A Robot Architecture Based on Situated Modules

Hiroshi Ishiguro, Toshiyuki Kanda, Katumi Kimoto and Toru Ishida

Department of Social Informatics, Kyoto University
Sakyo-ku, Kyoto 606-8501, Japan
E-mail: ishiguro/kanda/kimto/ishida@kuis.kyoto-u.ac.jp

The corresponding author

Hiroshi Ishiguro

Department of Social Informatics, Kyoto University
Sakyo-ku, Kyoto 606-8501, Japan

E-mail: ishiguro @kuis.kyoto-u.ac.jp

Fax +81-75-753-4820

The contributions of the paper and its potential applications

This paper proposes a robot architecture that enables us to progressively develop robots (especially, vision-guided mobile robots). For realizing intelligent robots works in an open environment by using various sensors, this kind of robot architectures is strongly needed.

A Robot Architecture Based on Situated Modules

Hiroshi Ishiguro, Toshiyuki Kanda, Katumi Kimoto and Toru Ishida

Department of Social Informatics, Kyoto University
Sakyo-ku, Kyoto 606-8501, Japan
E-mail: ishiguro/kanda/kimto/ishida@kuis.kyoto-u.ac.jp

Abstract

This paper proposes a robot architecture that enables us to progressively develop a robot. The architecture consisting of situated modules has merits of both the traditional function-based and behavior-based architectures in addition to the merit in the development. We have developed a robot based on the architecture. By reporting the development process, this paper discusses advantages of the proposed architecture.

1. Introduction

Intelligent robot research started in SRI [Nilson, 1984] has proposed various robot architectures to date. This paper also discusses a robot architecture. The originality of our architecture comparing with previously proposed ones is to focus on the development process of robot systems. This paper proposes a robot architecture that enables us to progressively develop a robot. One of the typical architectures for intelligent robots is a function-based architecture. The architecture consists of function modules that observe by sensors, represent environments based on sensor information, analyze the representation by using knowledge databases, plan actions and execute planned actions. The function modules are connected in a line and the information processed by each module is sent to the next module. Thus, sensing and action are coupled through various intermediate representations.

This loose coupling causes problems [Brooks, 1991]. For example, a robot often needs to reactively execute actions against to the sensor input. However, it is difficult for the function-based architecture to perform such reactive behaviors. On the other hand, Brooks [Brooks, 1986] has been proposed a behavior-based architecture called *subsumption architecture* that realizes close coupling to the real world. The unique concept of the subsumption architecture consisting of reactive modules is not to utilize any explicit internal representations, but to refer the real world as its own model.

Both of the traditional function-based and behavior-based architectures have merits and demerits. The behavior-based architecture is superior in reactivity to the function-based architecture and the traditional function-based architecture is needed for realizing deliberative behaviors based on environment representations. These architecture should be integrated and several researchers have already proposed the integrated architectures so far [Arkin, 1993; Inaba, 1997; Kuniyoshi, 1997]. However, they still remain an important problem of development methodology. This paper also proposes such a hybrid architecture, however the difference with previous works is to propose a coherent architecture which enable us to progressively develop the robot system. The characteristics of our architecture is as follows:

- The architecture enables us to progressively develop a robot system.
- The architecture enables the robot to adapt to the tasks and environments by controlling the execution order of the situated modules, which are basic components of the architecture.
- In the architecture, representations based on senory information represent relations between situated modules.

The frame work as a programming language is rather similar to PRS (procedural reasoning system) proposed by Georgeff [Georgeff, 1987] and Ishida's control method [Ishida, 1995] for production systems. The idea discussed in this paper basically follows their ideas of procedural control of reactive modules. However, the differences are to deal with problems of robot control and to emphasize the importance of the progressive development of the robot system. Therefore, we limit the control targets to condition-action pairs called *situated modules.* And further the condition-action pairs are not data-driven production rules, they are rather written by a general procedural programming language C.

Based on the architecture, we have developed a robot, which moves in an indoor environment by using visual information. The advantage of the proposed architecture is discussed based on the performance of the developed robot and its development process.

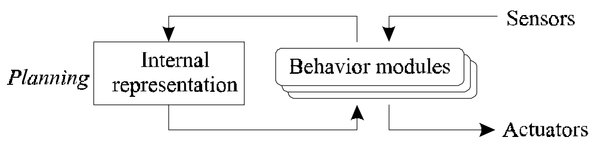

Figure 1: Behavior-based hybrid architecture

2. Robot architectures

2.1. Function-based and behavior-based architectures

The key concept of the behavior-based architecture is that “A robot consists of behavior modules and the feed back control scheme for the behaviors closely couples the robot and the real world each other.” With the traditional function-based architecture, the robot cannot maintain the close coupling with the environment, therefore the robot often loses the relation to the environment and needs to carefully observe again the environment to find the relation. We believe the architecture based on behavior modules is more suitable to robots that behave in dynamic environments.

However, the behavior-based architecture has a demerit that it is difficult to handle deliberative behaviors. In order to compensate this demerit, several approaches integrating both of the function-based and behavior-based architectures have been proposed so far. One method is to integrate the reactive modules and deliberative modules based on a common representation. Arkin and his colleagues [Arkin, 1993] used potential fields to represent both reactive and deliberative behaviors. The behaviors represented with potential fields are easily combined and it can determine robot actions by a vector computation. However, a problem of this approach is that a unique representation to be able to represent any kinds of robot behaviors dose not exist. Another approach is to prepare special modules that can handle environment representations in the behavior-based architecture. This approach is more popular and several researchers are proposing [Inaba, 1997; Kuniyoshi, 1997]. In this paper, we call this hybrid architecture as *behavior-based hybrid architecture* and follow the basic idea to construct a robot system based on behavior modules.

As shown in Figure 1, the hybrid architecture can be realized by adding new modules that can deal with internal representation in stead of the sensors and actuators. Based on this architecture, the robot behaves by accessing to both of the external and internal worlds.

2.2. Development methodology

The behavior-based hybrid architecture has both abilities of the reactive and deliberative behaviors. However, how is the architecture should be evaluated from the viewing point of system development? We consider there is few reports on how to develop robots in previous robotics. Basically, developers are developing robots based on their experience, however practical design methodologies are needed for efficiently developing more sophisticated robots. This is our major motivation of this paper.

Our purpose is not to develop robots that works in limited environments, such as factories, but to develop intelligent robots that can perform various tasks in open and complex environments. In such environments, it is difficult to acquire sufficient information for designing the robots before developing them.

The behavior-based architecture consists of behavior modules and a network connecting them. Especially, it is interesting that the system consisting of only reactive behavior modules generates complex behaviors of the robot [Brooks, 1991]. However, the development of the system is not easy. First of all, it is difficult for developers to decompose complex robot tasks into simple reactive behaviors. Of course, it is possible to find proper reactive behaviors for simple robot tasks, such as moving backward when the tactile sensor is activated and moving along a wall using ultra-sonic sensors. For the simple tasks, such as the obstacle avoidance, the reactive behaviors are closely related to the environment structure. Therefore, the developer can prepare the reactive behaviors without any conflicts. However, if we expect more complex behaviors to the robot, the conflicts easily occur. For example, suppose to add a reactive module for avoiding red objects to the robot moving along walls. In this case, if the wall is red, the robot may iterate to avoid the wall and to go toward the wall; and it will be held up. Further, the design policy of the subsumption architecture is to prepare modules that can be executed in parallel. Unfortunately, it is more difficult to find such reactive modules.

2.3. Internal representation

Another problem of the behavior-based hybrid architecture is its internal representation. Only behavior modules connect the internal world with the external world. Therefore, the internal representation should represent structure among behaviors according to the tasks and environments and it is natural to consider that the network structure of behavior modules is a representation.

In the function-based and behavior-based hybrid architectures, it is not coherent that the modules deal with environment representations given by developers. The utilization of the geometrical maps given by developers’ intuitions prevents the robot from autonomously obtaining and updating the representation. The representation should be obtained through the execution of the behavior modules by the robot itself.

Let us consider the behavior-based architecture again. In the behavior-based architecture, the behavior modules can be prepared in the case where all modules have close relations to the unique representation that the developer can easily understand as discussed already. In other words, the developer needs to carefully consider how the robot tasks are related to the environment and then represents relations between the robot and environment in the network of behavior modules. We consider this indirect representation method is tough for developers and more explicit methods are needed (In contrast, Mataric [Mataric, 1990] has proposed a method to represent the environment by using records of activated behaviors under an assumption that the robot has proper behaviors).

2.4. Requirements for the architecture

We consider an architecture that satisfies the following requirements is needed.

1. The developer progressively implements behavior modules and the system sequentially executes them.
2. The system consists of behavior modules, each of which deals with a particular task in a limited local environment; and the module is programmed based on intuitions and experiences of the developer.
3. The system automatically adjusts the execution order of the modules according to situations of the robot in order to compensate the incomplete module implementation by the developer.
4. The system represents relations between the modules and robot tasks on the module network.
5. The developer can add, remove and update the modules while the robot is working.

It is basically difficult to find modules that can be executed in parallel, and a robot usually executes a single task at a time since the modules cannot share the single robot body. It is natural and easy for developers to consider that the modules are sequentially executed.

The second requirement also guaranties easy programming for developers. However, this causes problems. The modules may not cover all situations to which the robot encounters. They may be redundantly implemented and perform similar functions to others.

The third requirement solves the problem of the incomplete module implementation. By controlling the execution order, the robot automatically adapts to the tasks and environments. For allowing the automatic adaptation, the developer needs to prepare a sufficient number of modules.

Complex tasks or environments may need many modules. In such a case, the system needs to memorize relations between modules for efficiently maintaining them. The environment representations in the fourth requirement maintain the relations.

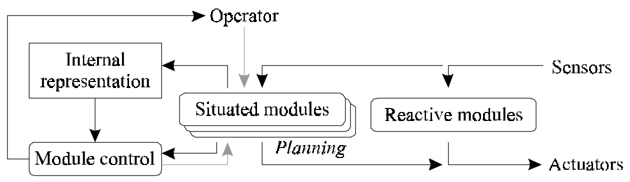

Figure 2: Architecture based on situated modules

Although we do not deal with in this paper, the last requirement is important. When the robot has many modules, the verification in a real environment takes a long time. In such a case, this requirement enables us to develop in an online manner. In a real and open environment, we cannot predict all situations *a priori*, however we should not stop the robot performing a task. The function of online development is strongly required for developing systems working in a real world.

2.5. A new architecture

Taking the requirements into account, we propose a new architecture as shown in Figure 2. The difference with the architecture shown in Figure 1 is as follows.

We categorize modules into the *reactive modules* and *situated modules*. Obviously, a robot needs primitive reactive modules to avoid dangerous situations, such as a module to go backward when colliding with an obstacle. A robot needs to execute the primitive reactive modules with the first priority. However, more sophisticated or high-level *situated modules* are programmed according to tasks and environments. This is also a natural understanding of human behaviors. We, human, have a low-level reactive behavior putting back one's hands by reflex when touching a heated object. However, for other higher-level behaviors, we sequentially execute them while continuously changing its attention [Ballard, 1991].

The *situated modules* are under control of the *module control*. The *module control* controls the execution order of the *situated modules*, find executable modules and evaluate modules for providing information to the developer.

The major difference with the previous architectures is in the role of the internal representation and the task planning, in addition to the module control. In the behavior-based architecture, especially the subsumption architecture that executes reactive modules in parallel, it is difficult to perform task planning. On the other hand, our architecture sequentially executing situated modules performs task planning by controlling the execution order of the modules. Almost all of the previous approaches perform the planning on environmental representations, however it is often difficult to prepare proper

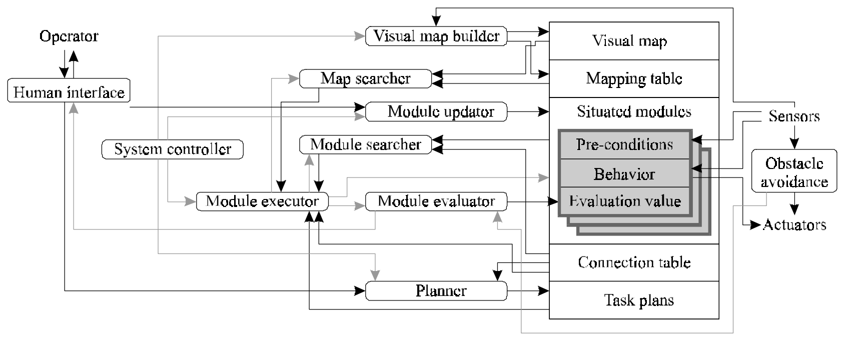

Figure 3: System configuration

representations that maintain the coherent relations between the robot actions and sensor information and multiple representations, whose utilization is rather complicated, may be needed. The planning on the network of the situated modules is much simpler and it can represent any kinds of robot tasks.

Finally, the internal representations memorizing the relations between situated modules are not used for the planning but for recovering relations between the robot and the environments. The robot refers the internal representations on a purpose to find executable situated modules.

3. Architecture based on situated modules

3.1. System configuration

We have implemented a robot system based on the new architecture as shown in Figure 3. The system consists of the *situated modules* and several components for maintaining the *situated modules.*

The developer accesses to the *module updator* and adds new *situated modules* through the *human interface*. And further, the developer accesses to the *planner* and gives plans to the robot. The *module executor* sequentially executes the *situated modules* by referring to the *task plans* and the *connection table*. In the case where the tactile sensor is activated while executing the situated modules, the *obstacle avoidance* is activated and the robot reactively avoids obstacles. The *module evaluator* evaluates the situated modules according to the execution of the *obstacle avoidance module*. When the *module executor* loses executable modules, the *module searcher* searches executable modules in the network of the situated modules. If they are not found, the *module executor* accesses to the *map searcher*, and the *map searcher* searches executable modules in the *visual map*. The following sub-sections explain several features of this system.

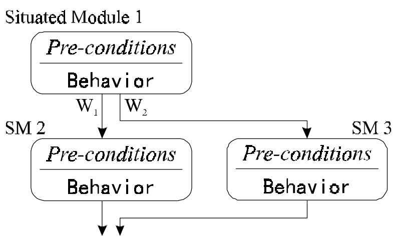

Figure 4: Situated modules

3.2. Situated module

For easy development of the modules, we define a module as:

A program which performs a particular robot behavior in a particular local environment.

It is tough for developers to take consistency between all modules into account. However, it is possible for developers to consider which sensor can be used and how the robot can move in the local environment in order to achieve the subtask. This situated module consists of pre-conditions and actions as shown in Figure 4. The developer progressively develops modules in order to achieve the pre-determined robot tasks. For example, the modules shown in Figure 5 can achieve a task that the robot gets out of the room and goes toward the fire hydrant. Of course, the combination of modules is not unique. The incompleteness of the modules is compensated by adding new modules, evaluating the modules and deleting the unnecessary modules.

3.3. Addition, evaluation and deletion of modules

Basically, the developer prepares a redundant number of

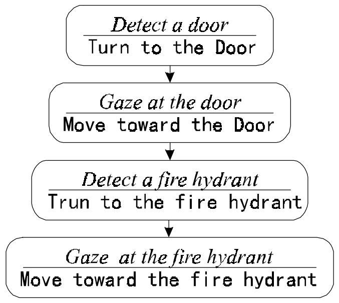

Figure 5: An example of a situated module sequence

modules in order to cover various situations as follows. First of all, the developer implements several modules needed for basic robot behaviors, then add modules for achieving a task. When the robot cannot perform the task, the developer knows that the module currently executed has a problem and adds new modules that compensate the problem.

Here, we do not delete the module implemented before as shown in Figure 4. The new module M3 is connected in parallel to the previously implemented module M2 and these are selected by the *module control* based on the evaluation values $W_1$and W_2.

3.4. A function as a system that never halt

As discussed already, one of the problems of this development method is that the robot often encounters a situation where it loses executable modules if it does not have a sufficient number of modules. In order to avoid this problem, the system has two functions of error recovery. One is the *module searcher*. The *module searcher* navigates the robot randomly and tries to find a situation where one of the modules can be executed. The *obstacle avoidance* takes the same role. When the robot collides with an obstacle, the *obstacle avoidance* navigates the robot in a different direction to the obstacle. These functions of error recovery guaranty the robot not to completely halt.

3.5. Evaluation of situated modules

As the developer increases the number of situated modules, the system often has multiple executable modules. The system needs to select the best module for the situation. For the module selection, the *module evaluator* evaluates modules according to results of task execution. In the following cases, the system finds that the task has failed.

(a) There is no executable module.

(b) The *obstacle avoidance* is activated.

In these cases, the system updates evaluation value $M_i, i = 0,...,N$ assigned to N situated modules executed so far by the following equation:

$$w_i = \gamma\, w_{i-1}, M_i = M_i - w_i$$

That is, the evaluation values are reduced w_i with a damping coefficient γ. On the other hand, when the task is accomplished, the evaluation values of related modules are increased a constant value.

The *module control* refers to the evaluation value and selects one of the executable situated modules. Further, the developer also refers to the evaluation values in order to update situated modules. Modules that have a small evaluation value will be deleted. Thus, the module evaluation compensates the incomplete implementation of the situated modules by the developer.

3.6. Task planning

Besides the *evaluator*, the *planner* controls the execution order of situated modules. In this architecture, a task plan is represented as a sequence of situated modules. The *planner* plans a path from the current module to the destination module on the module network.

Here, each module has different evaluation values for different robot tasks. Suppose there are three points A, B and C; and the robot can move among all points except a path between B and C. In this case, if the robot updates the evaluation values without distinguishing the tasks, it enforces the evaluation value of the modules navigating from A to B and selects an impossible path A-B-C to move from A to C. In order to avoid this problem, the evaluation value should be assigned for each task.

As increasing the number of tasks, the number of evaluation value increases. Therefore, the evaluation values for each task should be integrated after sufficient evaluation. This is one of our feature works.

3.7. Visual map as an internal representation

As discussed in the previous section, the network structure itself is the internal representation of the robot in our architecture. However, it is difficult for the progressive implement of the modules to represent the complex structure; it is rather one-dimensional representation. The system needs to build more complex structure that reflects complex relations between the robot and environment.

In our current implementation, the system analyzes the structure based on sensor information. Especially, the robot performs navigation tasks. Therefore, the omnidirectional vision sensor attached to the robot provides sufficient visual information to find relations between modules. The visual map in the architecture is the representation that represents relations between

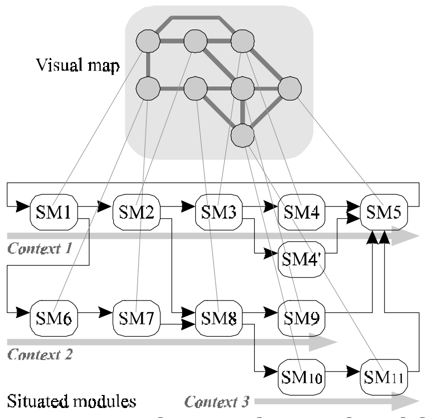

Figure 6: Visual map and situated modules

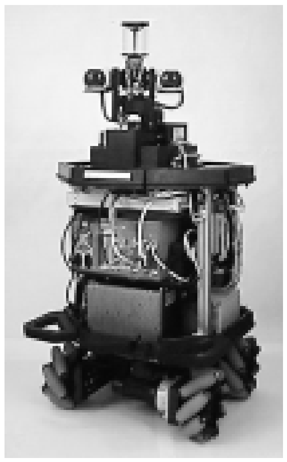
Figure 7: Developed robot

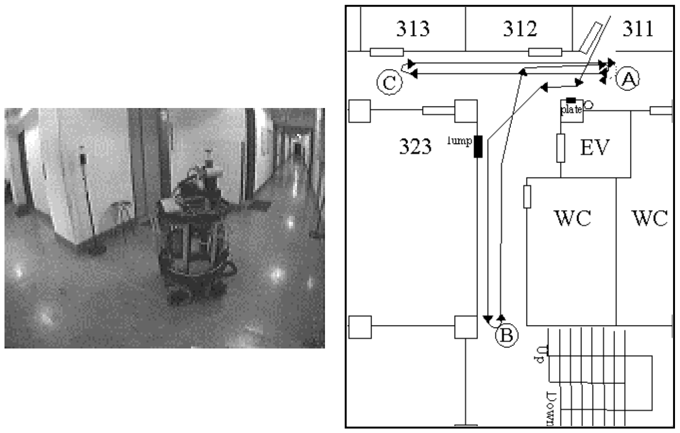

Figure 8: Indoor environment

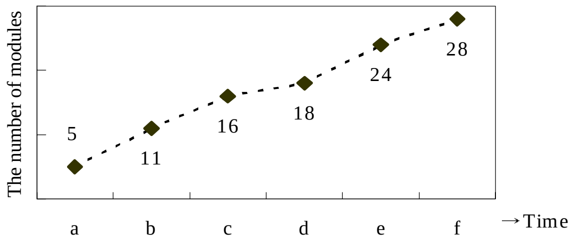

Figure 9: The number of implemented modules

situated modules (see Figure 6). Utilization of more general representations is remained as one of our feature works.

The *visual map* is updated when one of the situated modules is executed. The *visual map builder* acquires omnidirectional images and memorizes them with pointers to the situated modules. If similar omnidirectional images exist, the situated modules are regarded as they have close relations. For estimating similarity among omnidirectional images, Fourier transform is applied. Fourier transform decomposes the omnidirectional images into two components: phase components representing the robot orientation against the environment and magnitude components representing visual uniqueness of the location. The spatial relations are found by comparing the magnitude components [Ishiguro, 1996].

4. A robot and its development process

4.1. Hardware configuration

We have developed a robot [Ishiguro, 1997]. Characteristics of the hardware supporting the architecture are as follows:

(a) The robot has various sensors and a suitable body to the environment where people exist.
(b) The robot has sufficient computing resources for processing the sensory information.
(c) The robot can work sufficiently long.
(d) The operator can update the robot system though a wireless communication link.

The robot has mainly four types of external sensors: a stereo vision system of which camera parameters, such as zooming and gazing, can be controlled by a computer, an omnidirectional vision sensor [Ishiguro, 1998], four ultra-sonic sensors and sixteen tactile sensors. The size of the robot 60 cm in diameter; and it has a special mechanism to stabilize the pose against the rough ground surface and to direct itself in arbitrary directions. The robot can work while 4 hours and communicate with the operator through a wireless Ethernet connection with a bandwidth of 2 Mbps. As an on-board computing resource, the robot has a PC (Pentium 150MHz). The operating system is VxWorks. Figure 7 shows the developed robot.

4.2. Development process

Based on the architecture, we have implemented a system that navigates the robot in the indoor environment shown in Figure 8. The task of the robot is to move among point A, B and C shown in the figure. Figure 9 shows increase of the number of implemented situated modules and Figures 10, 11, 12 show implemented situated modules at times b, d and f, respectively. In the figures, the squares represent situated modules and the arrows show the execution order. The development process mainly consists of the following three steps.

[Step 1] Progressive implementation of situated modules for a single task

In the beginning, we have implemented modules for a single task to navigating the robot from the computer room (331 in Figure 8) to point B through point A. For this task, we have implemented 11 situated modules

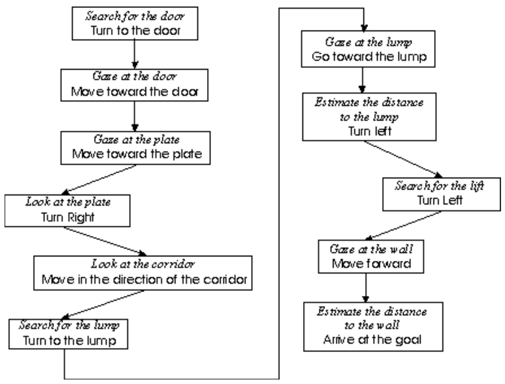

Figure 10: Implemented modules in Step 1

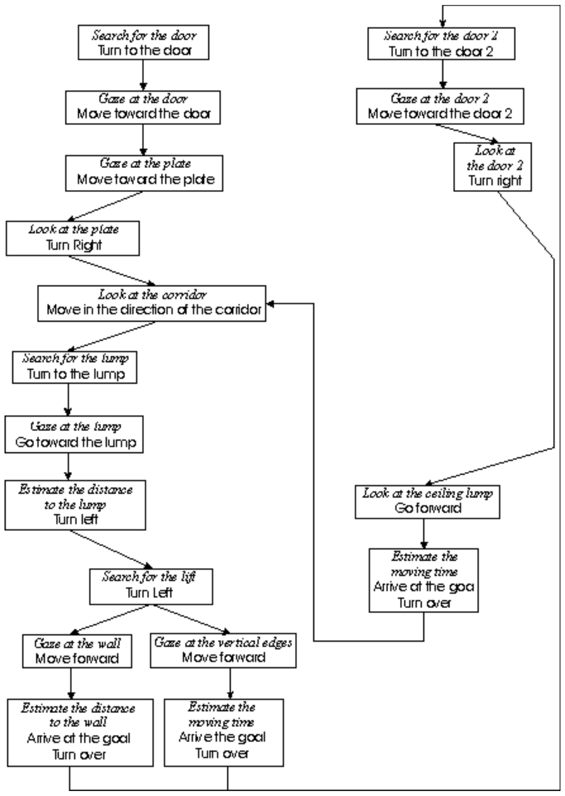

Figure 11: Implemented modules in Step 2

shown in Figure 10. At this moment, the situated modules were arranged in a line and the *module control* executed each module in order.

[Step 2] Module evaluation with multiple simple tasks

The second step in the development is to implement simple robot tasks with which the robot can keep working. We have implemented two tasks: moving from A to B and moving from B to A. The number of modules was 16. While iterating the tasks, the robot evaluated each module. Based on the evaluation, we have implemented another two modules that compensate drawbacks of pre-

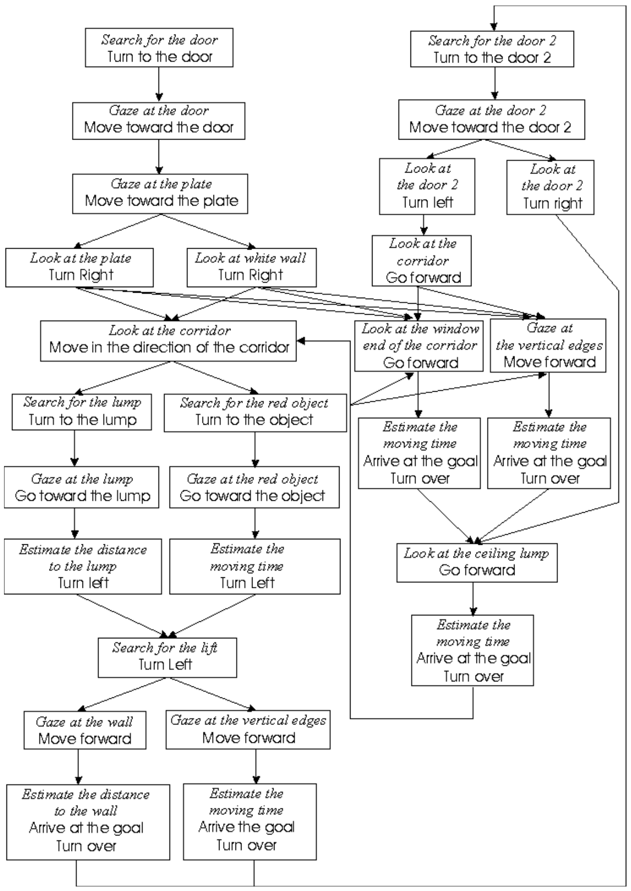

Figure 12: Implemented modules in Step 3

implemented modules. The robot has selected modules according to changes of lighting conditions. Figure 11 shows 18 modules implemented by this time.

[Step 3] Task planning and utilization of the visual map

In this step, we have implemented more complex robot tasks. Six tasks to navigate the robot between A, B and C have been implemented and the robot performed planning based on the evaluation values. When we have implemented 24 modules, the robot tended to be short of stability. The robot often lost executable modules, moved randomly and referred to the *visual map*.

Although we could verify the effectiveness of the error recovery functions, we have implemented another four modules in order to stabilize the robot behaviors. Figure 12 shows all modules finally implemented in our experimentation.

4.3. Performance of the robot

The whole task of the robot is to randomly select a destination point among A, B and C; and move toward it. Figure 8 shows an example of the robot path. With 28

Starting point	Destination	Mean time	Shortest time
A	B	55 sec.	45 sec.
A	C	73	55
B	A	62	45
B	C	137	64
C	A	90	55
C	B	150	70

Table 1: The mean time to move among A, B and C

situated modules finally implemented, we have estimated performance of the robot.

The robot arrived at the destinations 33 times while one hour. The robot executed the *module searcher* and *map searcher* in order to find executable modules while 7 min., and it executed situated modules while 53 min. Generally, it is difficult to navigate a robot with vision sensors in a dark and less-textured environment as shown in Figure 8. However, we could develop the robot working in the environment by using vision sensors. We consider this shows robustness of our architecture.

Table 1 shows the mean time for moving among A, B and C and the shortest time. The shortest time is a time that the robot takes when it moves along the shortest path with a constant normal speed. Of course, to get the shortest time does not mean that the robot has the best modules. However, it is still one of the performance measures. As shown in Table 1, the mean time is close to the shortest time except the paths between B and C. As the paths between B and C, the robot selected the paths B-A-C and C-B-A by referring to the evaluation value, since there is no stable visual feature between B and C.

5. Conclusion

Although the implemented robot behaviors are simple as a robot that works in a real environment, the experimental results has convinced us of the possibility of the proposed architecture. We believe it is possible to develop robot systems in a progressive manner based on the proposed architecture. Our next step is to implement situated modules for communicating with people and working in an outdoor environment.

We need more deep considerations, but the proposed architecture is general and covers several important concepts previously proposed. For example, the *context* [Chatila, 1991] can be represented as a sequence of situated modules as shown in Figure 6: Visual map and situated modules

.

References

[Arkin, 1993] R.C. Arkin, et al., Active avoidance: escape and dodging behaviors for reactive control. Active Robot Vision, World Scientific Publishing, pp.175-192, 1993.

[Ballard, 1992] D.H. Ballard and C.M. Brown, Principles of animate vision, CVGIP: Image Understanding, Vol. 56, No. 1, pp. 3-21, 1992.

[Brooks, 1986] R.A. Brooks, A robust layered control system for a Mobile Robot, IEEE J. of Robotics and Automation, 1986.

[Brooks, 1991] R.A. Brooks, Intelligence without representation, Int. J. Artificial Intelligence, Vol. 47, pp. 139-159, 1991.

[Chatila, 1991] R. Chatila, et. Al., From planning to execution monitoring control for indoor mobile robots, Proc. ISER, pp. 207-221, 1991.

[Georgeff, 1987] M.P. Georgeff, et al. Rreasoning and planning in dynamic domains: An experiment with a mobile Robot. SRI International Technical Note 380, 1987.

[Inaba, 1997] T. Oka and M. Inaba and H. Inoue, Describing a modular motion system based on a real time process network model, Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems, pp.821-827, 1997.

[Ishida, 1995] T. Ishida, et al. A meta-level control architecture for production systems. IEEE Trans. on Knowledge and Data Engineering, Vol. 7, No. 1, pp. 44-52, 1995.

[Ishiguro, 1996] H. Ishiguro and S. Tsuji, Image-based memory of environment, Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems, pp. 634-639, 1996.

[Ishiguro, 1997] H. Ishiguro and K. Kimoto, Town robot - Toward social interaction technologies of robot systems -, Proc. Int. Conf. Field and Service Robotics, pp. 115-120, 1997.

[Ishiguro, 1998] H. Ishiguro, Development of low-cost compact omnidirectional vision sensors and their applications, Proc. Int. Conf. Information systems, analysis and synthesis, pp. 433-439, 1998.

[Kuniyoshi, 1997] Y. Kuniyoshi, Fusing autonomy and sociability in robots, Proc. Int. Conf. Autonomous Agents, pp.470-471, 1997.

[Mataric, 1990] M.J. Mataric, Environment learning using a distributed representation, Proc. IEEE Int. Conf. Robotics and Automation, pp. 402-406 1990.

[Nilson, 1984] N.J. Nilson, et al., Shakey the robot, SRI International Technical Note 323, 1984.