the robot system, the vision planner transforms it into appropriate sequences of concrete vision operations, executes these operations, and if necessary, finds out alternative strategies. Experimental results demonstrate the clear advantage of this combination of multiple resources with the vision planner in solving typical vision problems for robotic tasks.

CR Categories and Subject Descriptors: I.2.8 [Problem Solving, Control Methods, and Search]: Plan execution, formation, generation; I.2.10 [Vision and Scene Understanding]: Architecture and control structures.
Additional Key Words: Robot vision, multisensory vision system, planning.

1 Introduction

Robots have large potential in industrial production. Properly used, they can free humans from harmful, strenuous, or dull jobs, reduce the manufacturing cost of products, and improve their quality. In spite of the strong social and economic incentives, however, only a very small fraction of the entire human work force in the world has been replaced by industrial robots. Moreover, robots that are used in most industrial applications today merely carry out complex movements in a structured environment by programmed control. Any unexpected event, say a workpiece not being in its programmed position, would lead to a manupulation failure. It is thus commonly agreed that robots of the next generation, although not required to act or look like a human, should be able to perform tasks that need *artificial intelligence* and *flexibility* [14].

Artificial intelligence means here the ability of a robot to perceive the actual situation it is faced with which may not be known a priori, to decide what actions to be done, and to plan these actions accordingly. The sequences of actions to be performed by the effectors of the robot are thus not preprogrammed. Instead, they are dynamically determined depending on the current situation. In such an intelligent robot system the use of a sensory subsystem as shown in Figure 1 is indispensable. The sensory system supplies the robot system with an initial high-level symbolic scene description for some robotic task, say assembly. Later on the sensory system performs, upon request of the robot system, particular vision tasks as an aid to monitoring and to further refining the robotic task.

Besides artificial intelligence, an intelligent robot system is also required to possess some degree of flexibility. This means the ability to perform a number of different tasks. Robot functions can be classified into active and passive tasks. Active tasks require the action of the robot effectors. Some examples are grasp, move, align, insert, sort, assemble, walk, etc. Passive tasks include recognize, locate, verify, inspect, etc. that are usually done passively, although sometimes manipulation is needed. In an unstructured environment the success of an active task is crucially dependent on the information from some passive tasks that are usually carried out by the sensory subsystem. The robot function assemble, for example, involves the following sequence of subtasks: *position-sensor, recognize, locate, grasp, move, align, insert*, and *verify*. The two vision subtasks *recognize* and *locate* supply the robot system with the current situation of the work cell. Only after they have been finished, the actual assembly action can begin. Thus a number of different tasks an intelligent robot system should be able to perform implies a number of different vision tasks. Some examples of such vision tasks are the determination of the location of an object of a certain type in the actual scene, the identification and localization of the topmost object in a pile of objects, the verfication if a 3-D path is free of obstacles, etc. The successful execution of these vision tasks represents a precondition for any active robotic task in an unstructured environment.

Object recognition is a difficult problem. Like other problems in pattern recognition

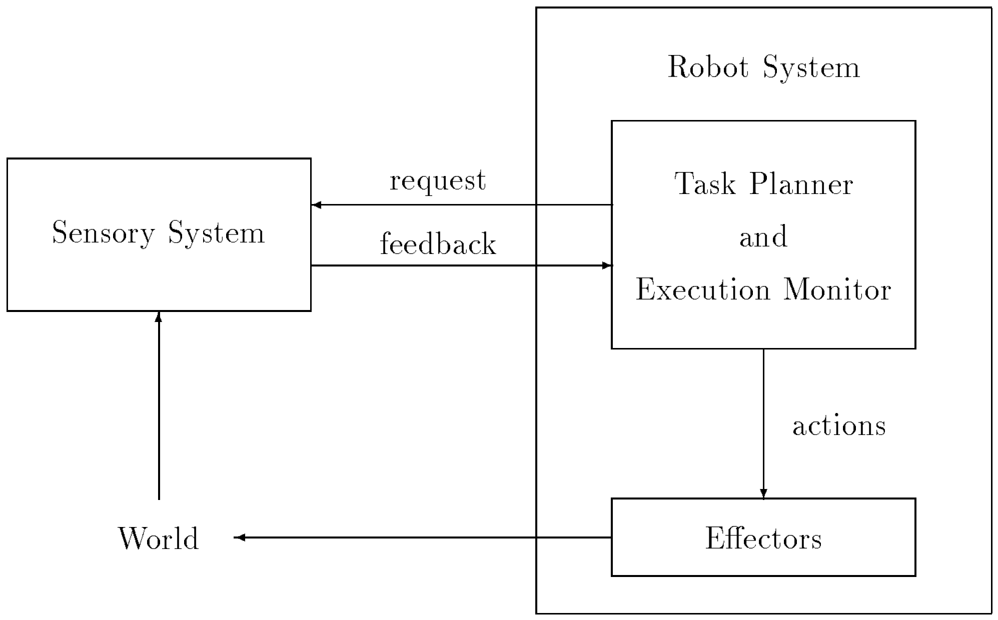


Figure 1: The sensory system and its integration in an intelligent robot system.

and computer vision, it is far away from being solved in terms of straightforward solutions to a given recognition task. Instead, plenty of object recognition methods have been proposed in the literature [1, 2, 8, 18]. They differ from each other in sensory data type (intensity image, range data, $\cdots$), object type they can handle (polyhedra, curved objects, $\cdots$), scene complexity (objects are isolated, touching, or overlapping), scene and model features used, matching strategies, etc. Each recognition method has its strength and its limitations. There is no best method for all possible recognition problems. Even for a particular recognition task, say the localization of a known polyhedron, there is often no consensus on the optimal recognition strategy. Since the vision system for an intelligent robot system is required to solve a number of quite different vision tasks, and these vision tasks may dynamically arise during the execution of some other task, for example assembly, the only design choice is therefore a system with a rich set of sensors, object representations, and recognition strategies.

In our multisensory vision system we use two sensors, a CCD camera and a range sensor, to acquire intensity and range images. In each step of the image data analysis and interpretation process, multiple methods are available in the vision system in order to solve one out of a number of different vision tasks. In such a system, however, the control problem becomes essential. Upon a request from the robot system, the vision system must be able to figure out an appropriate sequence of vision operations, trigger and monitor their execution, and in case of failure, find out alternative strategies. In our vision system we call this meta-level control component the *vision planner*.

The rest of this paper is organized as follows. First, an overview of our multisensory vision system is given, including a short description of the modules in the current

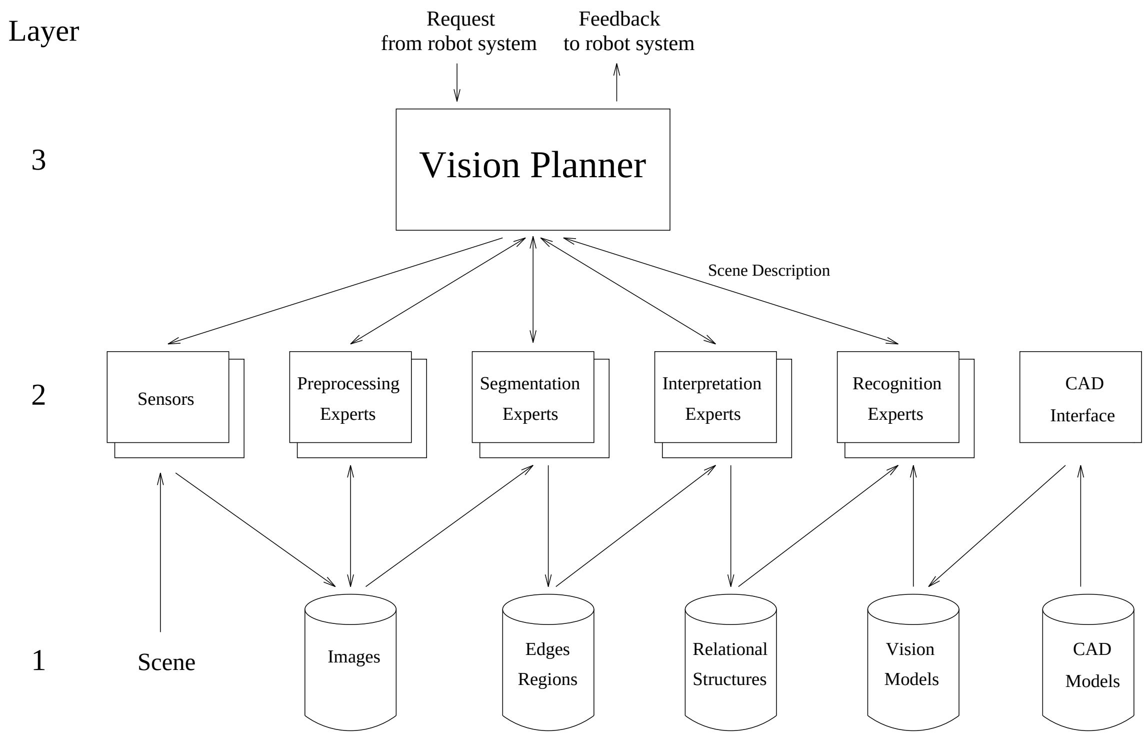


Figure 2: Overview of the multisensory vision system. Layer 1: data structures; layer 2: sensors and software modules; layer 3: vision planner.

prototype. Then, the vision planner is described in Section 3. Some implementation issues are given in Section 4, followed by experimental results in Section 5. Finally, some discussions conclude this paper.

2 Multisensory vision system

An overview of our multisensory vision system is shown in Figure 2. It contains three layers: data structures, sensors and software modules, and the vision planner. The so-called experts in the layer of sensors and software modules perform specialized tasks in the vision system. These vision tasks include image acquisition, image preprocessing, edge- and region-based segmentation, interpretation based on general, object independent knowledge for feature grouping and noise elimination, recognition methods, and a CAD-interface for automatic generation of vision models from CAD-models. In each processing step, several method are available in the system. The decision which method is actually used is made by the vision planner dependent on the particular request from the robot system.

Appropriate data structures are used to store results from the sensors, the software modules, and the CAD-interface. The entities to be stored include intensity and range images, enhanced images, segmented images, attributed relational scene descriptions, and vision models. Except the vision models, all these data structures are dynamically generated during run time.

The meta-level vision planner has knowledge about all the components in the first two layers. It accepts task-level vision commands from the robot system and sends the results of the vision system back to the robot system. Upon a vision task request, the vision planner transforms the task-level command into sequences of concrete vision operations, executes these operations and, in case of failure, finds out alternative sequences of vision operations.

We have realized a prototype of this multisensory vision system which contains the vision planner and a number of implemented modules. In the rest of this section we give a brief description of some important modules in the prototype. The vision planner is described in detail in Section 3.

2.1 Object recognition in intensity images

The intensity images are acquired by a CCD camera. From an intensity image, line and junction features are extracted by means of the following steps [10, 11]: convolution of the intensity image with a Gauss-Laplace operator; edge point detection by locating the zerocrossings in the convoluted image with subpixel accuracy; linking of the edge points to edge chains; segmentation of the edge chains into straight lines, arcs and ellipses; connection of the lines at junctions and classification of the junctions. The information of lines and junctions is represented by means of an attributed graph structure. Similar attributed graph structures for model objects are constructed by the CAD-interface and stored in the model database. The recognition is based on the hypothesis-and-verification principle. A small number of scene features are assumed to correspond to some model features (hypothesis generation). These correspondences allow the unique determination of the spatial transformation that causes the model to be transformed into the image plane. The hypothesis is then verified by using the transformation to project other model features into the image plane and check their existence. Due to the two stage strategy this recognition method is computationally very efficient.

2.2 Object recognition in range images

2.2.1 Range sensing

The range sensor we use is an active system based on the coded light approach [3]. Through a sequence of n binary patterns from a projector, 3-D space can be partitioned into 2^n thin regions. A camera receives n images, each corresponding to one binary pattern, and creates a stack of n bit maps assigning an n bit code to each observed scene point. The bit code and the geometric relationship between camera and projector, obtained through a calibration procedure, is used to calculate the distance of each observed scene point by triangulation. Our sensor acquires a range image of 512×512 points and a registered intensity image in a few seconds. For more details see [17].

We have established a range image database of about 20 scenes containing up to five polyhedral objects.

2.2.2 Range image segmentation through scan line grouping

The goal of range image segmentation is to partition range images into surface patches useful for subsequent interpretation tasks. For the analysis of scenes containing only polyhedral objects it is adequate to segment range images into planar regions. A fast method for doing this [12] is based on the observation that in a scan line, the points belonging to a planar surface form a straight line segment. On the other hand, all points on a straight line segment surely belong to the same planar surface. Based on these observations, we first divide each scan line into straight line segments and subsequently do region growing using the set of line segments as the segmentation primitives. Due to the use of straight line segments instead of the individual pixels, the data dimension that must be handled in the region growing process can be greatly reduced. This makes the segmentation method very fast.

2.2.3 Range image segmentation through variable-order surface fitting

Another region-based segmentation method [16] is an implementation and improvement of the algorithm described in [4, 5]. It segments range images into not only planar but also curved surface patches. In the first step, the range image is divided into subimages by jump edge detection. Each subimage is subsequently divided into smooth patches by pixel grouping based on the sign of mean and Gaussian surface curvature (surface type label). Then, in each region of the same surface type label, a seed region is extracted and expanded by iterative region growing based on variable-order bivariate surface fitting. To improve the region boundaries, a postprocessing step has been introduced in which each region is expanded or contracted according to its local approximation error.

The two segmentation algorithms described in the present section and in Section 2.2.2, respectively, have been tested on range images acquired by three different sensors: (i) images synthesized by a ray tracer developed at our institute, (ii) real range images from a laser range sensor, and (iii) real range images from our own database. The segmentation results for range images from these different types of range sensors were very satisfactory and demonstrate the robustness of the algorithms.

2.2.4 Object recognition by pose clustering

From the surface patches found by the segmentation algorithms, other scene features like vertices and lines can be easily extracted. We have implemented an object recognition method using three-dimensional line segments [7]. The idea is that if we assume a correspondence between a pair of model line segments and a pair of scene line segments, then we can compute the three-dimensional transformation that causes the model to be transformed to the location and orientation (pose) of the object in the scene. By taking all combinations of scene line pairs and model line pairs, a large number of transformations are produced. Among them, the correct transformation, i.e., the one that causes the model to coincide with an object in the scene, will occur much more frequently than incorrect transformations. Thus, the transformations that have a high frequency establish the identification and location of model objects in the scene. Note that since straight line segments are used for matching, this method works only for polyhedral objects.

2.2.5 Object recognition by subgraph isomorphism search

We have developed another object recognition method by subgraph isomorphism search [19]. From the segmentation results we construct a scene graph where a node stands for a surface patch while an edge represents the neighborhood relationship between two surface patches. The graph nodes and edges get assigned attributes, such as area, surface type (planar, curved), angle between two neighboring surface patches, etc. Similar attributed graph structures for model objects are constructed by the CAD-interface and stored in the model database. If part of the scene is an instance of a model object, then the corresponding part in the scene graph must be a subgraph of the model graph and the attributes in both graphs should be compatible. Thus, object recognition can be done by a subgraph isomorphism search. In our actual implementation we have introduced a special clustering method for the surface patches in combination with an extensive use of geometric constraints. This leads to a subgraph isomorphism search of linear time complexity.

2.3 CAD-interface

In our vision system, object recognition is model-based. As described in the previous sections we use a number of different recognition strategies, each of which needs a different model representation. An attractive method for the automatic generation of object models is that of CAD-model transformation. In industrial applications the integration of CAD-models is advantageous since CAD-representations of the objects under study are usually available, due to the application of CAD-systems in the process of object design. On the other hand, geometric representations common in CAD-systems usually cannot be directly applied for object recognition. To overcome this problem, we have built a CAD-interface [9] that automatically constructs the model representations from CAD-models generated by the commercial CAD-system Prime-Medusa.

3 Vision planner

The ultimate goal of our multisensory vision system is to meet the needs of various vision tasks required by an intelligent robot system. The robot system sends task-level vision commands to the vision system. It is then the objective of the vision planner to decide how the vision system should proceed to perform the required vision task. To do this, the vision planner must have knowledge about the modules in the system and the data structures used to store the results produced by the individual modules. Upon a request from the robot system, the vision planner transforms the task-level command into sequences of concrete vision operations by using the knowledge base, executes these operations, and if necessary, finds out alternative vision operations. Finally, the vision planner sends the results of the vision system back to the robot system to support its operations in an unstructured environment.

3.1 Communication with the robot system

One task of the vision planner is to communicate with the robot system. It accepts task-level vision commands from the robot system. Examples of the particular tasks to be performed by the vision system are

- find object X of model database Y in the scene,
- find all objects of model databases $Y_1, Y_2, \cdots, Y_m$ in the scene,
- find the topmost object in the scene.

In the present version of the system, the first two items of this list have been implemented. After planning and executing concrete vision subtasks necessary to perform the overall task-level vision request, the vision planner sends the results of the vision system back to the robot system. The feedback from the vision planner corresponding to the above vision commands is

- the location of object X in the scene,
- the identification and location of all objects found in the scene,
- the identification and location of the topmost object in the scene.

The robot system makes use of this information to guide its operations in an unstructured environment. Later on new vision tasks may arise, say in order to verify the assembly result or to support new robotic tasks. Then the robot system sends again task-level vision commands to the vision planner. The vision planner has to find out appropriate strategies to perform the new vision task and sends the results back to the robot system. Thus the robot system and the vision system work together in a request-and-feedback cycle.

3.2 Knowledge base

The vision planner transforms the task-level vision command from the robot system into sequences of concrete vision operations, executes these operations, and if necessary, finds out alternative strategies. Thus it can be regarded as a meta-level

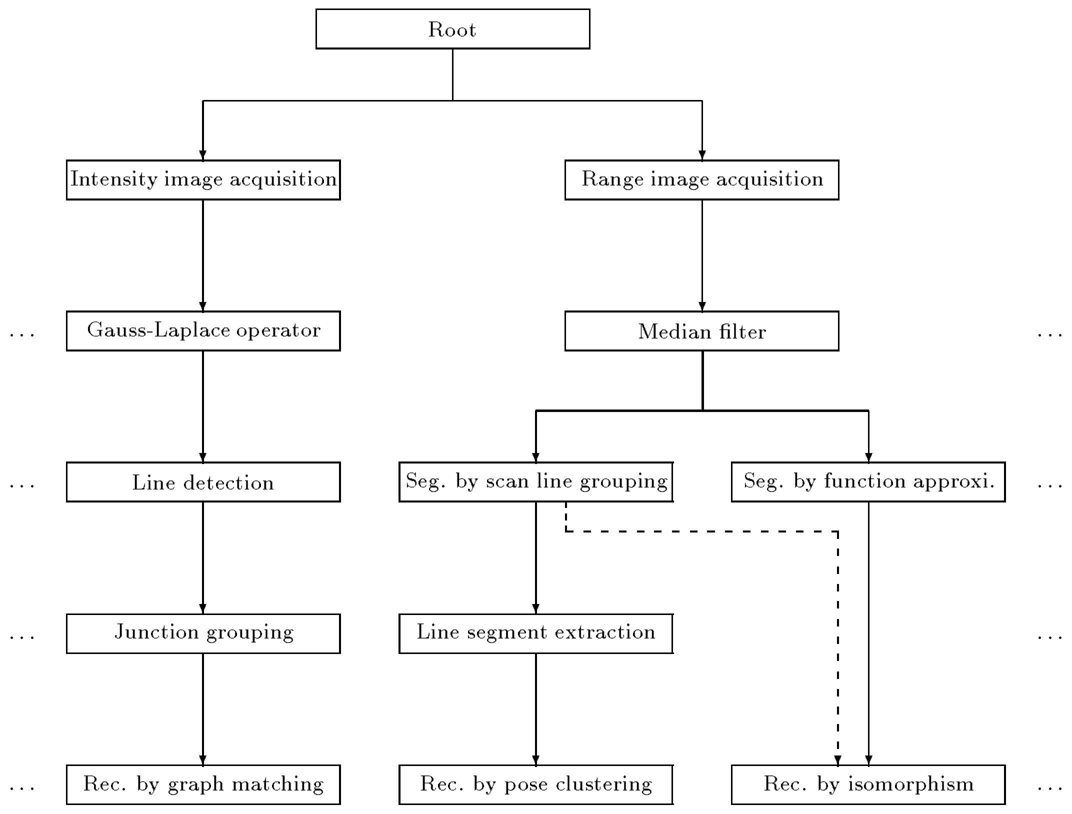


Figure 3: The module network.

component of our multisensory vision system. In order to do planning, the vision planner must have knowledge about all other components in the system. Referring to Figure 2, our vision system contains, besides the vision planner, modules for various vision subtasks and data structures. Thus, the knowledge base of the vision planner is divided into a part *module knowledge* and a part *data structure knowledge*.

3.2.1 Module network

The information on all modules available in the system is stored in a module network. Shown in Figure 3 is part of the module network containing the modules described in Section 2. Each module corresponds to a node in the network and is represented by a frame. Information about a module is stored in the slots of the corresponding frame. In the current version of the vision planner, two types of information are provided:

- runtime: the computation time of the module,
- predecessors: a list of possible predecessor modules.

The computation time of a module may be a constant T, or a function of some system parameters $f(P_1, P_2, \cdots, P_m)$. For example, the computation time of a low-level filtering operation (median filter or Gauss-Laplace operator) depends on the image resolution. Other modules, say region-based image segmentation or object recognition, require a variable computation time, dependent on the scene complexity. In this case we assign a range (T_1, T_2) to the runtime slot and the average $\frac{T_1+T_2}{2}$ is considered later as the actual computation time.

The slot "predecessors" stores the information on the temporal sequence of the modules. For a particular module M, this slot has the form

$$((M_1, C_1),\ (M_2, C_2),\ \cdots,\ (M_n, C_n)),$$

meaning that the modules $M_1, M_2, \cdots$, or M_n can be used to supply the data needed by module M. In certain situations the predecessor relationship between M and some M_i is not always meaningful, but crucially depends on the current vision task. In order to determine the location of an object X in the scene, for example, we may use the module "Rec. by isomorphism" in the module network. If X is a polyhedron, we may use either of the two region-based range image segmentation modules "Seg. by scan line grouping" or "Seg. by function approxi." to get the information necessary for the recognition task. If X is a curved object, however, the module "Seg. by scan line grouping" is no more an adequate predecessor of the recognition module since this segmentation module only partitions the range image into planar surface patches. To represent this kind of task-dependent predecessor relationships we associate each predecessor relationship between M and M_i with a condition C_i. For the example above this condition is formulated as "the object domain is polyhedral". In most cases C_i=true. In Figure 3 the predecessor relationship between M and M_i is shown by an arrow from M_i to M. The predecessor condition C_i is illustrated by a solid arrow for C_i=true, and by a dashed arrow otherwise.

While all frames in the module network have the runtime and predecessor information, some frames possess some additional slots. The most important examples are the recognition modules. They contain also slots with information about

- function: the function of the recognition module,
- domain: the object domain they can handle.

The slot "function" may have the value *find-one, find-all*, or *find-top*, corresponding to the different kinds of vision tasks requested by the robot system. Upon a particular request, this slot enables the planner to find out those recognition modules that are potentially useful to solve the vision task.

Not only the function of a module but also the object domain it can handle determines its usefulness for a particular vision request. If the vision system is required to find the location of a curved object in th scene, for example, then the module "Rec. by pose clustering" should not be considered as a candidate since it can only recognize polyhedral objects. The slot "domain" may have one of the values *polyhedral, curved*, or *free-form.*

The goal of the vision planner is to plan sequences of vision subtasks and to execute them. For the purpose of execution, the vision planner must have knowledge about the calling convention and the input/output data structures of each module in the network. In our system this information is coded in a procedure attached to each module. The execution of a module is done by calling its corresponding procedure. It takes appropriate data structures generated by earlier modules and establishes new data structures for the output of the current module. Then the actual execution of the module is triggered. After the execution the control is given back to the vision planner.

3.2.2 Data structures

Besides the modules in our multisensory vision system the meta-level vision planner has also knowledge about the data structures necessary to store the results from the modules and the vision models from the CAD-interface. As stated in Section 2, all data structures other than the vision models are dynamically generated during run time. The knowledge about these dynamic data structures is also coded in the procedure attached to each module that makes use of data structures generated by earlier modules and produces new data stuctures for later use.

The vision models automatically constructed by the CAD-interface are organized in a number of model databases. Each model database contains

- a list of model names,
- the domain of the models. This may be *polyhedral, curved*, or *free-form*, corresponding to the domains of the recognition modules.

In our vision system object recognition uses the model-based approach. Thus, each recognition module involves some model from the model databases to guide the actual recognition task.

3.3 Planning

Upon a task-level vision request from the robot system, the actual planning in the vision planner is done by a goal-driven best-first search process. The search begins with those modules whose function satisfies the request and recursively generates their predecessors. A solution is found if the special node Root in the module network is reached. During the search the different subpathes in the search space are rated by an evaluation function so that always an optimal solution is found. Then the vision planner executes the modules of the optimal solution. If the execution is successful, the vision planner feeds the final results back to the robot system and starts a new request-and-feedback cycle. Otherwise the vision planner does a replanning by continuing the search to find out alternative strategies.

3.3.1 Search

Given an object recognition request, we can easily determine the function (find-one, find-all, or find-top) and the object domains (polyhedral, curved, or free-form) it concerns. While the function comes immediately from the request, the domains can be derived from the particular object to be localized or the model databases specified in the request. The function and the object domains determine those recognition modules and in turn those sequences of modules that are potentially able to solve the current vision task. These two entities build the input to the goal-driven best-first search.

The search process is described in pseudocode in Figure 4. First, two filters are applied to the set of all object recogntion modules to filter out modules that obviously cannot solve the current vision task. After the function filter has been applied, only those modules remain whose slot function contains the input variable function. Similarly, the domain filter retains only modules that can handle the object domains in the input variable domains. Thus, the modules in the final set are all potential candidates for solving the current vision task.

The actual search is the well-known best-first search. Here we assume an evaluation function f to be available. It rates all the subpathes expanded so far in the OPEN list. In the present version of the planner, this function is the total computation time of a sequence of modules. Note that for node expansion, the predecessors are first checked against the predecessor conditions described in Section 3.2.1. Thus, only those predecessors whose condition is satisfied are really expanded. Since the OPEN list is sorted after each node expansion, the subpath corresponding to the first node in this list is always the optimal one among all subpathes expanded so far. Once the first element of OPEN, N, is the special module Root in the module network, we have reached an optimal sequence of modules that can be immediately executed.

For the purpose of execution we first find out the modules of this new sequence by following the pointers established during the search process. In the replanning phase the current sequence is compared with earlier sequences so that work already done will not be unnecessarily repeated (see next section for more details). The modules contained in the variable new_sequence are then executed. In practice this is done by calling the procedure attached to each module in the sequence that ensures a correct information (data structure) exchange between the modules. If the execution is successful, the search process stops. Otherwise the replanning phase starts to find out alternative strategies.

3.3.2 Replanning

The replanning is simply a continuation of the search process described above in order to find and execute the nth ($n = 2, 3, \cdots$) optimal sequence in turn. The search continues until the execution of some sequence is successful, or the module network has been exhaustively searched and no new sequence can be found (OPEN=Φ).

In contrast with the first optimal sequence, some redundancy may occur during the

```
Input:  function   /* find-one, find-all, find-top */
        domains    /* polyhedral, curved, free-form */

begin

        /* initialization */
   1:   rec_modules := { all object recognition modules }
   2:   rec_modules := function_filter(rec_modules, function)
   3:   rec_modules := domain_filter(rec_modules, domains)

        /* search */
   4:   sequences := $\Phi$
   5:   OPEN := rec_modules
   6:   sort OPEN according to the evaluation function $f$

   7:   if OPEN=$\Phi$ then return (failure)
   8:   N := first_element_of_list(OPEN)
   9:   if N$\neq$Root then
            /* node expansion */
  10:       predecessors := {predecessors of N}
  11:       predecessors := predecessor_filter(predecessors)
  12:       OPEN := OPEN $\cup$ predecessors
  13:       establish a pointer from each new element of OPEN to N
  14:       sort OPEN according to the evaluation function $f$
  15:   else
            /* execution */
  16:       new_sequence := track_solution(N)
  17:       new_sequence := delete_redundancy(new_sequence, sequences)
  18:       execute the modules in new_sequence
  19:       if successful then return (success)
  20:       sequences := sequences $\cup$ {new_sequence}
  21:   endif
  22:   goto 7

end
```

Figure 4: The goal-driven best-first search.

execution of later sequences. To illustrate this point let's assume the first and second optimal sequence being

$$S_1 = \{M_{11}, M_{12}, \cdots, M_{1m}\}, \quad S_2 = \{M_{21}, M_{22}, \cdots, M_{2n}\}$$

respectively. Let's furthermore suppose that the first two modules of S_1 and S_2 are the same, say,

$$M_{11} = M_{21} = \text{Range image acquisition}, \quad M_{12} = M_{22} = \text{median filter}.$$

Then after finding the sequence S_2, if we simply execute it, the range image acquisition and the median filtering would be unnecessarily done again. Similar situations may not only occur between sequences $i-1$ and i, but also between sequences $i-k$ and i, $k > 1$. In order to avoid this kind of redundancy we save all sequences found so far in the variable sequences. Once a new sequence is found, it is compared with all earlier sequences and only that part of the sequence not already done is actually executed. For the example above the execution of the sequence S_2 means just that of the modules $M_{23}, \cdots, M_{2n}$.

4 Implementation

We have implemented a prototype of the vision planner described in the last section on Sun workstations with Common Lisp and Common Lisp Object System (CLOS) [13, 15]. The current version runs in the interative Lisp environment. The communication with the robot system is simulated by a dialog with the user. The syntax of the vision commands are

```
(find-one-object 'database 'object)
(find-all-objects '(database database ...))
(find-top-object '(database database ...))
```

where `database` is one of the two model databases *db-poly* and *db-curved* currently defined in the system and `object` is one of the models in a certain model database.

The best-first search for the planning requires an evaluation function f that rates the candidate sequences of modules so that always the optimal one is found. In our prototype this evaluation function is set to the total computation time of a sequence of modules. Therefore, the solution with the least computation time is tried at first.

The prototype contains the modules described in Section 2. Since they have been developed in their own right and are now available not as subroutines but as stand-alone programms, the integration of these modules into a single system is realized at the operating system level. That is, the vision planner triggers the execution of these programms by sending an execution request to the operating system. The exchange of data structures is accomplished through files.

5 Experimental results

By two examples we demonstrate in this section the ability of the vision planner to find appropriate strategies for different vision tasks. In the first example, we assume that the task of the vision system is to recognize all polyhedral objects in the scene. The corresponding task-level vision command is

```
    (find-all-objects '(*db-poly*))
```

The first solution found by the vision planner is

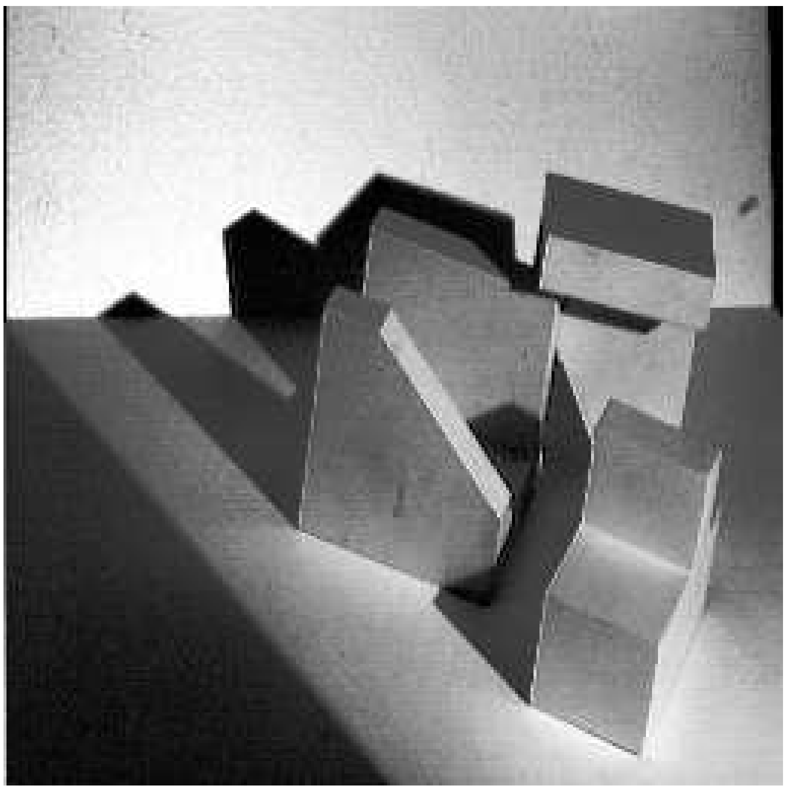
```
    Sequence S1:
        Range image acquisition
        Median filter
        Seg.by scan line grouping
        Rec.by isomorphism
```

Note that for the current vision task, the predecessor condition between the modules "Rec. by isomorphism" and "Seg. by scan line grouping" is satisfied and thus this predecessor relationship is actually used during the planning. Then the vision planner starts to execute this sequence. The range image acquired is shown in Figure 5(b). Also shown in Figure 5(a) is the intensity image of the test scene in order to give the reader a better understanding of the objects present in the scene. Figure 5(c) represents the segmentation result using the module "Seg. by scan line grouping" where the planar surface patches are colored by four grey levels in such a way that no two neighboring surface patches get the same grey level. Note that the segmentation result reveals some oversegmentation of large planar surfaces (the supporting plane and the wall behind the objects) into several smaller surface patches with similar orientation. If necessary, however, these patches can be merged later in the interpretation process and thus cause no serious problems. The module "Rec. by isomorphism" successfully recognizes the three objects in the scene and the pose of the objects is drawn in Figure 5(d). Since the first sequence succeeds, no replanning takes place and the vision planner returns the result back to the robot system.

The second example is to recognize an I-shaped object M_I from the model database $*$db-poly$*$ in the scene shown in Figure 6(a). The object M_I is located in the shadow area behind the large polyhedral object. The vision command

```
    (find-one-object '*db-poly* 'IShape)
```

is given by the user where `IShape` is the actual name of M_I is the model database. The first sequence found by the vision planner equals that in the first example. The range image acquired during the execution of this sequence is shown in Figure 6(b). For this test scene the area around the object M_I is seen by the camera but not illuminated by the projector. Thus, no range data can be computed for this area. Accordingly, after executing the median filtering, segmentation and recognition modules, the object M_I cannot be identified in the scene. This failure causes the vision planner to find the



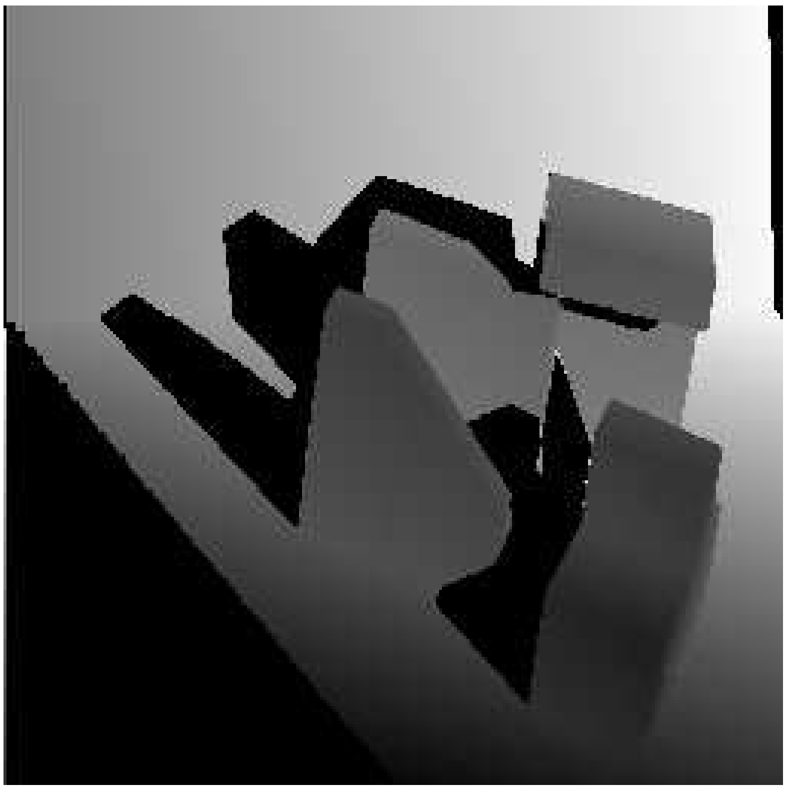
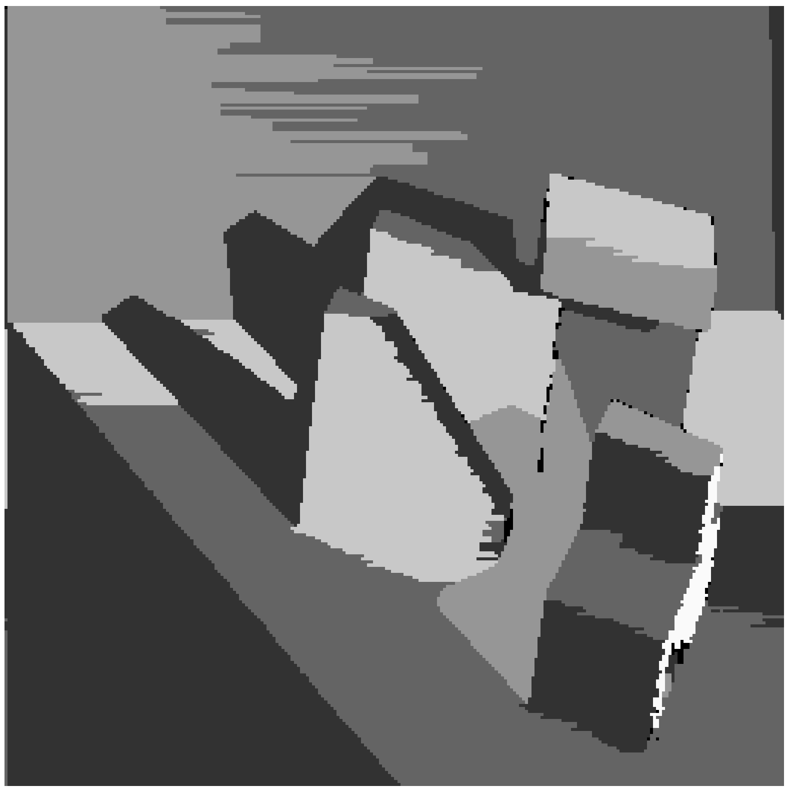


Figure 5: Scene 1. (a) intensity image. (b) range image.



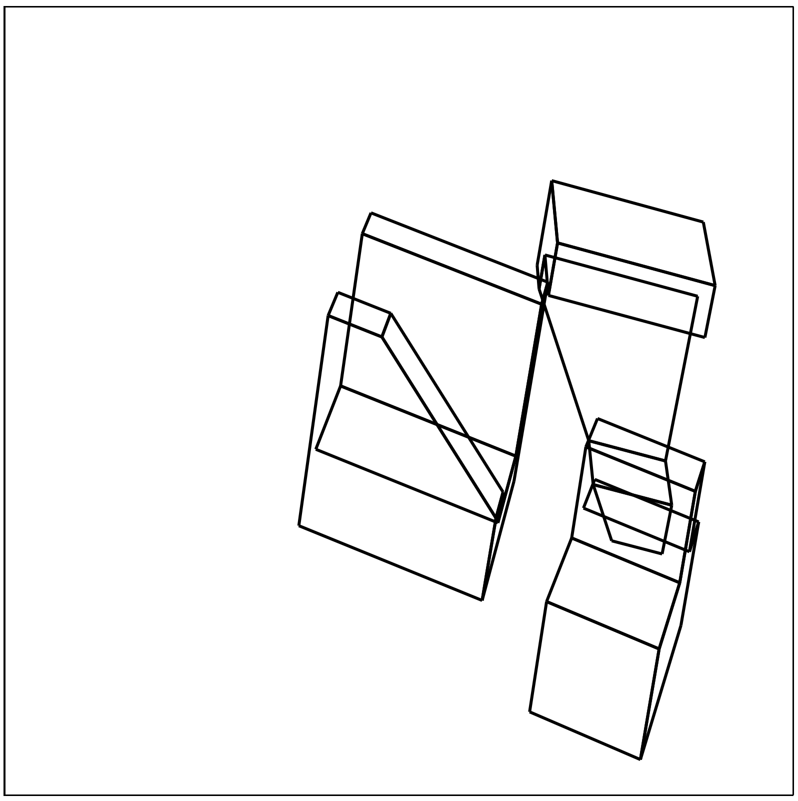


Figure 5: (Cont.) Scene 1. (c) segmentation result. (d) recognition result.

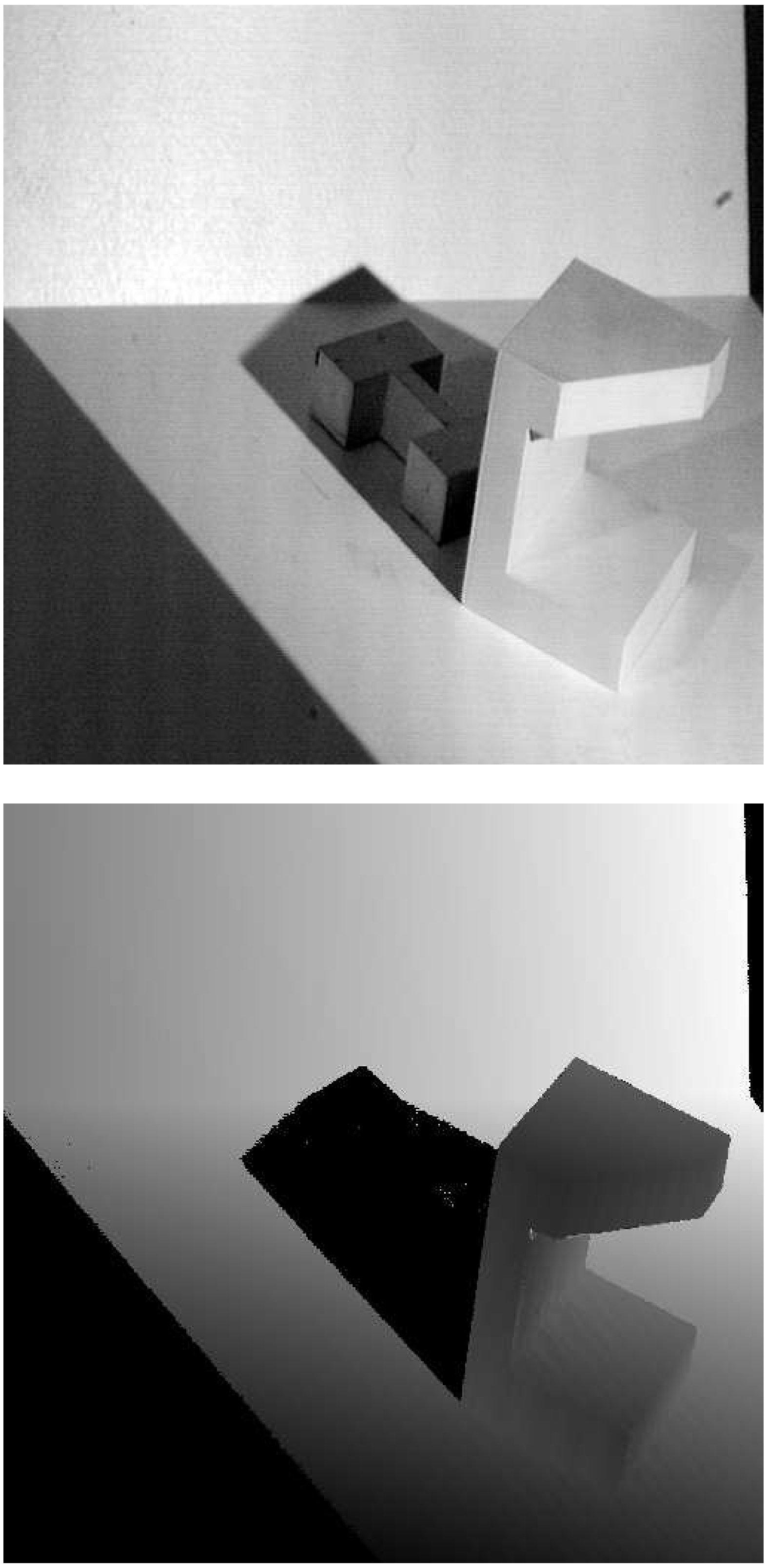
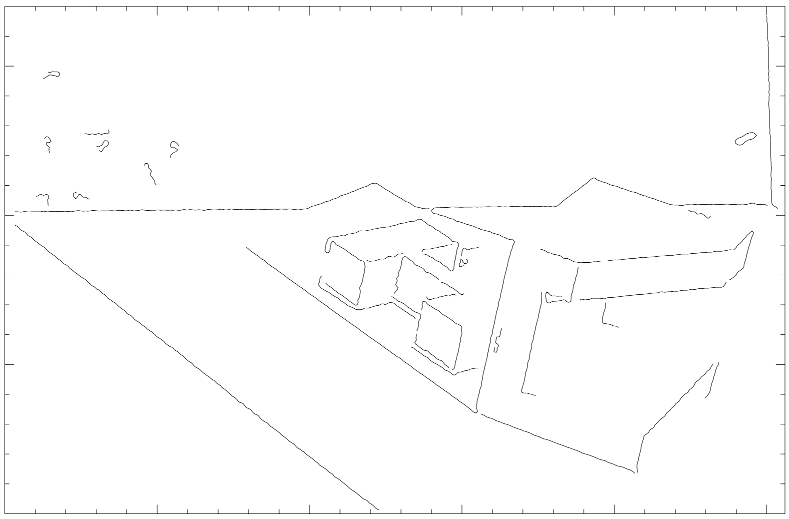


Figure 6: Scene 2. (a) intensity image. (b) range image.



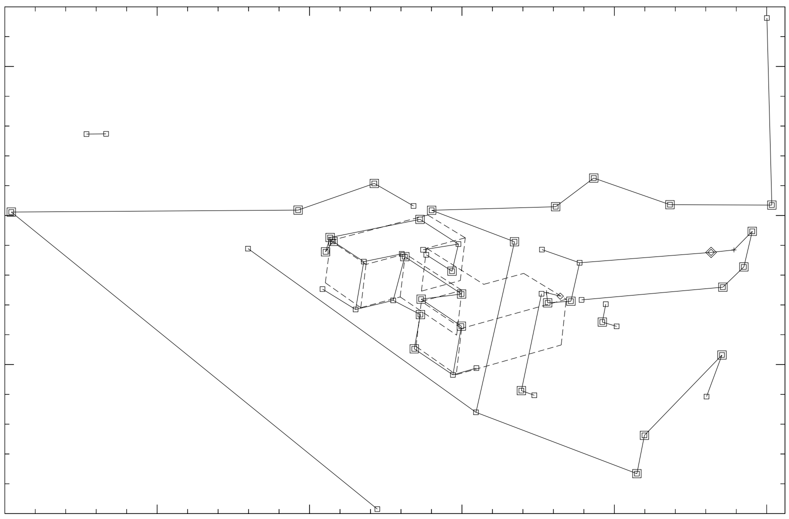


Figure 6: (Cont.) Scene 2. (c) edge points. (d) attributed graph and recognition result.

next candidate sequence

```
Sequence S2:
    Range image acquisition
    Median filter
    Seg.by scan line grouping
    Line segment extraction
    Rec.by pose clustering
```

As stated in Section 3.3.2 we compare a new sequence found in the replanning phase with all earlier sequences to avoid unnecessary reexecution of identical modules. Based on this comparison the vision planner decides to execute only the last two modules "Line segment extraction" and "Rec. by pose clustering" using data structures produced during the execution of the sequence S1. Obviously, the sequence S2 cannot be successful either. The next replanning cycle finds

```
Sequence S3:
    Intensity image acquisition
    Gauss-Laplace operator
    Line detection
    Junction grouping
    Rec.by graph matching
```

The intensity image acquired is shown in Figure 6(a). The edge points in subpixel accuracy are drawn in Figure 6(c), and the attributed graph in terms of lines and junctions in Figure 6(d). The object M_I is successfully recognized and localized. The result is shown in Figure 6(d) by projecting the model back into the scene (the dash lines).

6 Discussions and conclusions

In this paper we have presented a multisensory vision system that is intended to support the vision requirements of an intelligent robot system. Contrary to many other vision systems our system has two significant new features. It contains multiple sensors, object representations and object recognition strategies, and is controlled by a vision planner. So far, a number of modules have been implemented and successfully tested on real intensity and range images. The two examples discussed in last section illustrate the advantage of such a vision planner based multisensory vision system. The availability of multiple sensors, and image analysis and interpretation methods makes our vision system much more flexible than many conventional approaches to meet the need of various vision requests from an intelligent robot system. The vision planner ensures that the appropriate sequence of modules is chosen for a given task-level vision command. As a whole a highly powerful vision system results which is able to support the sensory requirement of robots of the next generation.

To make our multisensory vision system to actually work in an unstructured environment the current prototype must be extended in a number of ways. First of all, more modules should be integrated into the system to extend the functionality of the system. Potential new modules could be the identification and localization of the topmost object in a pile of objects [20], matching of free-form surfaces [6], a.s.o. In a working system the integration of the modules should be realized at a low level. That is, the modules are coded as subroutines and the exchange of information between the modules is done through internal data structures instead of files as in the current prototype.

Given a larger number of modules the maintenance of the system becomes more important. For the purpose of easy maintenance a viewing and editing tool for the knowledge base should be added to the system. That allows the visual representation of the knowledge base, in particular the module network. Instead of adding Lisp codes, the addition of new modules will be greatly simplified by using the tool. Furthermore, the visualization of the planning process may be desireable, too. All these auxiliary tools will make the system more transparent and easier to maintain.

Finally, it is worth to mention that some of the modules integrated in the current prototype have been adopted from the literature while the others are our own new developments. Although these new techniques have been developed in the context of our multisensory vision system, they are valuable in their own right and can also be used in other applications. Some examples of these new developments are the fast range image segmentation method [12] and the object recognition method in intensity images [10, 11] and range images [19]. Also, the CAD-model transformation approach for automatic generation of vision models [9] is becoming more and more important and may make a large contribution to the area of automatic model construction.

References

[1] F. Arman, J. K. Aggarwal, Model-based object recognition in dense-range images - A review, ACM Computing Surveys, Vol. 25, No. 1, 5–43, 1993.

[2] P. J. Besl, R. C. Jain, Three dimensional object recognition, ACM Computing Surveys, Vol. 17, No. 1, 75–145, 1985.

[3] P. J. Besl, Active, optical range imaging sensors, Machine Vision and Applications, Vol. 1, No. 2, 127–152, 1988. Also in J. L. C. Sanz (Ed.), Advances in machine vision, 1–63, Springer-Verlag, 1989.

[4] P. J. Besl, R. C. Jain, Segmentation via variable-order surface fitting, IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 10, No. 2, 167–192, 1988.

[5] P. J. Besl, Surfaces in range image understanding, Springer-Verlag, 1988.

[6] P. J. Besl, The free-form surface matching problem, in H. Freeman (Ed.), Machine vision for three-dimensional scenes, 25–71, Academic Press, 1990.

[7] B. Boyter, J. K. Aggarwal, Recognition of polyhedra from range data, IEEE Expert, Vol. 1, No. 1, 47–59, 1985.

[8] J. P. Brady, N. Nandhakumar, J. K. Aggarwal, Recent progress in object recognition from range data, Image and Vision Computing, Vol. 7, No. 4, 295–307, 1989.

[9] T. Glauser, E. Gmür, X. Y. Jiang, H. Bunke, Deductive generation of vision representations from CAD-models, Proc. of 6th Scandinavian Conference on Image Analysis, 645–651, Oulu, Finland, 1989.

[10] E. Gmür, Robuste und effiziente Erkennung von 3D Objekten mittels Hypergraph-Homomorphismen, in R. E. Grosskopf (Ed.), Mustererkennung 1990, Informatik-Fachberichte 254, 667–674, Springer-Verlag, 1990.

[11] E. Gmür, PHI-2: Ein effizientes und robustes Sichtsystem zur Erkennung dreidimensionaler Werkstücke, Technical Report, IAM-90-016, 1990.

[12] X. Y. Jiang, H. Bunke, Fast segmentation of range images into planar regions by scan line grouping, Machine Vision and Applications. (to appear)

[13] S. E. Keene, Object-oriented programming in Common Lisp, Addison Wesley, 1989.

[14] D. Nitzan, Development of intelligent robots: Achievements and issues, IEEE Journal of Robotics and Automation, Vol. 1, No. 1, 3–13, 1985.

[15] A. Paepcke (Ed.), Object-oriented programming: the CLOS perspective, The MIT Press, 1993.

[16] R. Robmann, Segmentierung von Tiefenbildern, Master's Thesis, 1991.

[17] T. G. Stahs, F. M. Wahl, Fast and robust range data acquisition in a low-cost environment, Proc. of ISPRS-Conference, SPIE Vol. 1395, Zurich, 496–503, 1990.

[18] P. Suetens, P. Fua, A. J. Hanson, Computational strategies for object recognition, ACM Computing Surveys, Vol. 24, No. 1, 5–61, 1992.

[19] A. Ueltschi, Modellbasierte Objekterkennung in Tiefenbildern, Technical Report, IAM-93-018, 1993.

[20] H. S. Yang, A. C. Kak, Determination of the identity, position and orientation of the topmost object in a pile, Computer Vision, Graphics, and Image Processing, Vol. 36, 229–255, 1986.