An Autonomous Behavior Generation Architecture for Humanoids

Masaharu Shimizu, Takayuki Furuta, and Ken Tomiyama

Aoyama Gakuin University, 6-16-1 Chitosedai, Setagaya-ku, Tokyo, Japan
shimi@artemis.me.aoyama.ac.jp

Abstract. A new concept of a decentralized behavior control architecture for a humanoid termed as Distributed Behavior Arbitration Network is proposed and its extensions are introduced in this paper. In the proposed architecture, each competence of a humanoid is differentiated based on the concept of a behavior-based framework and these differentiated competencies are represented by individual agents. In the extension introduced here, agents can utilize information generated by other agents as well as information received from physical sensors in determining their own behaviors. The proposed architecture is implemented onto humanoids constructed in our laboratory and is demonstrated to be able to generate complicated and intelligent behaviors.

1 Introduction

An autonomous control architecture for humanoids that is called Distributed Behavior Arbitration Network (DBANet) is proposed and an expansion of the network is introduced in this paper. In the proposed architecture, each competence of a humanoid is differentiated based on a behavior-based framework and these differentiated competencies are represented by individual agents. In the expansion introduced here, the agents can utilize environmental models generated by itself and other agents as well as information received from physical sensors in determining their own behaviors.

Autonomous mobile robots acting in dynamic environments of every-day life are required to have multi-faceted competence[1]. Humanoids can achieve composite tasks that satisfy multiple purposes using their arms and legs. For instance, they can hold an object with their arms while walking towards a goal with their legs. Another example would be fall avoidance where coordinated motions of whole body are needed. To realize multi-faceted abilities on humanoids, we have developed DBANet based on a behavior-based multi-agent scheme.

In the first stage of developing DBANet, we focused our attention on the scenario that the agents are able to construct explicit internal states and to exploit them in order to cooperate with each other. In the early stages of developing behavior-based frameworks such as Subsumption Architecture (SA)[2], behavior-based systems tend to completely avoid explicit models or representations of their environments or states. This tendency made these systems to miss opportunities of using simple models that could be constructed quickly and be useful in a practical manner. Constructing a simple model does not necessarily sacrifice the advantage of behavior-based systems such as robustness[3]. It is important to model internal states of the agent because of the two reasons. The first reason is that the agents can determine to execute its action or not by referring to internal states of other agents. This allows the proposed DBANet to realize flexible arbitration of the agent's behavior. Second, an observability of parts of behavior control architectures is needed because designers have to debug and evaluate a robot system through internal states in constructing robot behaviors[4]. Designers can easily and explicitly understand behavior of the agents or circumstances of interactions among the agents by observing internal states. Based on this observation, explicit internal states are exploited in the proposed DBANet. Because of the above reasons, internal states should include information on the type of action the agent executes and parts of the body used by the agent. Behavior patterns of the agent are described by a state transition network in a behavior decision part of the agent termed NetIBSI[5]. The type of action is represented as a state of a state transition network. A list of parts of the body corresponding to the type of action is used to arbitrate agent's behaviors in an arbitration part of the agent called Conflict Resolution Strategy (CRS) in the proposed DBANet.

In verification experiments, we have used the fallowing agents to describe behavior patterns of humanoids: Two types of Fall Avoidance agents, a Search agent and an Avoid agent. The DBANet with those agents is installed on to humanoids constructed in our laboratory. Composite behaviors are realized based on cooperation of agents using the internal states as described in [6].

Behaviors of these agents are however inefficient because the agents do not construct explicit environmental models. Naturally, the agents cannot use the environmental models constructed by other agents. In the second stage, we therefore expanded DBANet to include limited modeling of environmental state and sharing them among the

agents. The first expansion allows the agents to generate limited, local and incomplete but explicit environmental models and utilize them to decide their own behaviors. It is natural that these models become limited and local because the robots' competence is divided into the agents based on a behavior-based framework. From this, the agents, especially agents placed in charge of primitive behavior, do not need complete and global environmental models. It is, however, difficult to generate intelligent and efficient behavior only using interactions of primitive agents or behaviors[3]. Therefore, the second expansion makes the agents be able to exploit the environmental models generated by other agents. It has a distinctive advantage that the agents can obtain wider information using other agents' models than only using their own models. Because of modeling the environment and sharing them, the extended agents can act more efficiently and flexibly even though the agents are constructed based on a behavior-based framework. We developed three agents, a Kick Object agent, a Search Object agent and a Search Goal agent using this expanded DBANet. These agents are implemented onto our humanoid to achieve transporting an object to a goal destination by kicking.

2 Distributed Behavior Arbitration Network

DBANet for managing humanoid's behavior is introduced here and its expansion is explained in detail. First, a set of basic policies in developing DBANet architecture is enumerated. Second, the whole structure of DBANet is outlined and then its components are explained.

2.1 A Set of Basic Policies

The first policy is the basic policy of our humanoid study. It is to adopt the concept of behavior-based multi-agent system to construct a behavior managing architecture for humanoids. Thus, the total capability of the robot is differentiated based on the concept of behavior-based framework and these differentiated capabilities are represented by distinct agents. Each agent therefore has competence of executing a well-defined identifiable behavior pattern by itself.

Second, an internal state of each agent is positively modeled within that agent and is made available to other agents. This makes it possible for multiple agents to arbitrate among themselves. This even opens up a possibility of composite behavior due to cooperation of multiple agents. SA is known as the first behavior-based robot system with real-time arbitration capability. SA can quickly arbitrate conflicts in behavior in a dynamic situation. This is because it does not utilize flexible arbitration by exploiting explicit internal state model or other information. This consequently makes arbitration of behavior patterns be necessarily simple and rigid. On the other hand, the internal states in agents in the proposed architecture can make significant contribution in realizing efficient distributed autonomous arbitration among agents.

Third, the proposing architecture must be able to handle a large number of actuators without contradiction. This is because the architecture is for behavior control of a humanoid with many joints on its body including two arms and two legs.

Forth, the controller must be able to realize both quick responsive behavior and a well-planned series of behavior patterns that achieves a given goal. It is noted that these two behavior patterns, although needed for realistic humanoids, have fundamentally different characteristics and are not easily build into a single controller.

A set of basic policies as stated above however describes that the agents exploit only the explicit internal state model of each agent to arbitrate their behavior among themselves. Now, we add new basic policies that focus on using environmental models.

The first added policy is that the agent can construct explicit but limited environmental models and exploit them to select own behaviors. To realize this capability, the agent must be equipped with a function to model the environmental state that is directly influencing its behavior. It is important to limit the model in order not to jeopardize the advantage of behavior-based approach.

Second, the modeled environmental states are made available to other agents. This makes it possible for the agent to exploit an integrated environmental models that is closer to a global environmental model than any individual models of agents. This even makes it possible to realize advanced behavior due to integration of limited information.

A successful controller from our point of view must be designed taking these basic policies into consideration. We follow these policies and propose DBANet as a new concept of an autonomous control architecture for humanoid robots.

2.2 Whole Structure of DBANet

DBANet is composed of many agents that are individually assigned for a single capability of the robot (See Fig. 1). That is to say, distinct behavior patterns of the robot are implemented by individual agents and executed in parallel. Each agent consists of NetIBSI[5], Conflict Resolution Strategy (CRS) and a new component, Environment Modeler (EM) according to the new polices. Those components are explained in detail in the sequel but are briefly introduced here.

NetIBSI is an architecture that integrates behavior and sensation using state transition network and reactive selection of behavior. Here, a state in the state transition network represents an internal state of an agent. CRS clears situations where multiple agents simultaneously demand use of the same bodily resources. EM constructs explicit environmental models and preserves them to provide them for own NetIBSI or other ones. DBANet also includes shared memory for states. The internal states modeled by the agents are preserved in the shared memory. Thus, all agents are able to refer to the information through the shared memory. DBANet does not have a supervisor or an observer that manages the total behavior from an upper level. Thus, the proposed architecture is a completely autonomous distributed multi-agent system.

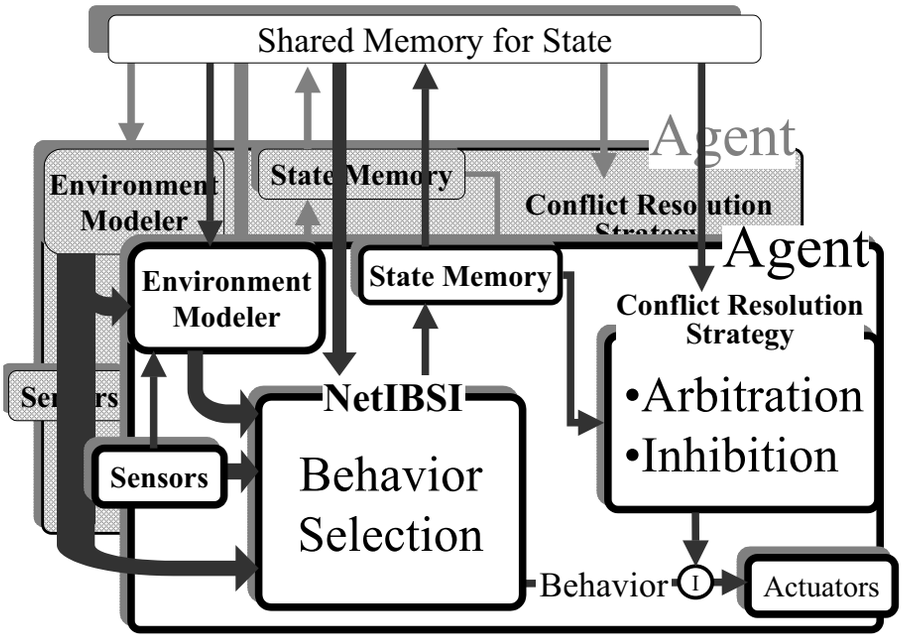


Fig. 1. Distributed Behavior Arbitration Network

2.3 Environment Modeler

EM creates an explicit environmental model. This environmental model corresponds to the agent's task that is identified based on behavior-based framework. Thus, the constructed environmental model consists only of the information that is necessary to achieve the task assigned to that agent. This allows EM to quickly create the model and the agent can generate quick response to a dynamic environment. To create the environmental model, EM does not only obtain information from physical sensors but also the internal state of each agent and the environmental models created by other agents (See Fig. 1). This makes it possible for EM to realize two advantages. The first advantage is that EM can modify the environmental model according to the internal states of agents when no physical sensor information is available to the agent. Second, sharing the environmental models makes it possible for the agent to obtain external information that is otherwise unavailable. Each agent can integrate multiple limited environmental models and is able to exploit more comprehensive model in determining the executions of its behavior.

2.4 NetIBSI

NetIBSI is an architecture that determines behavior of the agent in the proposed architecture. According to the set of basic policies, a robot acting in real environment must be able to respond quickly as well as to systematically execute a series of behaviors to achieve a desired objective. NetIBSI is able to represent both of those behavior patterns and uses states and transitions of states for this purpose. A single behavior pattern is defined by a state termed as Behavior State. A series of behavior patterns for a given intention can therefore be represented by a transition network of those Behavior States. Each Behavior State consists of several Sensor State-Behavior modules (SS-B modules)(See Fig. 2). Sensor information, an internal state of another agent, and/or an environmental state generated by itself or other agents are the inputs to SS-B modules and a behavior pattern corresponding to the input information is the output. An SS-B module becomes active when the input information coincides with the input specification that is described in that module. They are layered from top to bottom according to their priorities in the Behavior State. The behavior pattern of an SS-B module that is active and has the highest priority is chosen by the Behavior State. The agent's state is then transformed from the current Behavior State to the next Behavior State that is specified by the chosen SS-B module.

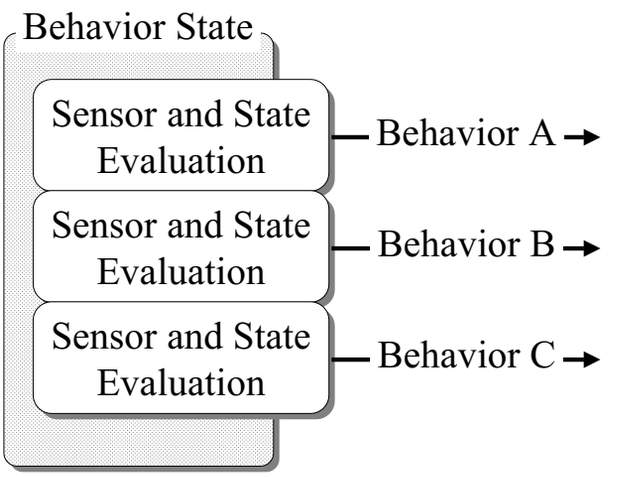


Fig. 2. SS-B modules in Behavior State

A sample schematic diagram of a NetIBSI is shown in Fig. 3 to explain the operation of NetIBSI. It is assumed that this NetIBSI is a part of Agent I. First, it is assumed that Agent I is in Behavior State A. There are three SS-B modules in this Behavior State. SS-B modules are placed from top to bottom in the descending order according to their priorities. Suppose that Sensor 2 has activated and the current internal state of another Agent II is State *a*. Then Agent I executes Behavior Pattern C and the Behavior State of Agent I is transferred to Behavior State C. State of the robot and/or the environment now are observed by SS-B modules in Behavior State C.

The method of representing an intended behavior pattern is the following:

1. The intended behavior is divided into multiple Behavior States based on a behavior transition rule, assuming that an agent executes a series of intended behavior without a failure.
2. SS-B modules that describe conditions for transition to next Behavior States are organized in each Behavior State.
3. SS-B modules that detect failure conditions and specify transitions to yet-specified Behavior States that are responsible for outputting behaviors corresponding to the failure conditions are also organized in each Behavior State.
4. The priorities of SS-B modules are pre-determined according to their importance.
5. The Behavior States for failure behavior are constructed similarly as steps 1 through 4.

A behavior transition rule is designed in such a way that a set of apparent changes in sensory data and/or internal states of other agent triggers the intended transition. For instance, the Behavior State ”walking forward” and ”walking backward” can be divided by the behavior transition rule dependent on the direction of the optical flow.

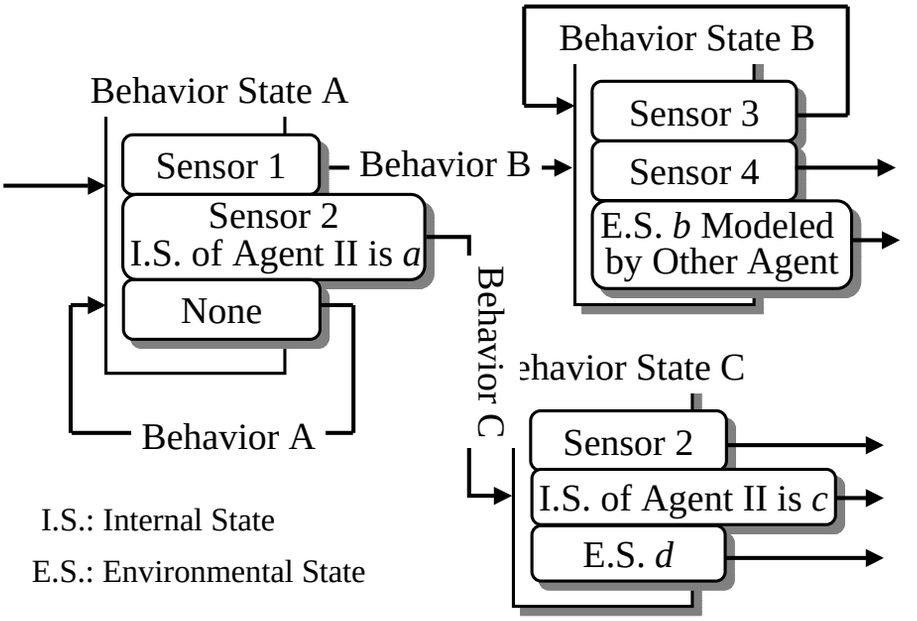


Fig. 3. A Sample of NetIBSI

The internal state of the agent can be represented and formulated by NetIBSI according to the Behavior State. For example, $Condition_{a_k}(BS_{a_k}(j))$ describes whether Agent a_k executes an intended behavior or not and $Use_{a_k}(n, BS_{a_k}(j))$ represents a usage condition of a specific body part that is exploited by Agent a_k. These are formulated depending on $BS_{a_k}(j)$ as follows:

$$Condition_{a_k}(BS_{a_k}(j)) = \begin{cases} 1 & \text{if } BS_{a_k}(j) \text{ is in failure condition.} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

$$Use_{a_k}(n, BS_{a_k}(j)) = \begin{cases} 0 & \text{if Agent } a_k \text{ uses the body port } n \text{ at } BS_{a_k}(j). \\ 1 & \text{otherwise} \end{cases} \tag{2}$$

$$(j = 1, 2, \ldots, l \quad n = 1, 2, \ldots, b)$$

where l represents the total number of Behavior States in Agent a_k, b shows the total number of body parts and n defines a specific body part.

Here, we summarize the features of NetIBSI:

- NetIBSI is able to realize both reactive and planned behaviors.
- The internal state of an agent is represented by a Behavior State in NetIBSI.
- It is easy for designers of the behavior and action of an agent to understand the behavior transition described in NetIBSI by tracing the modeled internal states.
- Multiple sensors can be monitored simultaneously and this makes behavior selection robust.
- Firing conditions for actions can flexibly be specified as the transitions of Behavior States.

2.5 Conflict Resolution Strategy

Conflict Resolution Strategy (CRS) clears conflicts among the agents using priorities. A flow of the strategy is discussed first. Then, a method of determining priorities of agents is explained.

Here, a conflict is defined by the following: Multiple agents demand the use of the same body part(s) at the same time. The conflict occurs because robot competencies are not divided and assigned to agents according to the parts of the robot body but rather by the functionality. Each agent must decide between activating and stopping its behavior when a conflict occurs. Agents use the logic in Fig. 4 for this purpose. First, the agent checks existence of a conflict with other agents. It does this by referring to internal states of other agents from the shared memory. As stated before, the internal states have information on the usage of body parts of all the agents. If conflicts are detected, own priority is compared with priorities of all conflicting agents. The agent with the highest priority

activates own behavior and all other agents become dormant. Then, the priorities of all the agents that are able to execute their behavior are adjusted as explained next.

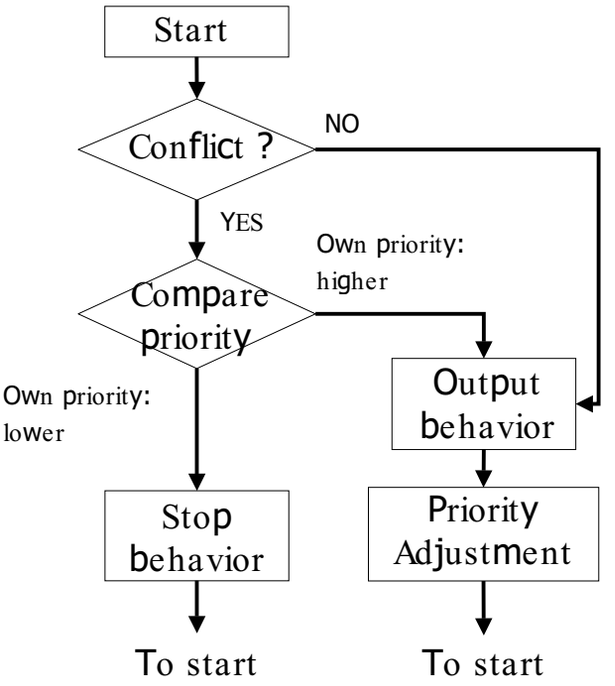


Fig. 4. Conflict Resolution Strategy

The priority is determined based on the following rules.

1. Behavior that is unsuccessful takes precedence. In other words, the agent whose behavior is prevented takes precedence.
2. If the behavior of an agent A cannot be activated unless that of another agent B, then the priority of agent B takes precedence.
3. The agent that uses smaller number of body parts is given precedence when the conflict is not cleared by the above two rules.

The priority is represented by a set of binary digits that is composed of *Fail Flag* bits as higher bits and *Body Flag* bits as lower bits (See Fig. 5). The number represented by the whole bits is the numerical value of the priority. Here, it is considered that the larger the number the higher the priority.

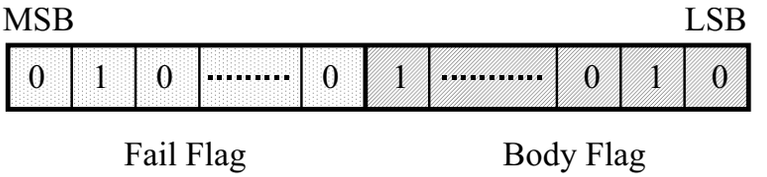


Fig. 5. Format of a Priority Describing Binary Digits

The priority can be formulated using Eq. 1 and Eq. 2. $Priority_{a_k}(BS_{a_k}(j))$ represents the priority in the Behavior State of agnet a_k $BS_{a_k}(j)$.

$$Priority_{a_k}(BS_{a_k}(j)) = Fail\_Flag_{a_k}(BS_{a_k}(j)) + Body\_Flag_{a_k}(BS_{a_k}(j)) \quad (3)$$

$$Fail\_Flag_{a_k}(BS_{a_k}(j)) = 2^{pd_{a_k}+b} \times Condition_{a_k}(BS_{a_k}(j)) \quad (4)$$

$$Body\_Flag_{a_k}(BS_{a_k}(j)) = \sum_{i=1}^{b} 2^{(i-1)} \times Use_{a_k}(i, BS_{a_k}(j)) \quad (5)$$

where a constant pd_{a_k} is decided a priori. More specifically, pd_{a_n} is larger than pd_{a_k} when the precondition is that agent a_n has to be able to achieve the intended behavior in order to execute behavior of agent a_k.

The summary of CRS's advantages is stated below.

- It is possible to realize the conflict resolution strategy that is based on the combination of a priori conditions and the extent of body parts usage, by defining the priority as binary digits as explained above.
- A designer who constructs the behavior of agents can easily and visually confirm the priori conditions and body parts usage conditions from binary digits as shown in Fig. 5.
- The priority can be varied according to the state of the agent that is changed based on the internal and external conditions.
- Conflict resolution is achieved by agents in decentralized manner.

3 Evaluation Experiment

Robots acting in real environment need two behavior types according to the basic policies as stated in Section 2. One is a reaction type that includes simple actions but requires fast response. Another one is a mission type that requires non-trivial planning and often results in a complicated behavior. The mission type behavior can be represented by a composition of simple behavior patterns. The proposed architecture is capable of generating both types of behavior. Two sample cases, one from each behavior type, are adopted to demonstrate this. The first type action is the fall avoidance in standing and dynamic walking. The second behavior type action is the travel-to-goal action where the robot searches a goal and approaches it while avoiding an obstacle. We intend to evaluate capabilities of proposed architecture in integrating the multiple environmental models and behavior of agents from the result of these experiments. Thus, we select a specific action of "the robot transports an object to a goal destination by kicking" for the second type action.

Hardware platforms and software configurations constructed for these experiments are discussed together with these experiments.

3.1 Fall Avoidance

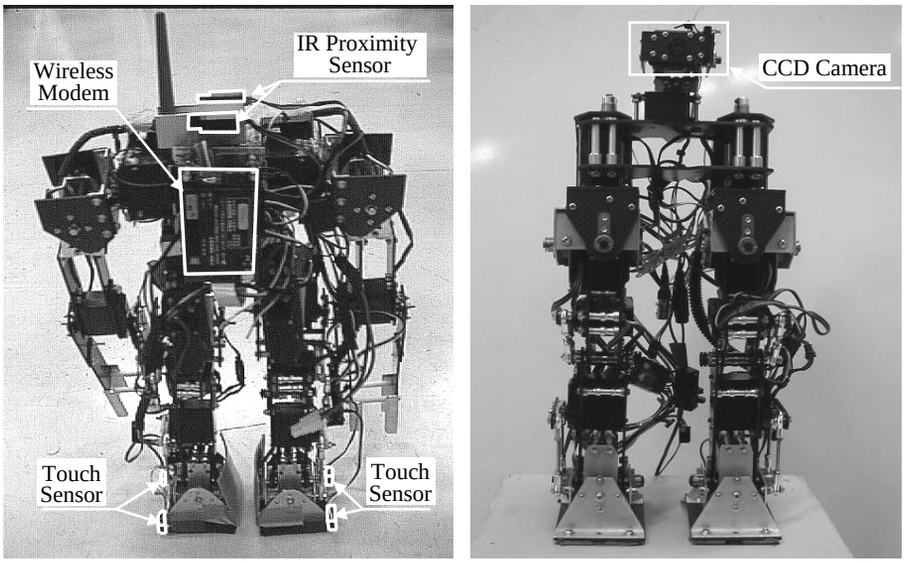
Fall avoidance of humanoids requires fast response and is used to evaluate real-time capability of the controller. It is natural that a fall of a humanoid can be initiated suddenly and the state of the robot can start changing rapidly. Moreover, the robot needs to recover standing straight status via a series of planned motions after a fall is avoided. Therefore, fall avoidance is a good function to evaluate capability of a behavior controller in dealing with rapid phenomena and recovering from them according to planned strategies. It can be confirmed if the controller, that is a single agent whose behavior part is constructed by using NetIBSI here, satisfies the basic policies in Section 2.

We execute two experiments of fall avoidance; *Case 1* and *Case 2*. *Case 1* is that a fall is forcibly initiated by adding force to the upper body of a standing humanoid called Mk.2. Mk.2 avoids the falling by using whole body, particularly its arms. In *Case 2*, customized Mk.3 that can execute dynamic walking stumbles over a step and then the robot avoids the falling by crouching down.

Next, we explain the two robots as the hardware platforms and illustrate the software configuration to realize two types of fall avoidance with these robots and then we discuss the results and their implications.

Hardware Platform (Mk.2 and Mk.3 Robots): A back view of a robot called Mk. 2 for the experiment *Case 1* is shown in Fig. 6(a). Mk.2 robot has two arms and two legs and is 0.4[m] in height and 3.0[kgf] in weight. The robot is equipped with sensors that can detect states of falling down initiation. The following sensors are implemented for this purpose: i) two infrared proximity sensors, ii) a clinometer and iii) four touch sensors. The proximity sensors are located at the upper portion of the front and the back of the robot. The clinometer is installed approximately at the center of the body. The touch sensors are fixed on the back of both feet, two per foot at the front and the back of their outer sides.

Mk.3 robot is customized and used in the *Case 2* experiment. It is used in experiments stated in the next subsection. Figure 6(b) shows the modified Mk.3 robot for the experiment of "Transport an Object to a Goal by Kicking." We explain here the basic specification of Mk.3 robot and then show an equipped sensor in the experiment for *Case*



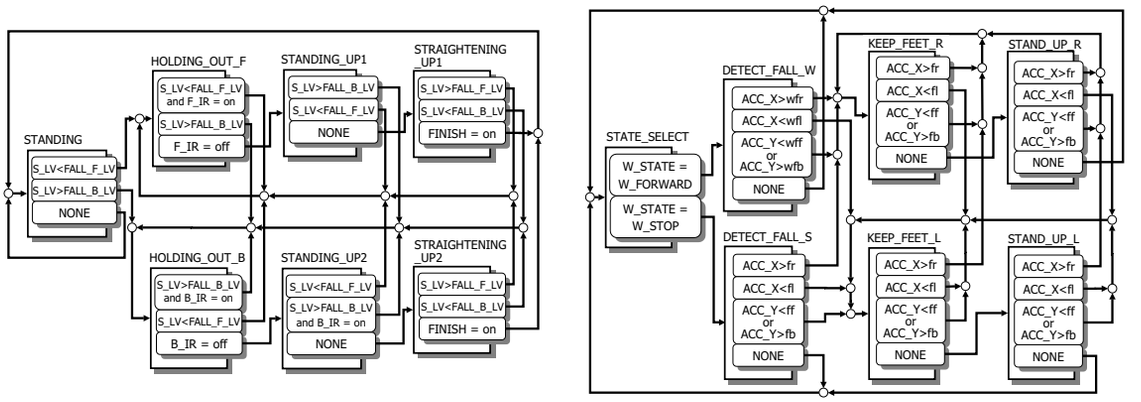
(a) Mk.2 Robot (back view) (b) Mk.3 Robot (front view)

Fig. 6. Mk.2 and Mk.3 Robots

2. Mk.3 robot is 0.30[m] in height and 1.6[kgf] in weight and is able to perform a dynamic biped walking and a dynamic turn action[7][8]. Although Mk.3 can perform dynamic walk with various specifications, it is set to walk at the step length of 50[mm] and the walking speed of 50[mm/s]. It is also possible to dynamically vary the step length and the speed.

In the experiment for *Case 2*, an accelerometer is installed to detect the falling onto Mk.3. This accelerometer can measure two horizontal axes of acceleration, corresponding to front-to-back and left-to-right.

Software Configuration (Two Fall Avoidance Agents): Two NetIBSIs representing fall avoidance in standing (*Case 1*) and in dynamic walking (*Case 2*), which are behavior selection parts of two fall avoidance agents, are shown in Fig. 7.



(a) Fall Avoidance in Standing (Case 1) (b) Fall Avoidance in Dynamic Walking (Case 2)

Fig. 7. Fall Avoidance in Standing (Case 1) and in Dynamic Walking (Case 2) Described by NetIBSI

Behavior States in Fig. 7(a) are listed and explained below:

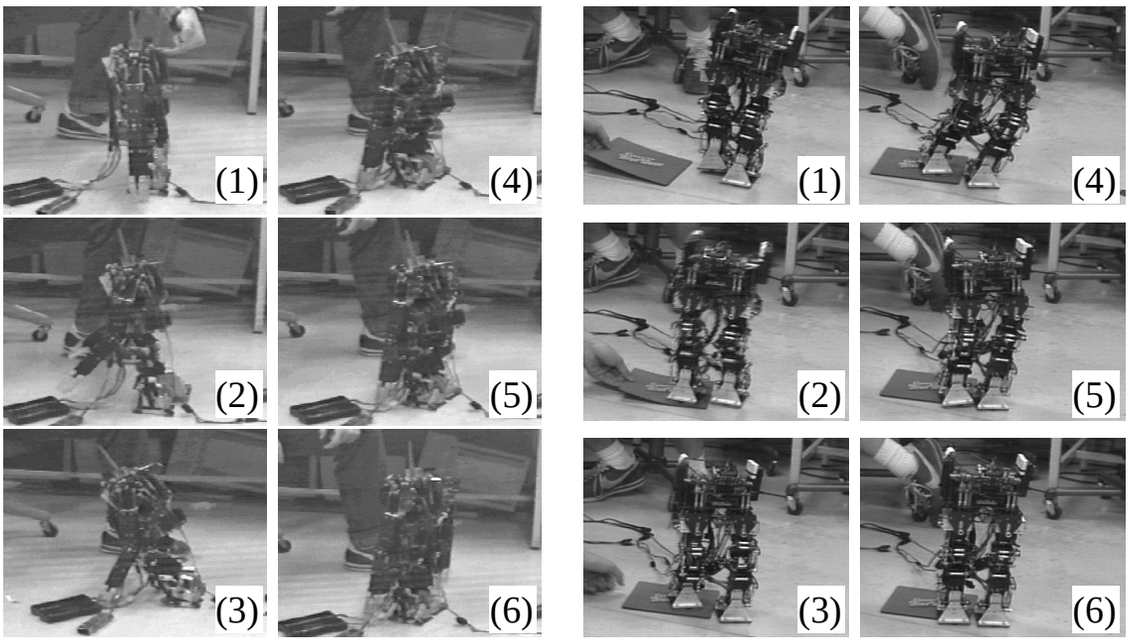
- **STANDING:** The robot is standing. SS-B modules to detect falling down are actively prepared in this state.
- **HOLDING_OUT_F:** The robot holds out its arms forward when a forward falling is sensed.
- **HOLDING_OUT_B:** The robot holds out its arms backward when a backward falling is sensed.

– **STRAIGHTENING_UP1:** The robot straightens up from the crouching position. The arms are retracted to the standard position from forward position.
– **STRAIGHTENING_UP2:** The robot straightens up from the crouching position. The arms are retracted to the standard position from backward position.
– **STANDING_UP1:** The robot changes its position from holding out its arms forward to crouching position. This moves the projection of the center of gravity within convex hull of the robot feet and makes it possible to lift arms.
– **STANDING_UP2:** The robot changes its position from holding out its arms backward to crouching position. This moves the projection of the center of gravity within convex hull of the robot feet and makes it possible to lift its arms.

Behavior States in Fig. 7(b) are listed and explained below:

– **STATE_SELECT:** A state where the walking condition of the robot is observed through the states of other agents. A state that is appropriate for the current walking condition is selected and a state transition to that state occurs. Next Behavior State is selected based on this state.
– **DETECT_FALL_W:** Reaction thresholds of sensors that are appropriate for detecting falling down when the robot is walking are sat in SS-B modules in this state.
– **DETECT_FALL_S:** Reaction thresholds of sensors that are appropriate for detecting falling down when the robot is standing are sat in SS-B modules in this state.
– **KEEP_FEET_R:** The robot keeps feet when falling right is sensed.
– **KEEP_FEET_L:** The robot keeps feet when falling left is sensed.
– **STAND_UP_R:** The robot straightens up from the crouching position in KEEP_FEET_R.
– **STAND_UP_L:** The robot straightens up from the crouching position in KEEP_FEET_L

Results and Implications The procedure of the *Case 1* experiment is simple. Fall is forcibly initiated by adding force to the upper body of a standing Mk.2 and the reaction of the robot is recorded. A typical standing up motion sequence from leaning forward is shown in Fig. 8(a). The procedure of the experiment for *Case 2* is that a fall is forcibly started by placing a step that is 5[mm] in height while Mk.3 robot is walking. A typical sequence of avoiding falling down by crouching is shown in Fig. 8(b). In the *Case 1* experiment, Mk.2 can avoid falls and



(a) Case 1 (b) Case 2

Fig. 8. Actual 'Standing Up' Motion from Leaning Forward and Actual 'Crouching Down' Motion from Stumbling Over a Step

recover its posture as shown in Fig. 8(a). Mk.3 is able to avoid falls by crouching as shown in Fig. 8(b) in the *Case 2* experiment. We can confirm the following facts from the results of two experiments.

- The robots controlled by NetIBSI can rapidly react to avoid falls.
- Appropriate motions to recover the standing position are initiated by checking the own state after avoiding a fall. The robots are able to recover the standing position.
- The behavior of fall avoidance using the whole body of the humanoid can be represented by NetIBSI.

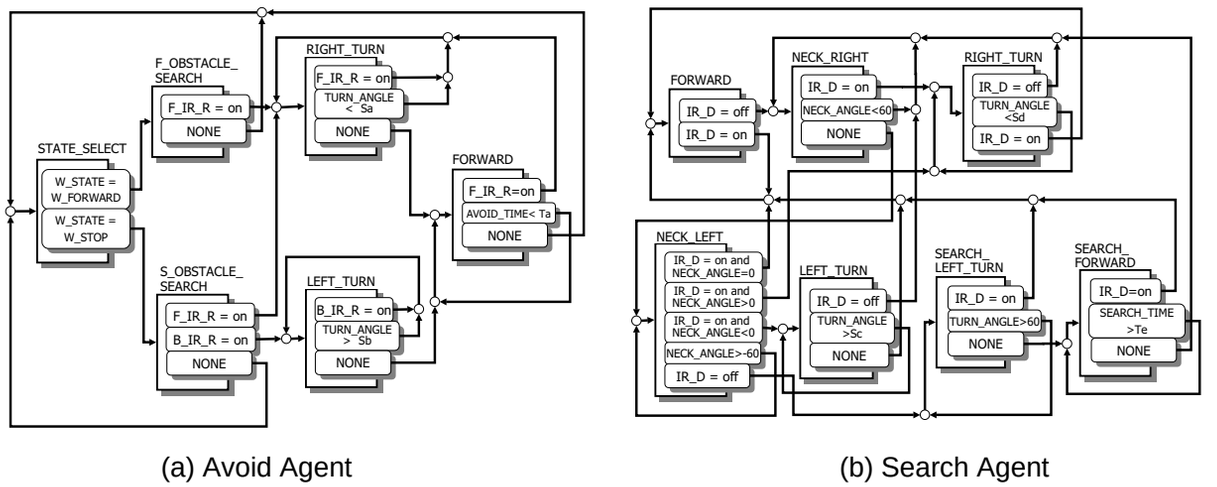
As stated above, it is shown that NetIBSI can realize both real-time capability and planned sequence of behavior patterns.

3.2 Search and Approach a Goal

Conflict resolution capability of DBANet is evaluated in this experiment. As stated before, the situation, ”a robot approaches a prescribed goal while avoiding obstacles,” is selected for this purpose. The robot needs two agents to deal with this situation. The first one is the Search agent that is capable to search a goal and to approach it. Another one is the Avoid agent whose competence is to avoid obstacles. It is expected that if these two agents work as planned, the robot controlled by them would achieve a composite behavior of finding a goal and approaching it while avoiding obstacles. To confirm this experimentally, we alter sensors of Mk.3 robot from those used in 3.1 and represent the two agents using the proposed architecture. Next, we discuss the hardware platform (customized Mk.3) and software configurations of two agents. The results and implications follow.

Hardware Platform (Mk.3 Robot): We customize Mk.3 for this experiment. The following sensors are installed to detect obstacles and a goal: i) one infrared detector and ii) two infrared proximity sensors. The proximity sensors are installed at the top portion and the backside of the body and can detect obstacles in front or rear of the robot. The infrared detector is located on the head of the robot that has one degree of freedom in yaw axis and is capable to sense a goal that emits infrared signals.

Software Configuration (Search Agent and Avoid Agent): Two agents are constructed to realize the composite behavior that consists of searching and approaching a goal while avoiding obstacles. Behavior patterns of the Search agent and the Avoid agent, represented by NetIBSI, are shown in Fig. 9.



(a) Avoid Agent (b) Search Agent

Fig. 9. Behavior Part of the Avoid Agent and the Search Agent Described by NetIBSI

Behavior States of the Avoid agent symbolized in Fig. 9(a) are defined as follows:

- **STATE_SELECT:** A state where the walking condition of the robot is observed through the states of other agents. A state that is appropriate for the current walking condition is selected and a state transition to that state occurs. Next Behavior State is selected based on this state.

- **F_OBSTACLE_SEARCH:** The robot watches obstacles in front while walking forward.
- **S_OBSTACLE_SEARCH:** Obstacles in front and rear are monitored while the robot is standing.
- **RIGHT_TURN:** The robot turns right to avoid obstacles that are detected in front.
- **LEFT_TURN:** The robot turns left to avoid obstacles that are detected behind.
- **FORWARD:** The robot walks forward for a fixed time duration after avoiding obstacles.

Here, STATE_SELECT, F_OBSTACLE_SEARCH and S_OBSTACLE_SEARCH are defined to be successful states and the rests are unsuccessful states. This is based on the claim that the robot can perform its intended task of approaching the goal when no obstacles are detected.

Behavior states of the Search agent in Fig. 9(b) are explained below:

- **FORWARD:** The robot moves towards the detected goal in this Behavior State. This Behavior State is a successful state because the robot is approaching the goal when it is active. All other Behavior States in this agent are unsuccessful states.
- **NECK_RIGHT:** The neck turns right to seek the goal.
- **RIGHT_TURN:** The robot turns right because of detecting the goal while rotating the neck. The turn angle is the same as the rotate angle of neck in NECK_RIGHT when the goal is found. The neck rotates counterclockwise while the robot turns in order to keep the goal in sight. A direction of the robot body faces the goal after completing this Behavior State.
- **NECK_LEFT:** The neck turns left in this Behavior State.
- **LEFT_TURN:** Similarly to RIGHT_TURN, the robot turns left to face the goal.
- **SEARCH_LEFT_TURN:** The robot turns left to seek a goal. This is activated when a neck turn is not successful in finding the goal.
- **SEARCH_FORWARD:** The robot walks forward to move to a new position for goal search.

As mentioned above, FORWARD is the only successful state and all other states are unsuccessful ones for the Search agent. This distinction is important in deciding the priority of this agent using CRS.

The priorities, which are represented by binary digits, of the Avoid and the Search agents are established based on the rules of Section 2. The specifics are shown in Table 1. *Success* denotes that the agent is in one of the successful states and *Failure* denotes otherwise. The placement of *Fail Flag* bits for agents depends on criticality of the behavior patterns of agents. For example, the Search agent can make the robot approach a goal provided when obstacles are avoided. It therefore cannot execute its behavior while the Avoid agent is failing. Then, *Fail Flag* of the Avoid agent is necessarily placed in a higher position than that of the Search agent. This makes the priority of the Avoid agent become higher than that of the Search agent. The bits in *Body Flag* digits signify the right leg, the left leg, the right arm, the left arm and the neck in this order from higher bits.

Table 1. Assignment of Binary Digits to Behavior and State

Agent	State	Fail Flag	Body Flag
Search	Success	0 0	0 0 1 1 0
	Failure	0 1	0 0 1 1 0
Avoid	Success	0 0	1 1 1 1 1
	Failure	1 0	0 0 1 1 1

Results and Implications The robot is able to achieve its objective of approaching the goal while avoiding the obstacle as shown in Fig. 10. In this figure, numbers denote the order of executing behavior and circles represent the goal point (a source of infrared light). A typical history of the priorities and behavior activation of two agents are shown in Fig. 11.

The priority of the Avoid agent becomes high at five seconds mark in Fig. 11(a) while the robot is moving forward because the robot has sensed an obstacle (See Fig. 10(2)). Since the obstacle is found but not avoided yet, the priority of the Avoid agent is high and the obstacle avoidance behavior is initiated by this agent. The priority of the Avoid agent is high until the robot has avoided the obstacle (See Fig. 10(4)). The Search agent conflicts with the Avoid agent because both agents demand the legs simultaneously while the robot detects the obstacle (from five seconds to thirty seconds). The Search agent stops executing its own behavior in this period by itself (See Fig. 11(b)) because the own priority is lower than that of the Avoid agent (See Fig. 11(a)). After thirty seconds, both agents execute their behavior simultaneously because there is no conflict between the agents. Thus, the arbitration between the agents succeeds and composite behavior is realized by the proposed architecture.

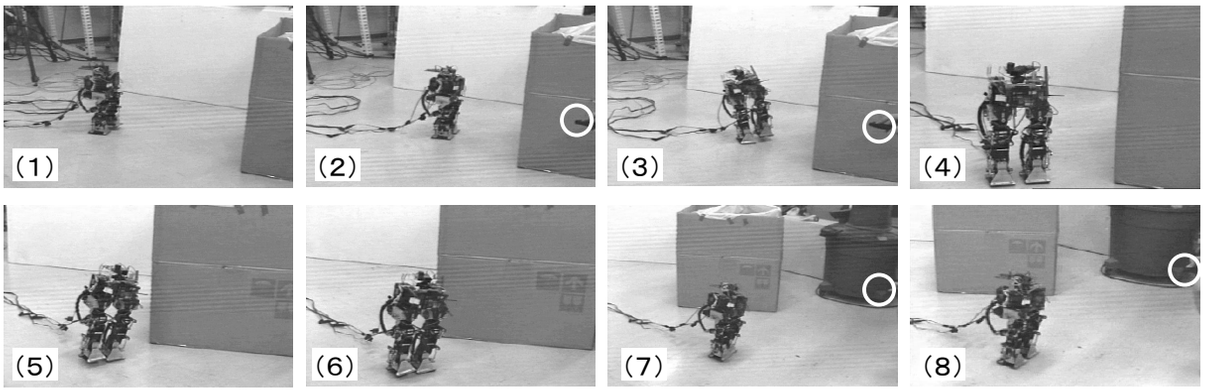
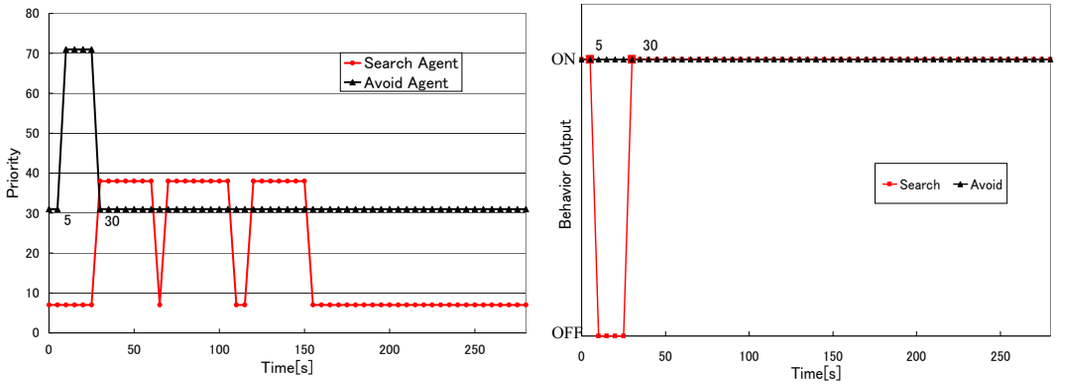


Fig. 10. The robot approaches the goal while avoiding the obstacle.



(a) Priority History (b) Behavior Activation History

Fig. 11. Priority History and Behavior Activation History

3.3 Transport an Object to a Goal by Kicking

Capability in integrating the multiple environmental models of the proposed architecture is evaluated in this experiment. We select the following situation for this experiment: a robot searches an object and a goal destination and transports an object to a goal by kicking. The robot needs three agents to handle this situation. The first agent, called the Search Object agent, has the ability to search an object and create a geometric model (a map) that shows the location of the object. Another search agent termed as the Search Goal agent is capable to seek a goal and represent the location of a goal in the map. The last agent named the Kick Object agent can make a robot move near an object and transport an object by kicking using the modeled information. We can check the capability in modeling the environmental state and behavior of a robot as a result of modeling environmental states and sharing them in this situation. Mk.3 is modified again and used for this experiment. The three agents are constructed using the proposed architecture. We discuss the hardware platform (the modified Mk.3) and software configurations of three agents and then explain the results and the implications of the experiment.

Hardware Platform (Mk.3 robot): Figure 6(b) shows the Mk.3 robot modified for this experiment. We add two new joints, pan and tilt, on top of the robot as a pedestal of a CCD camera. The CCD camera is used for finding an object and a goal. The robot can measure a distance between the robot and the goal or the object by exploiting a slope of the tilt joint and the height of the robot with one CCD camera[9].

Software Configuration (Search Goal Agent, Search Object Agent and Kick Object Agent): Three agents are constructed to realize the composite behavior that consists of searching the goal and the object and transporting the object to the goal by kicking. Behavior patterns of the Search Goal agent, the Search Object agent and the Kick Object agent, represented by NetIBSI, are shown in Fig.12.

Behavior states of the Search Goal agent in Fig.12(a) are explained below:

– **STOP_GOAL:** The robot is detecting the goal in this Behavior State. This Behavior State is kept for a fixed time once the robot finds the goal. This Behavior State is a successful state because the propose of this agent,

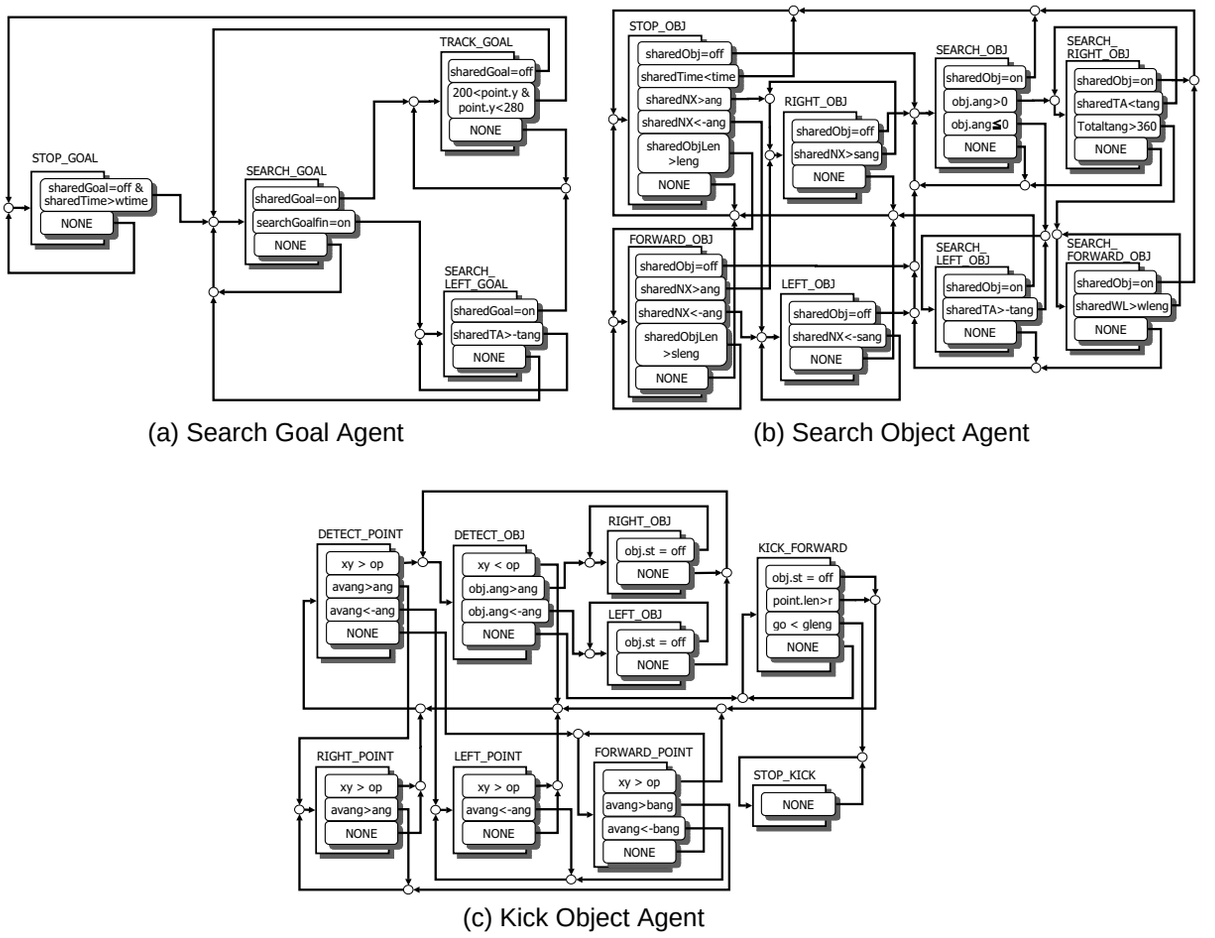


Fig. 12. Behavior Part of the Search Goal Agent, the Search Object Agent and the Kick Object Agent Described by NetIBSI

that is to detect the goal and model it, is achieved. All other Behavior States in this agent are unsuccessful states.

- **SEARCH_GOAL:** The robot turns the neck right and left to seek the goal.
- **SEARCH_LEFT_GOAL:** The robot turns left to detect the goal because the goal is not detected in SEARCH_GOAL.
- **TRACK_GOAL:** The robot tracks the goal using the neck joints to sight it in the center of view in this Behavior State.

As mentioned above, STOP_GOAL is the only successful state and all other states are unsuccessful ones for the Search Goal agent.

Behavior states of the Search Object agent in Fig.12(b) are explained below:

- **STOP_OBJ:** The robot stops and tracks the object using the neck joints to sight it in the center of view in this Behavior State. The robot detects the object in this Behavior State. This Behavior State is a successful state because the propose of this agent, that is to bring the object into sight, to approach the object and to model it, is achieved. All other Behavior States in this agent are unsuccessful states.
- **FORWARD_OBJ:** The robot walks forward while tracking the object detected in the front until a distance between the robot and the object becomes a fixed length.
- **RIGHT_OBJ:** The robot turns right while tracking the object detected in the right until the object is brought into the center of view.
- **LEFT_OBJ:** The robot turns left while tracking the object detected in the left until the object is brought into the center of view.
- **SEARCH_OBJ:** The robot turns the neck to the right and the to left to seek the object.
- **SEARCH_RIGHT_OBJ:** The robot turns right while tracking the detected object. The turn angle is the same as the rotate angle of neck in SEARCH_OBJ when the object is found. A direction of the robot body faces the goal after completing this Behavior State.

– **SEARCH_LEFT_OBJ:** The robot turns left to face the goal while tracking the object detected in left side.
– **SEARCH_FORWARD_OBJ:** The robot walks forward to move to a new position for object search.

As stated above, STOP_OBJ is the only successful state and all other states are unsuccessful ones for the Search Object agent.

The above two search agents construct maps that include the position of the object or the goal as the environmental model. The origin of the maps is the robot position and the position of the object or the goal is represented as a relative coordinate of the robot. These agents are able to modify the own map by using dead reckoning technique even if the object or the goal is not on a visual field. For instance, the agents make position of the object or the goal rotate left in each map when the robot turns right and the targets are out of view. Here, the map constructed by the Search Object agent is called object map and the map modeled by the Search Goal agent is named goal map.

Behavior states of the Kick Object agent in Fig.12(c) are explained below:

– **DETECT_POINT:** The position of the robot where the robot can kick the object to transport to the goal is determined using both the object and the goal map information.
– **FORWARD_POINT:** The robot walks to the kick point mentioned above.
– **RIGHT_POINT:** The robot turns right to face the object at the kick point.
– **LEFT_POINT:** The robot turns left to face the object at the kick point.
– **DETECT_OBJ:** The direction of the object in the object map is checked to face the object.
– **RIGHT_OBJ:** The robot turns right because the object map shows that the object is on the right side.
– **LEFT_OBJ:** The robot turns left because the object map shows that the object is on the left side.
– **KICK_FORWARD:** The robot walks forward and kicks the object to transport the object to the goal.
– **STOP_KICK:** The robot stops its motion when the mission of transporting the object to the goal by kicking is completed.

Here, KICK_FORWARD and STOP_KICK are defined to be successful states and the rests are unsuccessful states.

The priorities of the Search Object, the Search Goal and the Kick Object agents are established based on the rules of Section 2. These are shown in Table 2. The description of Table 2 is the same as Table 1.

Table 2. Assignment of Binary Digits to Agents and States

Agent	State	Fail Flag	Body Flag
Search Goal	Success	0 0	1 1 1 1 1
	Failure	1 0	0 0 1 1 0
Search Object	Success	0 0	1 1 1 1 0
	Failure	1 0	0 0 1 1 0
Kick Object	Success	0 0	0 0 1 1 1
	Failure	0 1	0 0 1 1 1

Results and Implications A sequence of scenes of a typical experiment is shown in Fig. 13. Figure 14 shows views seen by the robot and the generated map superimposed on it. In these figures, numbers denote the order of executing behavior. Initial positions of the robot, the object and the goal are shown in Fig. 13(1). The object is established as a blue ball and the goal is a red ball. In this experiment, the goal position is altered when the goal is out of the view field (See Fig. 13(6) to (8)) to evaluate the capability in modifying the map of the agents. Previous places of the goal are represented as a black ellipse pointed by a white thick arrow in Fig. 13(8) to (12) and Fig. 14(4), (6). Black double circles in Fig. 14 represent the map field generated by agents. The origin of the circles is the location of the robot. An inside black circle denotes 0.5[m] from the robot and an outside black circle is 1.0[m] from the origin. A white small circle shows the position of the goal and a black small square describes the position of the object in the map. Each position is individually placed on the view screen by each search agent using over-lay memory technique.

The robot moves the neck joints to find the goal (See Fig. 13(2)) and the Search Goal agent adds the goal point on the goal map (See Fig. 14(1)). Next, the robot seeks the object (See Fig. 13(3)) and the object point is added by the Search Object agent on the object map (See Fig. 14(2)). The Kick Object agent calculates the kick point by using the object and the goal point that are modeled by two search agents and makes the robot move the kick point while the robot is tracking the object (See Fig. 13(4) to (6) and Fig. 14(3)). The Search Object and the Search Goal

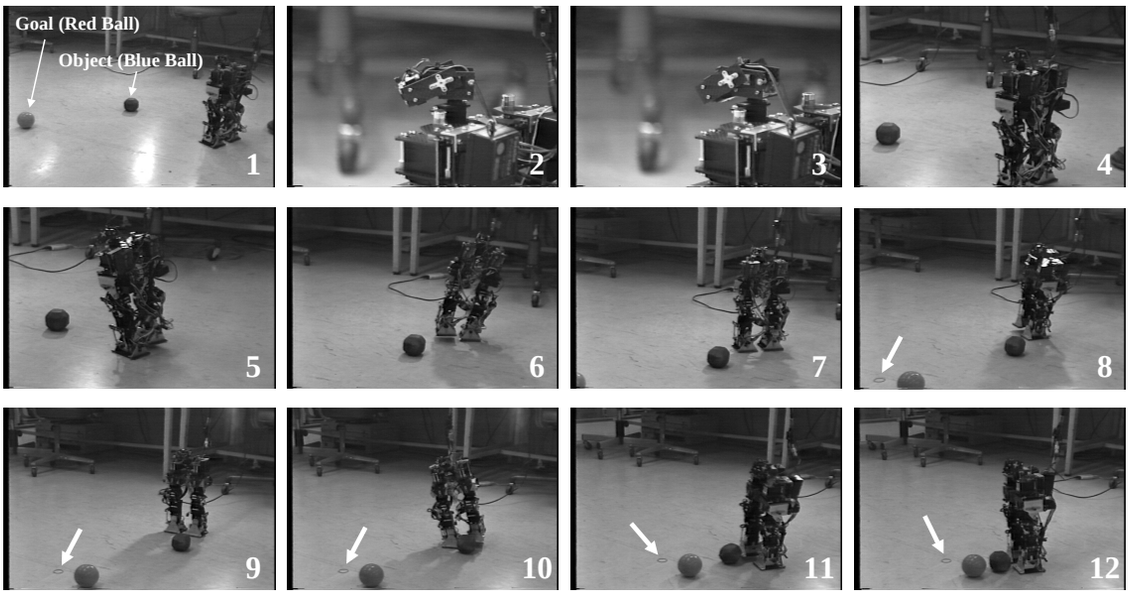


Fig. 13. The robot approaches the goal while avoiding the obstacle. (Views of experimental field)

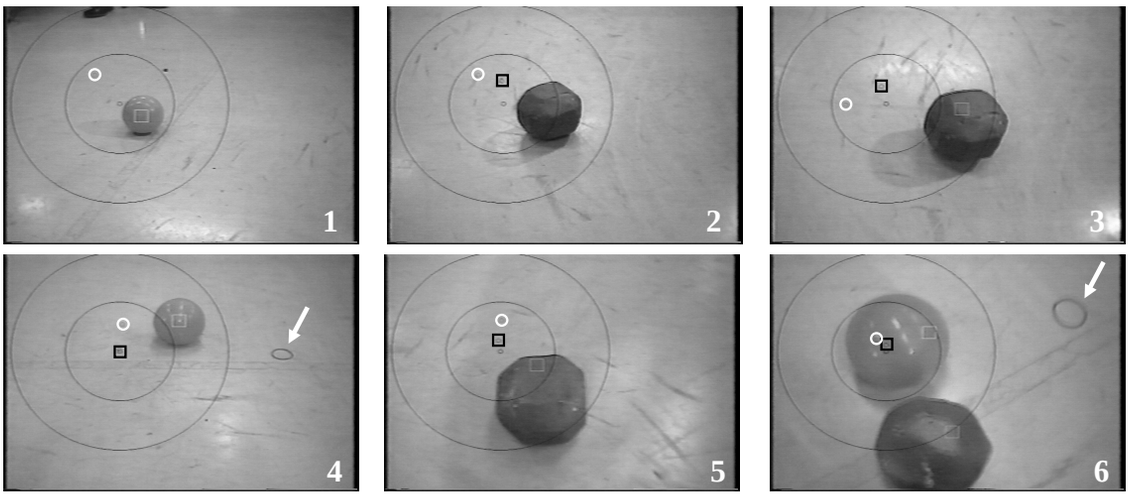


Fig. 14. The robot approaches the goal while avoiding the obstacle. (Views from robot eye)

agents turn to the failure state in turn while the robot is moving to the kick point because the robot can no longer observe the goal and the object due to the limit of a excursion of its neck joints (See Fig. 13(6) to (7)). First, the Search Object agent, therefore, makes the robot turn left to seek the object because the object is supposed to be on the left side according to the object map (See Fig. 13(8)). Next, the Search Goal agents activates behavior of searching the goal and renews the potion of the goal on the goal map (See Fig. 13(9) and Fig. 14(4)). The Kick Object agent confirms that the robot is on the kick point from the renewed object and goal maps and then makes the robot start kicking the object (See Fig. 13(10)(11) and Fig. 14(5)). Finally, the robot finishes transporting the object to the goal (See Fig. 13(12) and Fig. 14(6)). Thus, integration of the local and limited models succeeds and efficient composite behavior is realized by the proposed architecture.

4 Concluding Remarks

In this study, the DBANet and its expansion are proposed as a new concept for generating composite behavior of humanoids. It is based on the concept of multi-agent behavior-based flamework. The proposed architecture is implemented onto our humanoids to perform fall avoidance, goal search and approach, and transporting an object to a goal by kicking behaviors. Experiments are successfully accomplished. This proves that the intended capability of the proposed control architecture is practically realizable.

References

1. Toshihiro Matsui. How humanoid robots come closer to humans. *Journal of the Robotics Society of Japan*, vol. 15, no. 7, pp. 961–963, 1997.
2. Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, vol. RA-2, no. 1, pp. 14–23, 1986.
3. Yoshinori Kuno. Behaviors of behavior-based robots. *Journal of the Robotics Society of Japan*, vol. 11, no. 8, pp. 1178–1184, 1993.
4. Shooji Suzuki and Shin'ichi Yuta. A description and a programming method for the sensor based behavior of the autonomous mobile robot. *Journal of the Robotics Society of Japan*, vol. 10, no. 2, pp. 254–265, 1992.
5. Masaharu Shimizu, Takayuki Furuta, and Ken Tomiyama. Behavior/sensory architecture based on NetIBSI for autonomous mobile robot. *Proc. of the 16th Annual Conferece of the Robotics Society of Japan*, vol. 1, pp. 335–336, 1998.
6. Masaharu Shimizu, Takayuki Furuta, and Ken Tomiyama. Distributed behavior arbitration network: An autonomous control architecture for humanoid robots. *Proc. of International Symposium on Micromechatronics and Human Science*, pp. 267–274, 1999.
7. Takayuki Furuta, Hideaki Yamato, and Ken Tomiyama. Biped walking using five-link virtual inverted pendulum model. *Proc. of 3rd Int. Conf. on Advanced Mechatronics*, pp. 614–619, 1998.
8. Takayuki Furuta, Hideaki Yamato, and Ken Tomiyama. Biped walking using multiple link virtual inverted pendulum model. *Journal of Robotics and Mechatronics*, vol. 11, no. 4, pp. 304–309, 2000.
9. Hitoshi Fukui, Kazuya Noguchi, and Takayuki Furuta. Realtime active vision for mobile robot with a high-speed correlator. *Proc. of the 74th JSME Spring Annual Meeting*, vol. 1, pp. 546–547, 1997.