Robotica (1998) volume 16, pp. 387–399. Printed in the United Kingdom © 1998 Cambridge University Press

Integrated control architecture based on behavior and plan for mobile robot navigation

Jiyoon Chung, Byeong-Soon Ryu and Hyun S. Yang

Center for Artificial Intelligence Research, Dept. of Computer Science, KAIST, 373-1 Kusong-dong, Yusong-ku, Taejon, 305-701 (KOREA). E-mail: hsyang@cs.kaist.ac.kr

(Received in Final Form: September 16, 1997)

SUMMARY

One of the most difficult challenges in mobile robotics is real-world navigation. A real world can change suddenly and this change makes the robot relinquish planning actions in advance. In order to overcome such a change, behavior-based navigation was introduced. However, it had a difficulty in planning deliberate actions and in communicating with humans.

We propose a new control strategy combining both the merits of behavior-based and planner-based approaches. The architecture consists of three major parts: *Behaviors*, *Planner*, and *Coordinator*. The *Planner* plays two important roles: 1) as a flexible human interface and 2) as the planner itself. The *Coordinator* serves as an interface between *Behaviors* and *Planner* and guides *Behaviors* to accomplish meaningful tasks according to the guidelines from the *Planner* and the *Position estimator.*

We also provide a brief description of the intelligent mobile robot CAIR-2 and for '95 IJCAI/AAAI Robot Competition and Exhibition when the robot was placed first in the Office Delivery event.

KEYWORDS: Mobile Robot; Navigation; Behaviors, Planner; Coordinator.

1. INTRODUCTION

Mobile robots are rapidly increasing the application areas for robotics in today's world compared with a couple of decades ago. One of the most difficult challenges in mobile robotics is autonomous navigation in a real world. A real world can change suddenly, so a robot in this environment easily makes wrong estimates of which events can happen and which events can produce unwanted side effects. This forces the robots to relinquish planning actions in advance. Traditional approaches (e.g., ordered lists of actions, triangle tables, precedence diagrams, and *and/or* graph) which were used to represent task plans[1] before operation. Even if they had more flexibility and intelligence, they could not be applied in such an environment without the aid of a program-like exception handler.

In order to overcome such uncertain and drastical changes, *behavior-based navigation* was introduced. In this paper, the *behavior* denotes a computational module that rules the robot's "behavior" that originally denotes "the response of an individual, group, or species to its environment". Though the term is also used for the original meaning somewhere in this paper without notification, we think such a confusion may not be serious since the computational module and original "behavior" are closely related. In most behavior-based navigation systems, each behavior has its own goal and produces action commands which are incorporated with the action commands generated by other behaviors to produce an actual action command that is used by the actuator. Since most behaviors were designed to achieve only one simple and specific task and to have minimal memory, behavior-based navigation could produce a fast response. Hence this approach seemed to be a better way to solve real world navigation problems. Brooks introduced the subsumption architecture as one example of behavior-based controls which showed successful experimental results.[2] This work has been a center of attention because it was quite fast and seemed more intelligent than the previous work because they processed whole functions serially in a single control loop. Hence they seemed dull regardless of their potential ability, but behavior-based navigation had multiple control loops and sensory requests which were usually written as a set of concurrent processes. They thus seemed to have much more flexibility. However, they had some drawbacks due to the multiple control loops. Much effort was required to design an arbitration mechanism to resolve control output conflicts resulting from attempts to control the same actuator simultaneously. Moreover, these behaviors had to cooperate to accomplish a meaningful result, but it was difficult to coordinate them in the most fruitful way. The problem is that as complexity increases, interactions between behaviors increase as well, the point where it becomes difficult to predict the overall behavior of a system.[3]

We had learned some lessons from the experiments in developing an outdoor navigation for the '93 Taejon World Exposition. Two of them indicated that action plans are necessary for robots to achieve more sophisticated tasks and a central coordinating mechanism is also necessary for humans to give commands and to watch the robots' states. We thus introduced the notions of explicit plan representation and a central coordination mechanism into a behavior-based robot. We have recognized that these notions interfered with a certain reactivity, but they could be desirable if a robot navigated in a partially known environment. Fortunately, few experiments of navigation had been done in absolutely unknown environments such as the planetary surfaces and the bottom of the sea. Most experiments had been done in partially known environments with many movable independent objects such as humans, robots, and vehicles. In such environments, a robot had to have a communication system with the objects. For example, when a robot was in a trap, it could escape from the trap with a human's assistance through a communication channel like voice. We tried to design a communication system similar to that of humans; otherwise we would have to learn how to command and how to receive the assistance requests. Hence we decided to apply a *topological map* for the representation of an environment.

For these reasons, we designed a new control strategy combining both the merits of behavior-based and planner-based approaches. More emphasis on the planner's role may weaken the reactivity, so we made behaviors play the major role of navigator while the planner just recommended which direction should be best. The architecture has three major portions: *Behaviors*, *Planner*, and *Coordinator*. The *Planner* plays two important roles as a flexible human interface and the planner itself. It can interpret a sentence like "take a right at the signal light" and build a topological map corresponding to the sentence. Such a sentence can be communicated to the planner through multimodal human interfaces like voice and gesture. The *Coordinator* served as an interface between *Behaviors* and *Planner* and guided *Behaviors* to accomplish meaningful tasks according to guidelines from the *Planner* and the *Position estimator* which estimated a robot's posture in the topological map. In order to manage *Behaviors* the Coordinator made a *wake-up list* which contained those behaviors which should be activated. Since *Behaviors* are independent processes in the view of an operating system, they are scheduled by an operating system and only activated behaviors are able to produce their control outputs. These control outputs can try to control the same actuator simultaneously because they are made without consideration for the other behaviors. In order to resolve this conflict we designed a *two-stage blender* which considered how to surmount the drawbacks of *integration* and *winner-take-all* strategies.

In this paper, we will present the Intelligent Mobile Robot CAIR-2. Especially we put great emphasis on an integrated mobile robot control architecture based on behavior and planner with flexible human interfaces. We also present the ability of CAIR-2 at the '95 IJCAI/AAAI Robot Competition and Exhibition[4] when the robot was placed first in the Office Delivery event. CAIR-2 lasted three months outdoors for real world demonstration during the '93 Taejon World Exposition.

1.1. Related work

Brooks introduced the subsumption architecture,[2] which was layered and had multiple independent task-achieving modules. Since it is quite fast and robust in comparison with previous work, it has been the center of attention. Behavior-based control is quite different in splitting the problem of autonomous navigation because it is split by task rather than by function. Traditional functional splitting attempted to construct general-purpose functional modules, such as a modeler and a planner, and placed them in a single control loop. For example, the *Stanford Cart* and the *CMU Rover* used the navigation system developed by Moravec which is based on a two-dimensional grid map. Meaningful features are extracted from multiple sensors, and then they are plotted on this map. A best shortest path is then chosen for movement. The cart would advance by 3 feet and then repeat the above processing. The *SHAKEY*, SRI International 1966–1972, *Hilare*, LAAS laboratory in Toulouse 1977–, and *NAVLAB*, CMU 1983–, can be placed in this category. Some of them tried to build precise models of the world and the navigation problem became the automatic generation of abstract plans which were generated and executed in simulated worlds where all the necessary information was available, complete, and correct. On the other hand, behavior-based control advocates the construction of behaviors that are independent processes and operate in parallel. Individual behaviors are similar to traditional approaches except they are simple and use minimal memory.

Many behavior-based robots were implemented which have the following features in common and, by which we can roughly define the characteristics of the behaviors. The first is that they usually are simple and fast; the second is that they seem to have reactivity. These are both significant advantages over traditional ones. The former is natural because behaviors operate in parallel rather than in series; therefore fast behaviors need not be delayed by slower ones. Also these behaviors are task-specific rather than general-purpose; therefore, the behaviors' designers can take advantage of the structure of the tasks in order to simplify the behaviors. However, the latter has been the subject of a controversy, since there are two commonly accepted meanings for reactivity. The first definition is that a system response to the input is sufficiently short. The second is that a system minimizes the use of the internal state. The first definition is self-evidently a desirable property for a robot control system, but there is a growing consensus that the second definition may not be the best way to achieve it under all circumstances.[5] We also allowed the use of memory and many behaviors were implemented with storages.[6–8] In addition, a few behaviors were implemented by using traditional *AI* techniques and although they could not be reactive, they were able to handle high level decisions. It has been claimed that traditional *AI* techniques cannot be used to produce reactive behavior,[9] but recent researchers have challenged this claim.[3,7,10]

Since the late 1980s, some work has focussed on how a higher-level deliberation process extends reactive systems and there is growing consensus that either the planner-based or the behavior-based system alone is insufficient. Also, the line between the planner-based approach and the behavior-based one becomes blurred.[3,7,8,11,12]

D. Payton, J. Rosenblatt, and D. Keirsey presented a set of architectural concepts that addressed the needs for integrating high-level planning activities with low-level reactive behaviors[8] by using a specialized behavior in order to enable the vehicle to execute long-range traversal from one specified location to another. In their cross-country experiments performed with the ALV, they had been experimenting with two independent behaviors for road-following and obstacle avoidance and found a difficulty of sharing the information for each behavior because behavior was designed to meet the goal of maintaining modularity between behaviors. They also found that the explicit plan in the road-following behavior was hard to handle for unexpected situations. They then proposed a *Fine-grained behavior* and *Internalized plan* in order to minimize information loss between behaviors. Internalized plan used a gradient description of a plan, so they could avoid unnecessary abstraction.

On the other hand, M. Mataric tried to integrate a map representation into a reactive, subsumption-based mobile robot.[11] This work needed no specialized behavior, so she could remove the distinction between the control program and the map. The robot's sensors and its navigation behavior were treated as atomic elements in the topological representation. In this architecture, distributed over a collection of behaviors, the map itself performs constant-time localization and linear-time path planning. A. Saffiotti, E. Ruspini and K. Konolige proposed a fuzzy controller for such a mobile robot that could take abstract goals into consideration.[12] Since multiple behaviors want to control the same actuator simultaneously, such a fuzzy blender can play an important role in resolving these conflicts. Our approach is an integrated approach to those works.

2. PROBLEM CONSIDERATIONS

We had learned some lessons from the experiments in developing an outdoor navigation for the '93 Taejon World Exposition. The following paragraphs are the problem considerations in the design processes of the proposed control architecture:

Graceful Recovery of Robot's Posture: Since it is difficult to localize the position of a robot and to obtain precise metric features, we adopted a topological map which represents the world as a graph of nodes and arcs. The nodes are distinctive places in the environment and the arcs represent paths between places. We expanded a topological map into a pseudo-sequential machine that is represented as a tuple ⟨node, percept, node⟩ and we regarded it as navigation on a topological map. At a location a robot can extract certain features, which can be handled as *percepts*. If we obtain perfect features, navigation on a topological map is equivalent to operation of finite state automata. Unfortunately, it easily localizes a robot's position even if it navigates via a topological map.

We faced a few problems: all features are extracted with uncertainty, some features can be missed, and some features can be duplicated. A navigator must be able to handle these problems, otherwise robots can never escape from a trap. As an example, most robots (except DERVISH[13]) entered in the AAAI '94 Robot Competition had to retry several times due to false estimated positions. Hence we propose the concept of a *fuzzy state* which refers to the approximate position on a map.

Cooperation: The programming of individual behavior itself is known to be easy, but the programming of a meaningful task with behaviors is not easy because these behaviors must cooperate in the most efficient way. For this reason, although several reactive or behavior-based approaches were reported as successful, they applied only for small application areas such as obstacle avoidance, wandering without collision, and simple indoor navigation. In addition, behind these successes were large workloads which required developers to spend many hours testing and re-configuring relationships between behaviors.

Who is responsible for the efficient conduct of the overall physical action for a meaningful task? In the subsumption architecture,[2] interconnection wires have all responsibilities. Behaviors are layered and higher level behaviors are able to control inputs and outputs of lower level behaviors. Therefore, every individual behavior must know the meaning of the input and output signals of other behaviors, and developers must re-wire behaviors' signals whenever the task to be executed is changed. We propose a *Coordinator* in order to elicit cooperation between or among behaviors.

Uniform Appearance: We can say that the behavior-based approach is superior to the previous approaches only if a complex task splits into a number of simple tasks, and then each simple task can be programmed and tested individually. To do this, it must have a uniform interface with the other components, and it can hide information effectively. Consequently we permit only one structure of behaviors.

Information Loss: In a non-layered architecture, vital information can be lost. For example, suppose that you designed two independent behaviours. Both create their own output independently, so the output of one can not be compatible with the other. Therefore, each behavior must not abstract information too much before sending it to a blending module. We propose a *two-stage blender* in order to blend control outputs and to mediate the compatibility with the other behavior.

3. INTEGRATED CONTROL ARCHITECTURE

The reason why the integration of both the planning and reactivity is necessary to build an autonomous mobile robot is straightforward. Before describing our control architecture, it is better to discuss how both are integrated. Figure 1 shows some possible approaches. Figure 1(a) shows a conceptual model of Arkin's the Autonomous Robot Architecture[6] that incorporated the motor schemas (behavioral modules) within a planning framework. Since there was an explicit mission planner, it was able to effectively communicate with human operators and handle the symbolic knowledge when necessary. Our approach adopted these features. In such an approach, the emphasis on the planner's role tends to weaken the reactivity since it is difficult to predict the response time of *AI* methods while a robot operates in real-time. In order to avoid this problem we designed some behaviors by using "any-⟨dimension⟩" algorithms[10] that halt when they have reached a certain threshold along such a dimension.

Figure 1(b) shows a Payton's approach where one or more behaviors were able to interpret *internalized route plans* in the form of gradient field representations.[8] They can be treated as planners. One of them can be expanded to a human interface if necessary.

Figure 1(c) shows a homogeneous framework where low-

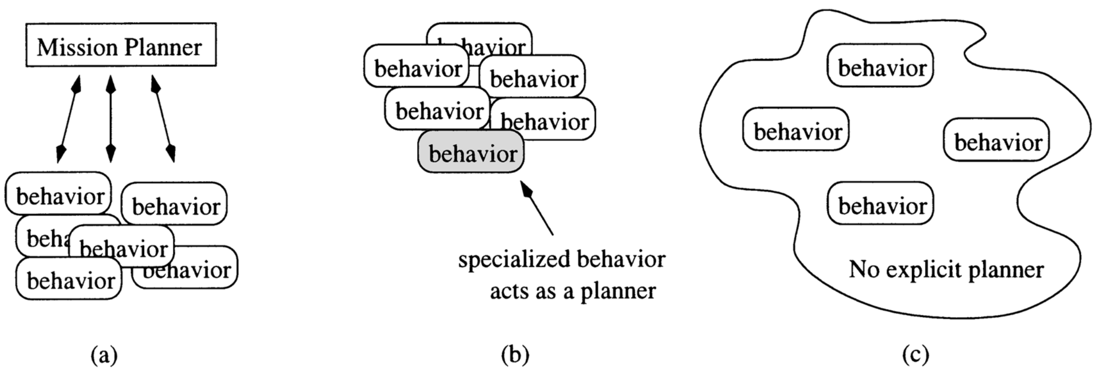


Fig. 1. Integrated approaches: (a) Incorporation of the explicit planner within a behavior-based framework. (b) Homogeneous framework with explicit planners. (c) Homogeneous framework without explicit planners.

level control and high-level deliberation activities could be tightly intermingled. Mataric introduced a distributed map representation that merges directly into a homogeneous subsumption-based system, thus eliminating the need to separate the planning and excecution parts of the system.[11] In this category, it is also possible to build a robot control system by using the same representation method for both levels.

3.1. *The proposed control architecture*

The proposed architecture is similar to that of Figure 1(a). *High-level Task Executor* is a planner with a topological map that is not concerned with locomotion details and acts as a human interface which presents a robot's state verbally and obtains commands. It is important to mask the information about behaviors; otherwise a human operator must know all of the behaviors. So we designed *High-level Task Executor* to hide whole information about behaviors and to give multimodal human interfaces like voice, gesture, and wireless control pads. Robots can be controlled by simply sending topological descriptions through one of these interfaces.

We designed the architecture in order to effect multiple complex tasks; therefore, a certain behavior may be activated or deactivated depending on the task. We designed *Coordinator* to assume the responsibility for activation control of behaviors. Once the *Coordinator* decides which of behaviors are necessary for achieving a certain task, these selected behaviors are activated by a real-time kernel. The behaviors activated by the *Coordinator* take full responsibility for locomotion of a robot.

Individual behavior is a stand-alone and self-contained task, so it needs not any other information from the other modules, except from the sensors themselves. In other words, the relations between behaviors are very weak in comparison with the subsumption architecture where higher level behaviors usually suppress input signals of lower behaviors and inhibit output signals of lower behaviors. In our case, behavior input and output signals never need to be suppressed or inhibited.

However, this causes another problem. Because all behaviors work independently, their control outputs conflict with each other as a result of attempts to control the same actuator simultaneously. In order to resolve these conflicts a *two-stage blender* was designed to determine the merits of a *winner-take-all strategy* and integration techniques such as *Potential Field Method* or *Fuzzy Logic*.

In addition, there was *shared memory* to minimize information loss between behaviors and *sensory data queue* to provide various demands for all behaviors. Since the instruction set consists of statements that reference the locations on a topological map, the planner must know the position. *Position estimator* was designed to predict a robot's position and orientation.

As described above the architecture has seven components as shown in Figure 2. Almost half of them, Sensory data queues, Behaviors, and Blender, are for behavior-based control. The others are for planner-based control. The *Coordinator* guides behaviors in accordance with the results of the *Planner.*

3.2. *Behavior-based control*

3.2.1. Behavior. As stated above, a behavior works independently without interaction with other behaviors. Because individual behavior is usually designed to achieve only a simple and specific task, it is important to coordinate multiple behaviors in order to accomplish a meaningful action. In order to interact with neighboring components each behavior has six links* as shown in Figure 2. Each behavior is able to obtain data from two input sources: *Sensory data queues* and *Shared memory. Sensory data queues* deal with raw sensory data and *Shared memory* deals with the information made by behaviors. A behavior's results can be stored in *Shared memory* in order to minimize the information loss between behaviors, but they also must be sent to the *Position estimator* to predict the robot's position and orientation. Most behaviors produce their control signals for *two-stage blender* in form of *Vector* or *Permission_Ring*. There are behaviors that do not produce any control outputs except by-products like topological features. Figure 3 shows three kinds of behaviors.

The link from *Coordinator* to *EDF-Based Real-Time Kernel* is not connected to behavior; however, in the view of an operating system a behavior is a process, so it can be scheduled or blocked by a real-time kernel. With our proposed architecture, behaviors can produce control out-

* One of them is an indirected link.

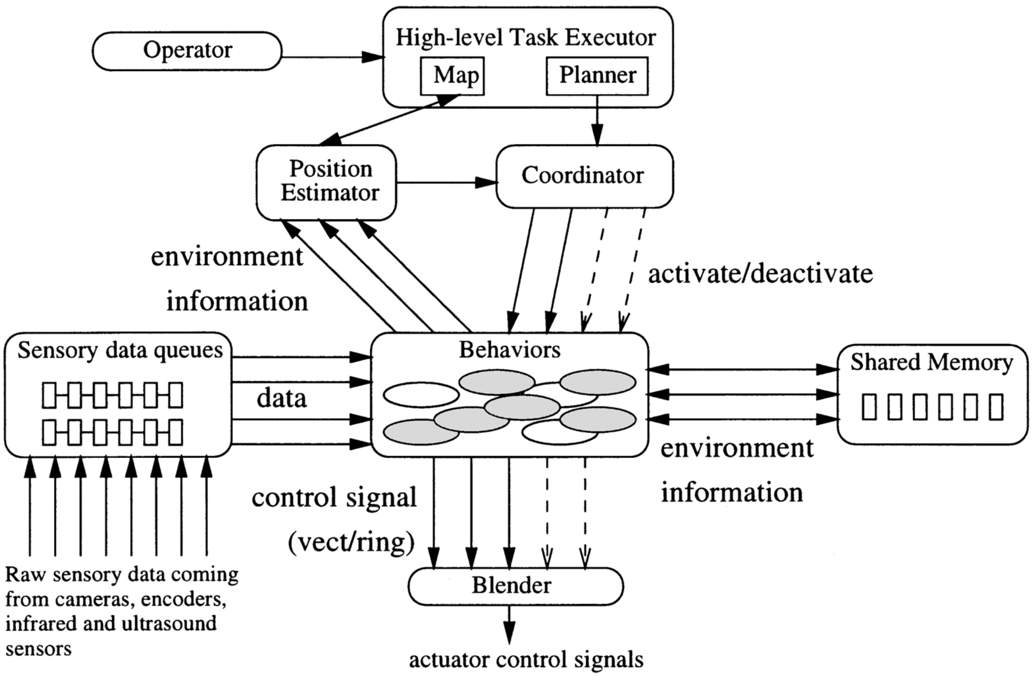


Fig. 2. Control architecture of CAIR-2.

puts only when they are scheduled. In order to manage behaviors the *Coordinator* sends *wake-up* or *block* signals in accordance with the task.

3.2.2. Sensory data queues and shared memory. Before explaining the *Sensory data queues*, let's consider a behavior that extracts a corridor from a certainty map made by the latest 60 readings of an ultrasound sensor. *Queue* is one of the best data stuctures for solving such a problem. We have two choices to place the queue: inside or outside of behavior. If it is placed in a behavior, it must run forever and cannot pause or it will produce unwanted results while the queue is being filled up. Hence we put the queue outside of behaviors so that behaviors can produce their outputs without delays for filling up the queue. In addition they can minimize their internal states. To do this, we insert *Sensory data queues* between the sensor and the behavior, even if it is common knowledge that the behavior is connected directly to the sensor and actuator. Therefore, the time required to set up a queue can be saved and the time and the space for handling a queue can also be saved if two or more behaviors use the same sensor.

It is desirable that the response time is stable regardless of the computation requirement. To do this, multiple behaviors could be developed in accordance with the response time. Therefore, if in a hurry, the behavior that requires fewest readings would be activated even if the behavior that requires more readings produces more precise results.

We implemented *Sensory data queues* as a circular queue, but in the view of behaviors *Sensory data queues* is not a queue but rather a stack because they are designed to set the pointer of the queue to always indicate the latest readings.

Shared memory is used to store the information produced by behaviors. It is a store and is given a memory block to behavior on demand. Although it is a compound structure, it is treated as an atomic element by *semaphore*. Therefore, it can be asynchronous and the information in it is non-consumable and can be overwritten. All incoming pieces of

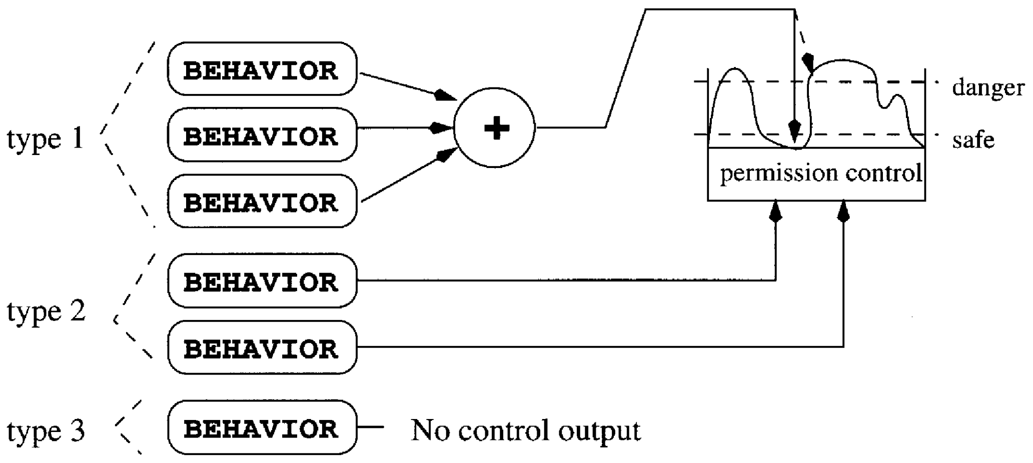


Fig. 3. Overview of *Two Stage Blender* and three kinds of behaviors.

information are timed in order to keep the latest one and to discard it when it is worn out.

3.2.3. Two stage blender. Multiple control signals coming from activated behaviors would cause many conflicts. In order to resolve them many methods have been derived for use in the reactive robotics. They can be divided into two categories: Integration and Winner-take-all:

Integration: One famous set of solutions integrates these signals by using the potential field method, fuzzy logic or vector summation.[6,12,14] The control signals are easy to distort when their physical dimensions are different or there is a malfunction in some behaviors. Moreover, the behaviors' outputs can be abstracted too much before being sent to a blending module, so the blending module will face a lack of information[15]; this demonstrated a limitation of the potential field method for mobile robot navigation.

Winner-take-all: The other solution set is a winner-take-all strategy which selects out of control signals and uses it without modification.[2,16] The selected signal easily disregards minor but potentially critical information[8] and showed that a winner made undesirable control outputs, and it is commented that this was caused by information loss due to incompatibility between behaviors.

Since both cases have their drawbacks, we considered them simultaneously when applied to the blender. Each behavior has two kinds of output format: *Permission_Ring* and *Vector. Vector* represents the desired direction and velocity while *Permission_Ring* represents those directions which are prohibited.

Before the navigation, the *Coordinator* decides the importance of each behavior and endows it with a weighting factor. Each behavior produces its output in one of two forms. In the case of *Vector*, the desired orientation and velocity are calculated and then multiplied by the weighting factor. All orientations and velocities are summed up to generate a single resultant vector

$$\theta = \arctan 2(\Sigma x_i\,\omega_i,\ \Sigma y_i\,\omega_i) \tag{1}$$

$$v = \sqrt{(\Sigma x_i\,\omega_i)^2 + (\Sigma y_i\,\omega_i)^2} \tag{2}$$

where $\Sigma\omega_i = 1$, $(x_i y_i)$ is the output of behavior i and ω_i is the weighting value of behavior i. The robot's new direction and velocity are then adjusted to θ and v. This adjustment can be cancelled and modified when the robot's direction is prohibited.

Certain behaviors can produce their output in the form of *Permission_Ring* which is designed to represent those directions in which a robot is prohibited to go. A *Permission_Ring* is composed of 32 arcs of 11.25° in which a confidence measure is stored. The confidence measure denotes how many seconds are left before a collision with obstacles may occur. More than 10 seconds is treated as 10 seconds. The time is more important than the distance to the obstacle. For example, in a situation where an obstacle is 30 cm away and the diameter of a robot is 60 cm, the physical distance between them is very short, so the robot is in danger. However the logical distance varies in accordance with the velocity of the robot. When it is 30 cm/sec, only one second is left before collision, but when it is 3 cm/sec, ten seconds are left. Therefore, safety depends on both the distance and the velocity. Since multiple behaviors work, multiple *Permission_Ring* are produced. They adjust their coordination system. All arcs that are placed in the same direction are merged into one arc by a *min* operation that chooses the minimum value among them. Before applying a *min* operation the confidence measures are slightly changed according to the control frequency. In other words, all behaviors do not operate synchronously which means that the completion of a single loop of the control cycle takes more or less time than the others. Therefore, the slower ones must interpolate their value to adjust to the fastest one.

The desired velocities $[v_r, w_r]^T$ are finally calculated by the *Vector* and *Permission_Ring*. Direct delivery of the velocities to the actuators causes a serious positioning error: the higher the acceleration, the more wheel slippage occurs due to friction. Moreover, it also leads to an electric power problem. Mobile robots use batteries, however, and the capability of batteries is limited and may be insufficient for another action. A transient overload of actuators must be avoided Therefore, we introduced an open-loop controller for smooth locomotion.

The blender makes control signals every T seconds and it assumes that the robot goes as far as P_r:

$$P_r = \begin{bmatrix} v_r(\sin(\theta + w_r) - \sin\theta)/w_r \\ -v_r(\cos(\theta + w_r) - \cos\theta)/w_r \\ w_r \end{bmatrix} T \tag{3}$$

However, the robot usually does not advance P_r because this calculation is made without considering any dynamics. The estimated trajectory should be P_c which might differ from P_r. The difference between them is the error vector, P_e:

$$P_e = \begin{bmatrix} x_e \\ y_e \\ \theta_e \end{bmatrix} = \begin{bmatrix} \cos\theta_r & \sin\theta_r & 0 \\ -\sin\theta_r & \cos\theta_r & 0 \\ 0 & 0 & 1 \end{bmatrix} (P_r - P_c) \tag{4}$$

The error vector P_e can be used in computing the control signals in the next run. The following equation was derived[17] for mobile robots to track stable

$$\begin{bmatrix} v \\ w \end{bmatrix} = \begin{bmatrix} v_r \cos\theta_e + K_x x_e \\ w_r + v_r(K_y y_e + K_\theta \ \sin\theta_e) \end{bmatrix} \tag{5}$$

where, K_x, K_y, and K_θ are positive constants.

3.3. *Planner based control*

We have shown that it is difficult to estimate the position of a robot and easy to distort metric features like the distance between places. One of the good solutions of the inevitable problems of dealing with movement uncertainty in mobile robots is the use of a topological map. Topological maps represent the world as a graph of vertices and edges. The vertices are distinctive places in the environment and the edges represent paths between places. The vertices can be a room, lobby and corridor. The map is represented by an undirected graph T that is an ordered pair (S, E), where S is a set of places and E is a set of multisets of two elements from *S*.

There are two components related to planning; *Position estimator* and *Planner itself*. The Position estimator is to estimate the position on a topological map. The Planner can not make a plan completely before execution due to the incompleteness of information about the environment, therefore, we make a partial plan whenever new percepts from sensors arrive.

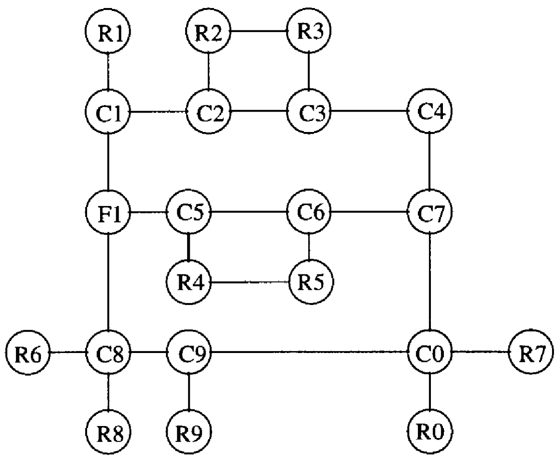
3.3.1. Position estimator. Let's consider that there is a robot in a room and can obtain many topological features necessary to confirm its position on a topological map while it is moving to another room. The navigation thus is the same as graph traveling, so we have expanded a topological map into a pseudo sequential machine in order to consider navigation as the operation of such a sequential machine which can be represented as a function from $S \times U$ into S, where U is a power set of a finite nonempty set P of percepts (topological features) coming from behaviors. A place in S is treated as this machine's state and a percept is treated as an input. It can be exactly the same as with deterministic finite automata if all possible percepts can be collected. Unfortunately, the inputs are determined uncertainly and they can be lost or duplicated, so we apply neither nondeterministic nor deterministic sequential machines for modeling navigation. William G. Wee introduced fuzzy automata to handle fuzzy values and fuzzy states.[18] We adopted fuzzy automata to estimate the position of a robot.

Fuzzy sequential machines The position estimator is a modified fuzzy sequential machine F, that is a system (u, S, f) where u is an input measure function from U to $[0, 1]$, S is a finite nonempty set of places, and f is a state transition function from $S \times u(U) \times S \times T$ into $[0, 1]$ where T indicates time. There is neither a start place nor an initial distribution function because *a priori* information is not given. Most sequential machines usually exclude the T since they are stationary. However, in our case T is important because percepts have not arrived in the order we want because of the variance of the complexity. Therefore, we cannot handle these percepts directly and we must devise a tool for percepts. To do this, we considered the difficulties. The first difficulty is that they are extracted with uncertainty. The second is that features can be missed. The third is that features can be duplicated. Finally, they are not obtained in a deterministic order; in other words, the first percept in the spatial order cannot arrive first in the temporal order.

Table I shows the desired percepts for each arc in the graph shown in Figure 4. The table has three major columns: *Action*, *Stationary*, and *Transit*. The *action* column represents an arc from the place S to the place D and a robot's orientation is O which can be determined by a digital compass attached to robot's head. The *stationary* column denotes the percepts that a robot can obtain in the place S, where F, L, R, and B represent whether objects are in the

Table I. Desired percepts.

Action			Stationary								Transit		
S	O	D	F	L	R	B	S	N	P	M	L	R	W
c1	n	r1	d		o	o	3	f		c			d
c1	s	f1	o	o		d	3	r	c				w
c1	e	c2	o	d	o		3		r	f	d	o	w
c2	n	r2	d	o	o		3		c	c			d
c2	e	c3	o	d		o	3	c	r		d		x
c2	w	c1	o		d	o	3	c		r		d	x
c3	n	r3	d	o	o		3		c	c			d
c3	e	c4	o	d		o	3	c	r		d		x
c3	w	c2	o		d	o	3	c		r		d	x
c4	s	c7	o		o		2			c		o	w
c4	w	c3	o	o			2		c		o		w
c5	s	r4	d	o	o		3		c	f			d
c5	e	c6	o		d	o	3	f		r		d	x
c5	w	f1	o	d		o	3	c	r				w
c6	s	r5	d	o	o		3		c	c			d
c6	e	c7	o		d	o	3	c		r		d	x
c6	w	c5	o	d		o	3	c	r		d		x
c7	n	c4	o	o		o	3	c	c		o		w
c7	s	c0	o		o	o	3	c		c		o	w
c7	w	c6	o	o	o		3		c	c	o	o	w
c8	n	f1	o	d	o	d	4	r	r	c			w
c8	s	r8	d	o	d	o	4	f	c	r			d
c8	e	c9	o	o	d	d	4	r	f	r	o	d	w
c8	w	r6	d	d	o	o	4	c	r	f			d
f1	n	c1	o		o	o	r	c		c			w
f1	s	c8	o	o		o	r	c	c				w
f1	e	c5	o	o	o		r		c	c			w
r1	s	c1	d				r						d
r2	s	c2	d	d			r		r				d
r2	e	r3	d		d		r			c			d
r3	s	c3	d		d		r			r			d



C1	*Corridor*	(R1 N) (F1 S) (C2 E)
C2	*Corridor*	(R2 N) (C3 E) (C1 W)
C3	*Corridor*	(R3 N) (C4 E) (C2 W)
C4	*Corridor*	(C7 S) (C3 W)
C5	*Corridor*	(R4 S) (C6 E) (F1 W)
C6	*Corridor*	(R5 S) (C7 E) (C5 W)
C7	*Corridor*	(C4 N) (C0 S) (C6 W)
C8	*Corridor*	(F1 N) (R8 S) (C9 E) (R6 W)
C9	*Corridor*	(R9 S) (C0 E) (C8 W)
C0	*Corridor*	(C7 N) (R0 S) (R7 E) (C9 W)
F1	*Foyer*	(C1 N) (C8 S) (C5 E)
R1	*Room*	(C1 S)
R2	*Room*	(C2 S) (R3 E)
R3	*Room*	(C3 S) (R2 W)
R4	*Room*	(C5 N) (R5 E)
R5	*Room*	(C6 N) (R4 W)
R6	*Room*	(C8 E)
R7	*Room*	(C0 W)
R8	*Room*	(C8 N)
R9	*Room*	(C9 N)
R0	*Room*	(C0 N)

Fig. 4. Topical topological map with the description.

front, left, right, and rear view, respectively, S denotes the property of the junction, and N, P, M represent the type of the neighbors. The *transit* column denotes the percepts to be obtained when a robot arrives at the place D where L, R, and W represent which object had been recognized and disappeared in the left, right, rear view. Symbols in these tables are described in Table II.

All behaviors are able to send some topological features to the Position estimator. These can be represented as a stream of a percept p_i which belongs to one of the classes of percepts; in other words, one of $\{FLRBSNPM, LRW\}$, and i denotes the temporally ordered sequence. They are not ordered spatially, so they must be segmented to make an element of the power set U. All percepts in *Transit* category are used to segment this stream.

Given a segmented section, we remove the duplicated features as follows:

$$f_i^j(a) = Max_{k=i}^{j}\{p_k \mid p_k \text{ belongs to the class } a\} \qquad (6)$$

Next, we evaluate the input measure function u as follows.

$$u_i^j(s, d, t) = \sum_{k \in P} \{\rho_k \, likelihood(f_i^j(k), d(s, d, k))\} \qquad (7)$$

where $d(s, k)$ denotes the desired percept k at the arc (s, d) and $likelihood(a, b)$ is a likelihood measure function between two percepts and $\Sigma_{k \in P}\{\rho_k\} = 1$.

State transition function. In navigating, suppose that a robot takes a percept $p_i \in P$. It can affect multiple states. This is natural. For example, we are also confused in a strange street. A man stands in a strange street and sees two red buildings and he then decides to go to the left red building. While walking, he obtains new information that can help to confirm his decision. If the decision is right, the confidence value will increase, otherwise he will worry about the decision and eventually change his mind. In this case, the man has two states for each red building. As information collects, only one candidate remains. The following shows how to increase the confidence value.

Whenever a new percept p_i is arrived at, the input measure function u_i^j is re-evaluated and a state evaluation function f is also re-evaluated as follows:

$$f(s, d, t) = (1 - \rho)u_i^j(s, d, t) + \rho CV(s, t - 1) \qquad (8)$$

where CV is a certainty value for the place s and ρ is a positive constant between 0 to 1. When the u_i^j exceeds a certain threshold, the time stamp t is increased and the CV is updated as the value of the state evaluation function f like this.

$$CV(d, t) = Max_{s \in S}\{f(s, d, t)\} \qquad (9)$$

At time t we use the estimated position of a robot as follows:

$$P_{est}(t) = s \text{ such that } \exists_{s \in S} F(s, d, t) \geq Max_{x, y \in S}\{f(x, y, t)\} \quad (10)$$

Table II. Percepts used in our architecture.

	Stationary		Transit	
	FLRBS	NPM	LR	W
d	found a door		passed by a door	passed a doorway
o	found an open		passed by an open	
‘ ’	found a wall	prohibited	detected nothing	
r	in a room	room		
f		foyer		
c		corridor		
e				passed a junction
x			passed a junction if door opens	

3.3.2. Planner itself. Once the position is determined, a planner must decide the destination. This work is relatively simple and easy. We adopted the A^* algorithm whose cost function uses three pieces of information; *Hard*, *Risk*, and *Dist*. *Hard* depends on the type of place and its neighbor, for example, passing through a doorway is more difficult than passing through a corridor. *Risk* reflects the dynamic feature of an environment. The environment may change drastically. Therefore, some paths can be broken without notice. For this reason, *Risk* is introduced to reflect the number of alternative paths. *Dist*, is the distance between nodes, but it can be omitted since it is not given initially. A cost function is described as follows:

$$Cost_{ij} = \rho_1 Hard_{ij} + \rho_2 Risk_{ij} + \rho_3 Dist_{ij} \tag{11}$$

where i and j are adjacent places.

Whenever new information is gathered, the cost function is re-evaluated and the results are stored in a one-dimensional array. Therefore, planning can be treated as table look-up like this.

`d=Des[s]`, where `s` is the current estimated place and `d` is the place a robot goes next.

3.3.3. Simulation. Now, we present the performance of our positon estimator. Table III shows the simulation results of the *Position estimator*. We used the same instruction "move to *Room 1* from here (*Room 0*)" in the simulation, but used different simulator's parameters.

In the *Scenario A*, the simulator did not make any noise, so the input measure function u could be 1.00. In the *Scenario B*, `R7`'s door was sensed as a corridor, `R6`'s door was not detected, and the other features were obtained with 20% noise. In the *Scenario C*, we used the same parameters of *Scenario B* except `R9`'s door was not detected. `R9`'s door was a very important feature since missing this feature caused it to fail to recognize the place `C9`. However, the simulator shows that overall performance was not degenerated even if the door was missed.

3.4. *Coordinator*

As mentioned above, the position estimator is waiting to receive new percepts from behaviors and the planner just recommends which direction is desirable. They both are passive modules; in other words, they never produce control signals for locomotion. Only behaviors are able to move the robot. Once the destination is determined by the planner, the coordinator decides which physical action needs to arrive at the destination. Since such a physical action cannot be achieved by a single behavior, the coordinator selects multiple behaviors that should cooperate together to perform this action. To do this, it produces a *wake-up table* that contained those behaviors which should be scheduled by a real-time operating system. The wake-up table has three columns for each physical action.

```
[1] Pre_activation
[2] Fire_condition
[3] Post_activation
```

Before explaining the above items, let's consider a situation that there is a robot in a room and it is trying to leave the room. Its first task is to examine the room in order to find a door. The second is to find a door, and then to leave through the door. What is the most important task in this

Table III. Indoor navigation in a simulated environment

	Scenario A			Scenario B			Scenario C		
T	Decision	u	CV	Decision	u	CV	Decision	u	CV
1	r8 ⇒ c8	1.00	0.70	r8 ⇒ c8	0.98	0.68	r8 ⇒ c8	0.98	0.68
	r9 ⇒ c9	1.00	0.70	r9 ⇒ c9	0.98	0.68	r9 ⇒ c9	0.98	0.68
	r0 ⇒ c0	1.00	0.70	r0 ⇒ c0	0.98	0.68	r0 ⇒ c0	0.98	0.68
	r4 ⇒ c5	0.85	0.60	r4 ⇒ c5	0.82	0.58	r4 ⇒ c5	0.82	0.58
2	c0 ⇒ c9	1.00	0.91	c0 ⇒ c9	0.82	0.78	c0 ⇒ c9	0.82	0.78
	c8 ⇒ r6	0.57	0.61	c7 ⇒ c6	0.75	0.57	c7 ⇒ c6	0.75	0.57
	c9 ⇒ c8	0.50	0.56	c8 ⇒ r6	0.40	0.48	c8 ⇒ r6	0.40	0.48
	c5 ⇒ f1	0.50	0.53	c9 ⇒ c8	0.32	0.43	c9 ⇒ c8	0.32	0.43
3	c9 ⇒ c8	1.00	0.97	c9 ⇒ c8	0.95	0.90			
	c5 ⇒ f1	1.00	0.86	c6 ⇒ c5	0.95	0.84			
	c6 ⇒ c5	1.00	0.82	c5 ⇒ f1	0.95	0.79			
	c8 ⇒ r6	0.57	0.57	cr ⇒ c2	0.60	0.54			
4	c8 ⇒ f1	1.00	0.99	c8 ⇒ f1	0.77	0.81	c8 ⇒ f1	0.77	0.67
	f1 ⇒ c1	0.50	0.61	f1 ⇒ c1	0.68	0.71	f1 ⇒ c1	0.68	0.59
	c0 ⇒ c7	0.57	0.49	c1 ⇒ r1	0.55	0.53	c1 ⇒ r1	0.55	0.46
	c1 ⇒ r1	0.32	0.37	co ⇒ c7	0.50	0.44	c0 ⇒ c7	0.50	0.41
5	f1 ⇒ c1	1.00	1.00	f1 ⇒ cl	0.90	0.87	f1 ⇒ c1	0.90	0.83
	c1 ⇒ r1	0.82	0.76	c1 ⇒ r1	0.76	0.74	c1 ⇒ r1	0.76	0.71
	c8 ⇒ f1	0.68	0.55	c8 ⇒ f1	0.57	0.48	c8 ⇒ f1	0.57	0.48
	c7 ⇒ c4	0.50	0.50	c7 ⇒ c4	0.45	0.45	c7 ⇒ c4	0.45	0.44
6	c1 ⇒ r1	1.00	1.00	c1 ⇒ r1	0.92	0.91	c1 ⇒ r1	0.92	0.90
	f1 ⇒ c1	0.75	0.69	f1 ⇒ c1	0.73	0.65	f1 ⇒ c1	0.73	0.65
	r4 ⇒ c5	0.57	0.43	r4 ⇒ c5	0.52	0.40	r4 ⇒ c5	0.52	0.40
	c2 ⇒ r2	0.50	0.40	c8 ⇒ f1	0.42	0.38	c8 ⇒ f1	0.42	0.37

Table IV. Some entries of the *wake-up* table.

Action	Pre_activate	Fire_condition	Post_activate
Exit Room	BH_AVOID_COLL BH_AVOID_OBST BH_OPEN_SPACE[1]	BH_FIND_DOOR	BH_EXIT_DOOR BH_COLL_AT_DOOR[2]
Turn Left	BH_AVOID_COLL BH_AVOID_OBST BH_FLOOR_TRACK BH_GO_FORWARD	BH_FIND_OPEN	BH_SET_DIRECTION BH_AVOID_COLL BH_AVOID_OBST
Go Hallway End	BH_AVOID_COLL BH_AVOID_OBST BH_FLOOR_TRACK BH_GO_FORWARD	BH_HALLWAY_END	

[1] BH_OPEN_SPACE_EXPLORER. [2] BH_AVOID_COLL_AT_DOOR.

scenario? We thought it is to find a door. The coordinator decides one or more behaviors as the representatives, which make topological features belong to *Transit* column of Table I. The Fire_condition denotes these representatives. In order to assist these representatives, many behaviors are assigned to the Pre_activation or the Post_activation according to the state of the representatives. The Pre_activation denotes a group of behaviors which are activated before the representatives do not complete their own task. The Post_activation denotes a group of behaviors which are activated after the representatives finish the task. They complete the action and prepare for the next action.

Table IV shows some entries of the *wake-up* table. In the case of leaving a room, a door is the most important feature. BH_AVOID_COLL, BH_AVOID_OBST and BH_OPEN_SPACE_EXPLORER are activated to assist BH_FIND_DOOR to find a door effectively. In other words, they move a robot to the best place where a door can be easily found by the vision system. Once a door is found, they are deactivated and BH_EXIT_DOOR and BH_AVOID_COLL_AT_DOOR are activated to pass through the door.

3.4.1. An Example: Task "Turn Left". In this section we will describe "Turn Left at the *i*th corner" in order to present how architectural components actually cooperate to complete a specific task. Such a task can be represented as Figure 5 in the view of our proposed architecture.

In the figure, six behaviors are working to find corners, to correct robot's posture and to turn left. BH_AVOID_COLL requests *Sensory Data Queues* to send one reading of ultrasound sensor and infrared sensor respectively and makes *Permission_Ring* which is sent to the second stage of blender. BH_AVOID_OBST and BH_GO_FORWARD use data obtained from both *Shared memory* and *Sensory Data Queues*. Their control outputs are added into *vector sum*. BH_FLOOR_TRACK and BH_FIND_OPEN don't produce any control outputs, but their symbolic information is stored in the *Shared memory*. This informatin is used by the other behaviors. For example, the *number of opens* is used by BH_SET_DIRECTION in order to decide when it starts to turn robot left. Most of the behavior's symbolic information is sent to *Position estimator* in order to estimate the robot's posture.

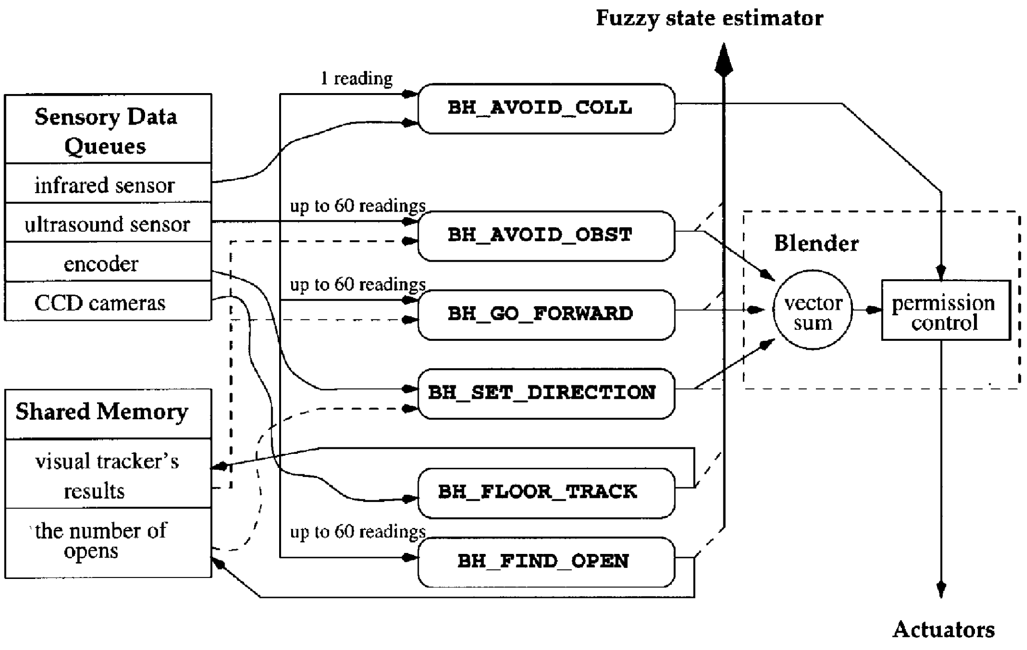


Fig. 5. Architectural description of a task: "Turn *i*th Left".

4. EXPERIMENTAL RESULTS

When a robot navigates real space, several kinds of data are involved such as velocity, position, and sensory data, so it is difficult to present the result on the two dimensional space like this paper. For this reason, many scientific and technical societies offer another method like video proceedings, in which more realistic and dynamic results can be presented. Some of our results are also dynamic, so these can be presented in a video.[19]

We performed many experiments to demonstrate the ability of our architecture. Two of the major experiments were in the '93 Taejon World Exposition and the 4th Annual Mobile Robot Competition sponsored by IJCAI and AAAI. In this chapter, we present the robot competition and some related results.

4.1. IJCAI Robot Competition

The Robot competition was held in Montreal, Canada, on 21 through 24 August 1995, in conjunction with the 1995 International Joint Conference on Artificial Intelligence. The competition consisted of two separate events: (1) Office Delivery and (2) Office Cleanup. We took part in the first event.

In addition to the traditional goal-directed navigation, the first event of the robot competition was designed to promote the ability of robots to detect when there is a problem and ask for help through effective robot-human interaction. The event took place in a re-creation of a typical office environment with partitioned offices as shown in Figure 6. The robots were asked to follow a series of instructions that told them to which room they were supposed to go. The instructions consisted of statements such as "exit the room and turn left", "go to the end of the hallway and turn right".

4.1.1. CAIR-2's run. At the final round, we set the maximum velocity of CAIR-2 to half because CAIR-2 had been placed first at the first preliminary round and we thus believed that CAIR-2 was fast enough. Lower velocity can give a lot of advantages such as safety and sensor accuracy. CAIR-2 did the final round (about 51 meters) in 325 seconds. Hence, its velocity was about 15 cm per second.

The environment is similar, in size, shape and furnishings, to the first floor of an office building containing a foyer as well as offices. CAIR-2 starts in *Room E* with only one exit and is given instructions verbally by speech recognition on how to get to the goal room. The 6th instruction was faulty and CAIR-2 must be able to overcome this. No topological map as to the office environment is given. CAIR-2 starts to wander around the room looking for an exit marked with a landmark pattern. It recognizes target patterns in the landmark using vision and keeps tracking the target in real time while approaching the exit to adjust its position and orientation. After confirming the position and orientation several times, CAIR-2 starts to leave the room. After that, CAIR-2 leaves the room safely and it speaks "Exit Room" as a confirmation of its first subtask completed. Now it moves toward its goal room (*Room D*). It knows it must make a right turn at the first junction to take the hallway. It finds two doors using sonar and confirms by saying "Found Door" until it reaches the next junction for another right turn. It knows it is on the correct path to the goal room so far. Measuring the distance from its body to the left and right side walls using ultrasound, it keeps its path around the center of the hallway. It finds a foyer and again confirms it verbally. CAIR-2 now tries to find the third right open where it is supposed to make a right turn as given by initial instructions. However, it cannot find any third open!

CAIR-2 activates a behavior responsible for finding any mistake. It fails in finding any mistake and an alternative path to the goal and concludes that the given instruction was incorrect. CAIR-2 announces it verbally and that is the end of the first phase of the task.

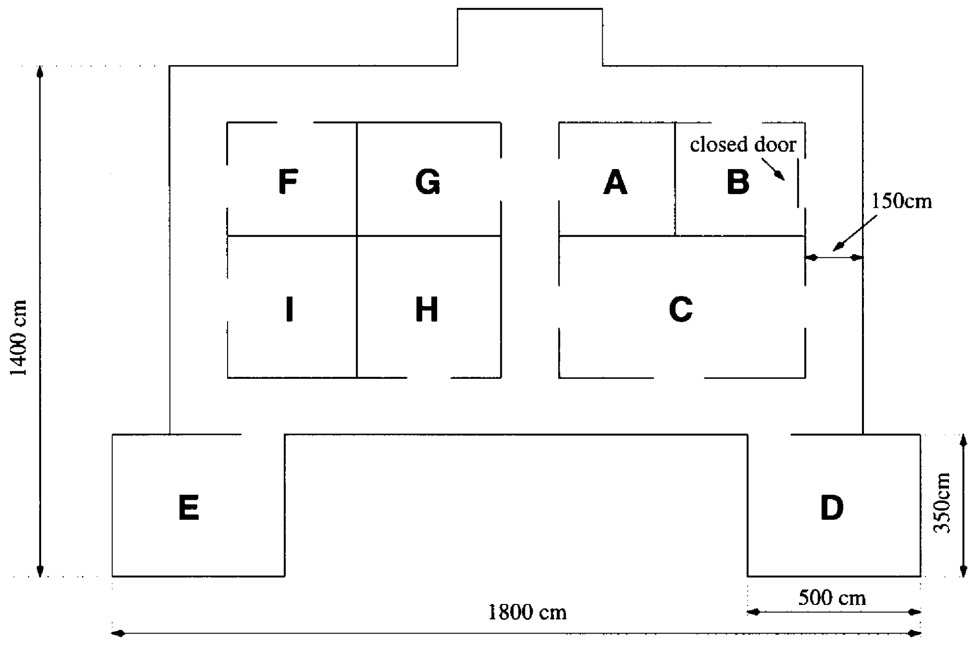


Fig. 6. The competition arena.

The second phase of the task is rather straightforward and similar to the first, except there is no incorrect instruction. CAIR-2 finally approaches the goal room and finds the entrance. CAIR-2 enters the room very carefully by keeping his body at the center of the entrance. No single mistake. CAIR-2 completed his mission better than we expected. Figure 7 shows the final stage of CAIR-2's run.

4.1.2. Behaviors. For the competition, we have used 22 behaviors. In the following, we describe some of them.

BH_AVOID_COLL: avoids the collision by keeping the robot a safe distance from the obstacle using sonar and infrared sensors. The format of the output is the *Permission_Ring*.

BH_AVOID_COLL_AT_DOOR: is similar with BH_AVOID_COLL, but more sophisticated and time consuming. It tries to reconstruct its surrounding environment to remove false detections. It also produces the output in the form of the *Permission_Ring*.

BH_AVOID_OBST: understands the locations of obstacles and avoids them if possible. It uses up to 60 readings of sonar and then builds a certainty grid with these sonar readings. It also uses environmental features made by BH_FLOOR_TRACK through *Shared memory*.

BH_OPEN_SPACE_EXPLORER: moves the robot to an open space and let it wander around.

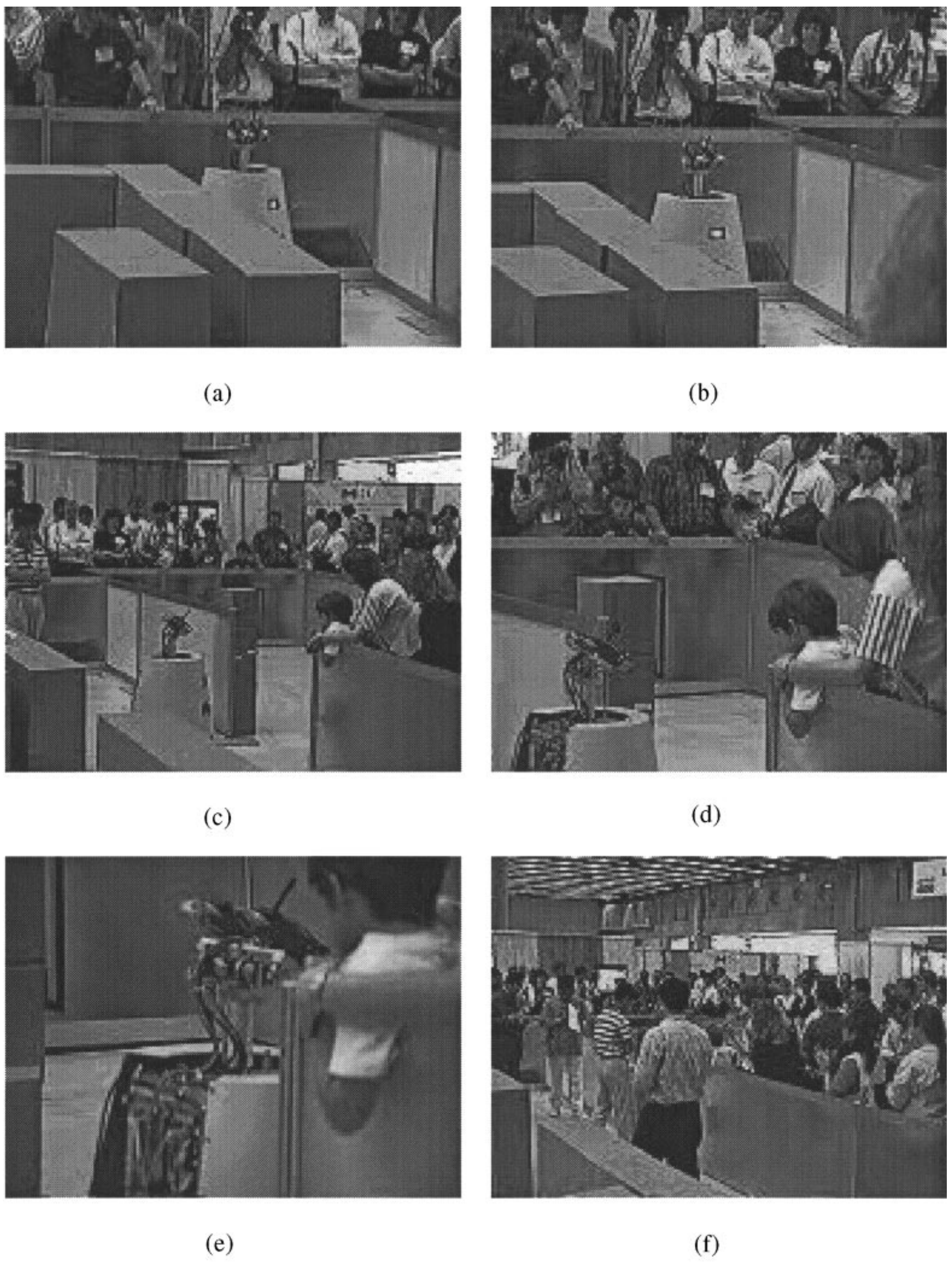


Fig. 7. Indoor Navigation: These pictures were taken during the IJCAI Robot competition by Pete Bonasso and they can be retrieved from the URL http://www.ai.mit.edu/people/dmiller/ijcai/robots-95.html.

BH_GO_FORWARD: measures the distance from the robot body to the left and right side walls using sonar and try to keep its path along the center of the hallway. It can correct the orientation of robot when the computation is available.

BH_PASS_FOYER: find the lobby and pass safely.

BH_SET_DIRECTION: change the direction of the robot when the condition (the number of opens) meets.

BH_FIND_OPEN: find junctions and doors by sonar sensors. We implemented three kinds of door finding algorithms. The difference between them is how many sensor readings are required. Therefore, a fast response time can be accomplished if a faster algorithm can find them. In addition, even if the faster one loses them, they have another chance to be found by slower ones which are more sophisticated and robust. This behavior does not produce the control output but important information for the other behaviors, for example, BH_SET_DIRECTION needs the number of junctions and doors.

BH_HALLWAY_END: find the end of the hallway. It does not produce the control output, but can send information to the coordinator to initiate the next instruction when the end of the hallway is found.

BH_ENTRAP: check the validity of the current instruction and see if there is any mistake when the robot is trapped. In case any mistake is detected they try to solve it. Otherwise, deactivate all currently active behaviors and announce the situation verbally.

5. CONCLUSIONS

In this paper, we presented an integrated robot control architecture in order to maximize the usefulness in developing an intelligent system. We also presented a brief description of the intelligent mobile robot CAIR-2 and the IJCAI Robot competition as an experimental result.

References

1. L.S. Mello and A.C. Sanderson, “And/or graph represenation of assembly plans” *IEEE Trans. on Robotics and Automation* **6**(2), 188–199 (1990).
2. R.A. Brooks, “A robust layered control system for a mobile robot” *IEEE J. of Robotics and Automation* **2**(1), 14–23 (March, 1986).
3. R.G. Simmons, “Structured control for autonomous robots” *IEEE Trans. on Robotics and Automation* **10**(1), 34–43 (1994).
4. H.S. Yang, J. Chung, B. Ryu and J. Lee, “Cair-2: Intelligent mobile robot for guidance and delivery” *AI Magazine* **17**(1), 47–53 (1996).
5. E. Gat et al, “Behavior control for robotics exploration of planetary surfaces” *IEEE Trans. on Robotics and Automation* **10**(4), 490–503 (1994).
6. R.C. Arkin, “Integrating behavioral, perceptual, and world knowledge in reactive navigation” *Robotics and Autonomous Systems* **6,** 105–122 (1990).
7. D.M. Lyons, “Representing and analyzing action plans as networks of concurrent processes” *IEEE Trans. on Robotics and Automation* **9**(3), 241–256 (1993).
8. D.W. Payton, J.K. Rosenblatt and D.M. Keirsey, “Plan guided reaction” *IEEE Trans. on Systems, Man, and Cybernetics* **20**(6), 1370–1382 (1990).
9. R.A. Brooks, “The behavioral language user’s guide” *Ai memo. 1127 AI* Lab., MIT, Cambridge, MA, 1990.
10. D.J. Musliner et al, “Circa: A cooperative intelligent real-time control architecture” *IEEE Trans. on System, Man, and Cybernetics* **23**(6), 1561–1574 (1993).
11. M. J. Mataric, “Integration of representation into goal-driven behavior-based robots” *IEEE Trans. on Robotics and Automation* **8**(3), 304–321 (1992).
12. A. Saffiotti, “Some notes on the integration of planning and reactivity in autonomous mobile robots” *Proc. of the AAAI Spring Symposium on Foundations of Automatic Planning* (1993) pp. 122–126.
13. I. Nourbakhsh, R. Powers and S. Birchfield, “Dervish: An office-navigating robot” *AI Magazine* **16**(2), 53–60 (1995).
14. E. Badreddin, “Recursive behavior-based architecture for mobile robots” *Robotics and Autonomous Systems* **8,** 165–176 (1991).
15. Y. Koren and J. Borenstein, “Potential field methods and their inherent limitations for mobile robot navigation” *Proc. IEEE Int. Conf. on Robotics and Automation* (1991) pp. 1398–1404.
16. D.W. Payton, “An architecture for reflexive autonomous vehicle control” *Proc. IEEE Int. Conf. on Robiotics and Automation,* San Francisco, CA (1986) pp. 1838–1845.
17. Y. Kanayama, Y. Kimura, F. Miyazaki and T. Nohuchi, “A stable tracking control method for an autonomous mobile robot” *Proc. IEEE Int. Conf. on Robotics and Automation* (1990) pp. 384–389.
18. E.S. Santos and W.G. Wee “General formulation of sequential machines” *Information and Control* **12**(5), 5–10 (1968).
19. J. Chung and H.S. Yang “Fast and effective multiple moving targets tracking method for mobile robots” *Video Proc. of IEEE Int. Conf. on Robotics and Automation,* Nagoya, Japan (May, 1995).