Developmental Realization of Whole-Body Humanoid Behaviors Based on StateNet Architecture Containing Error Recovery Functions

Fumio Kanehiro[1], Masayuki Inaba[2], Hirochika Inoue[2], and Shigeoki Hirai[1]

[1] Electrotechnical Laboratory, AIST, MITI, 1-1-4 Umezono, Tukuba, Ibaraki, 305-8568, JAPAN
{kanehiro, hirai}@etl.go.jp
[2] The University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8656, JAPAN
{inaba, inoue}@jsk.t.u-tokyo.ac.jp

Abstract. In this paper, a basic architecture is proposed for a developmental realization of whole-body humanoid behaviors. This architecture is called “StateNet” and it contains error recovery functions. Humanoid has a body structure that is similar to human and coexists with human in the environment prepared for human. As a result it is expected that humanoids can be used in broad tasks. However, on the other hand, many functions must be realized in order to draw out potential abilities and apply humanoids to them. In order to use humanoids in those applications, following characteristics are required. (1)Functions can be added incrementally on demands. (2)Humanoids must be able to continue to work by recovering autonomously even if an error occurs while working in the uncertain real world. A proposed architecture contains these characteristics.

1 Introduction

Humanoid is a kind of human friendly robots[oHFR98] which has a body structure similar to human and works in the environment prepared for human. It is expected that humanoids can be applied to broad tasks such as maintenance tasks of industrial plants, home services, disaster rescue and construction and entertainment[INO98]. However, a great effort is required in order to apply humanoids to various applications. A strong base architecture is indispensable to reduce it. This architecture is called “StateNet” and it has following two features.

The first feature is a easiness of incremental extensions and integrations. So many functions and their integration are required for humanoid applications. And it is impossible to make a humanoid which has all functions required from the beginning. In order to realize them, many developers must cooperate and it takes for a long period. Consequently, an appropriate development method for humanoids must be consists of two steps. In the first step, a base architecture which enables to upgrade humanoid incrementally is prepared. In the second step, required functions are piled up step by step based on the architecture. That architecture must provide functions which make it easy to add new functions and integrate sub-systems that are developed individually.

The second feature is an autonomous error recovery function. It is an indispensable ability for the robot in the uncertain real world to select an appropriate action, execute it and then continue to work even if a certain error occurs. The more the robot has various actions, the more variety of errors increases. As a result, it is impossible for a developer to predict all errors and add error recovery routines for each error. Therefore, in order to realize this ability, a mechanism is required that connects elemental actions based on a causal relationship between them and generates an action sequence for recovering to a normal state when an error is detected.

The humanoid works in a complex and uncertain environment where humanoids and human coexist. In addition, the humanoid robot itself moves most of its working time in an unstable posture like a standing on its foot. Because of these reasons, a possibility of an error occurrence, for instance falling down, must be very high. Most of researches on humanoid focused on an improvement of robustness and adaptability of actions such as walking. But it is impossible to prevent an error occurrence completely. As a result, it is very difficult for human and humanoid to coexist if humanoid doesn’t have an autonomous error recovery ability.

In a StateNet architecture, the action space is represented as a state transition graph which exists in a state space which is defined by sensory information. Using this representation, extensions by adding new behaviors and integrations of sets of behaviors can be done by simple graph operations. And a robot is able to detect an error occurrence, search an error recovery path from the space and recover from it by itself. In this case, it is not necessary for a developer to write an error recovery method explicitly. Only by describing a essential process of a target action in an upper level, the robot can continue to work even if an error occurs. Because an error recovery function is embedded in a lower level.

In following sections, an outline of a developmental realization and details of a StateNet architecture is explained. And finally, an action space of a remote-brained humanoid robot is represented and a validity of this scheme is confirmed by an experiment. In this experiment, the humanoid falls down while shifting to a walking action, and then it stands up automatically and start walking.

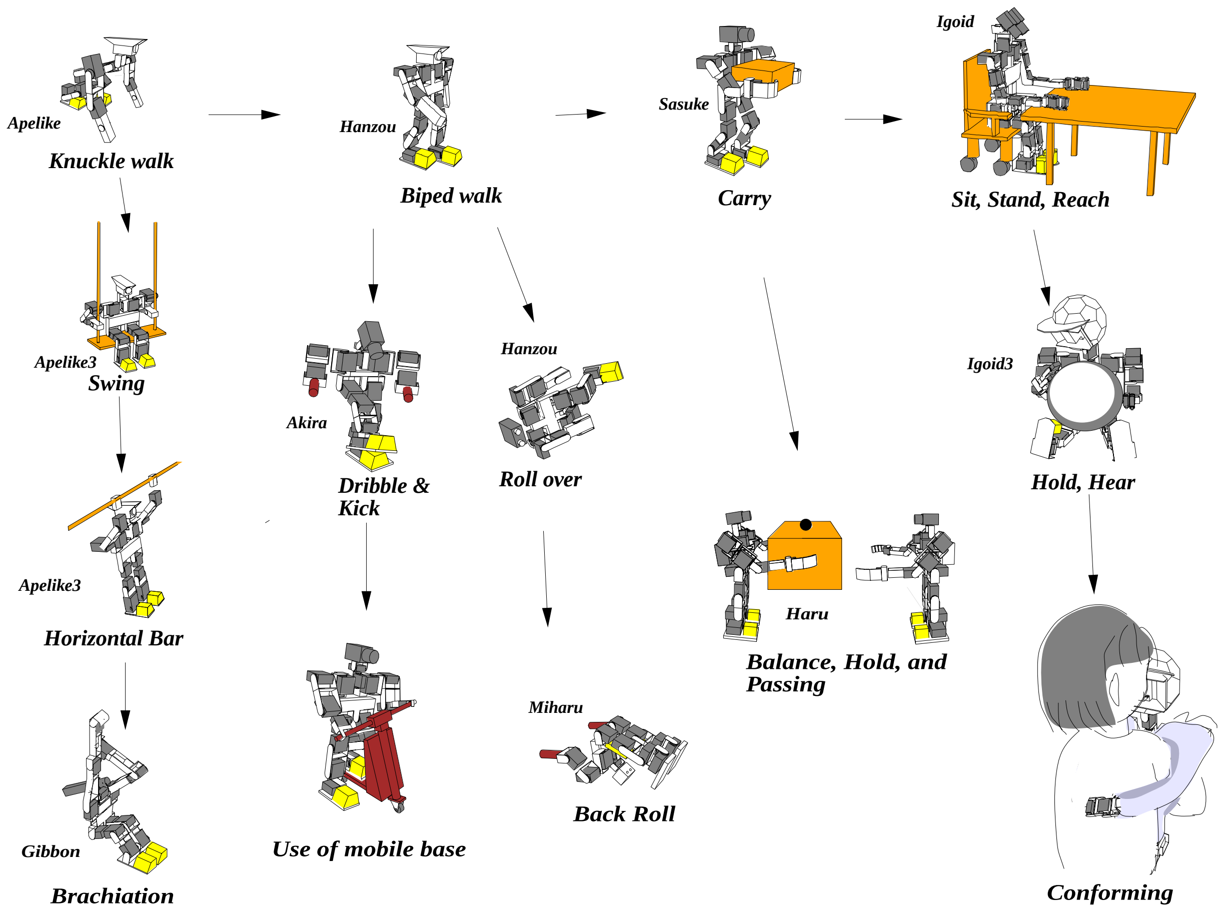


Fig. 1. Developmental Process of Humanoid Robots

2 Developmental Realization of Whole-Body Humanoid Behaviors

2.1 Developmental Realization

Based on a developmental methodology, several behaviors are realized on different hardwares shown in **Fig.1** [HII98,KKT^{+}00,NKII97,IKKI95,KII96,IKK^{+}96,IIKI96,INH^{+}97]. In developmental realization method[KTII99], generational changes in both hardware and software are repeated and many developmental stages are passed step by step as human grows up taking a long period. While passing these stages, each functions are brushed up and variety of behaviors is enriched. Humanoid is one of embodiments of human friendly robots and it can be utilised in various scenes in a daily life. Because of this, humanoids require a developmental method like this. Different systems are developed for each functions so far. But it is difficult to inherit or integrate those systems. Following three are regarded as a developmental elements.

2.2 Skill Development

This development is equal to improving robustness, adaptability, energy efficiency and so on of an action unit. For instance, improving a walking speed and widening area where the robot can walk is a kind of this development for a biped walking. A development which decrease a degree of dependence on developers, for example, using a learning algorithm, is also included in this development even if result behaviors are the same. Most of researches in robotics aim at this skill development.

In order to make it possible to inherit and recycle software when the body hardware is remodelled, an uniform robot representation which is independent on its body is required. An upper level software should access its hardware body through it. If there is not representation like this, overheads for catching up to the previous generation increases for each generational changes and it becomes difficult to work on issues in the new generation.

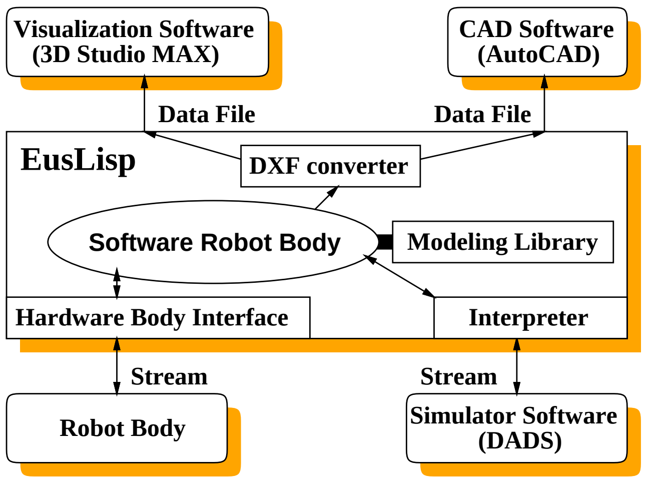


Fig. 2. Structure of Support Environment

Considering with this representation centered, designing a body is regarded as describing it in this representation form, examining performance and an algorithm is regarded as moving this representation in a simulated world, moving a hardware body is equivalent to generating motor commands through this representation. Namely, this representation is none other than the robot model. This model contains all information concerning the robot body such as specifications of actuators, the structure model, shapes and colors. They are used for a conversion of joint angles into motor commands, for simulation and for a body design.

In this research, a model description library is developed which is written in EusLisp[MI90]. EusLisp is an object-oriented Lisp containing a geometric modeller. This library includes four information levels, (1)Data level(convert a motion command to actuator signals, maintain an internal state of the robot and so on), (2)Structure level(simulate motions which don't need a shape), (3)Convex hull level(simulate motions on the flat ground) and (4)Shape level(simulate motions includes more complex contacts). Using this library, robot bodies in each generations are modelled and can be handled in the same way. Placing this model at the core, a software environment shown in **Fig.2** is developed which cooperate with external softwares for CAD and a simulation. This environment shortens a developmental cycle and quicken a development of behaviors. This environment is used for design examinations of robot bodies and acquisitions of behaviors by learning in the simulator world, such as walking or standing up[KII99].

2.3 Phyisical Development

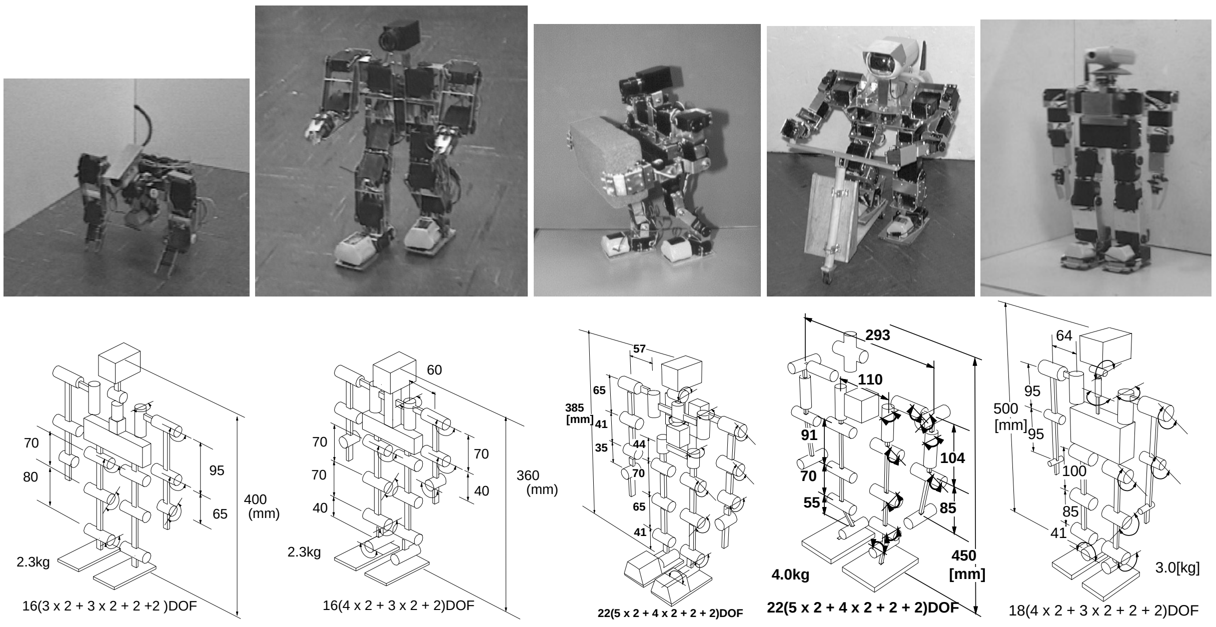
A development of movement function is done by replacing actuators, remodelling structures of joints and rearranging them. This development improves speeds of motions and extends a workable area. A sensory development is done by an addition of sensors or a performance improvement of sensor elements. These developments affect skill and functional developments. In order to realized physical developments like this, the environment is required that enables to build up different bodies in a short period.

In order to change a body structure in a short period like building blocks, bodies should be separated into modules. In developmental stage, an easiness of handling the robot body precede, small size bodies are constructed. They are build as chains of rotational joint modules which is actuated by a servo module for R/C model toys. Humanoid robots in **Fig.3** are developed based on this method.

A information transmission system which corresponds to a nervous system of human is required to make it easy to attach, detach and replace sensors and actuators. They can be done by only modifying local wiring in the case a network which connects sensors and actuators is spread on the whole body. Based on this idea, a onbody sensor-motor network shown in **Fig.4** is equipped on two humanoid robots in **Fig.3** from the right side[KMK$^+$98].

2.4 Functional Development

Various behaviors of humanoids can be classified in several categories, such as locomotion and manipulation. In each categories, several behaviors which are used in different situations belong to. A functional development means



Vision	○	○	○	○	○
Inclinometer		○	○	○	○
Foot Sensor				○	○
Joint Torque				○	○
Touch Sensor					○
Onbody LAN				○	○

Fig. 3. Developed Humanoid Robots, Apelike, Hanzou, Sasuke, Haru and Saizo

to increase the number of categories and the number of behaviors in each categories. For a functional development a description scheme is a key point. That accumulates various actions in a structured form and draws out them on demands. A StateNet architecture is a framework which promote this development.

3 Action Representations of Robots

3.1 Nested Conditional Flow Representation

The simplest way of describing robot actions must be designing a conditions for switching actions using sensory informations and mapping actions to corresponding conditions as shown in left of **Fig.5**. This scheme has an advantage that a representation can be implemented easily and in a short time. But this advantage is valid only while the number of actions is small and an error occurrence is ignored and reset manually. However, when the variety of actions increases or an error handlers are implemented, it becomes difficult for developers to understand a system behavior and extend the system.

3.2 Application Dependent State Transition Graph Representation

In order to integrate various actions in a simple way, a representation scheme is required that enables to add a new action only by a local modification without understanding whole system. A popular scheme for settling this problem must be an utilisation of a state transition. In this scheme, a state of the system is described by several variables and possible combinations of those values are defined as named states. Transitions are invoked by internal and external events and the system is represented as a finite state automaton. In most of representations like this, nodes correspond to elemental actions and arcs correspond to conditions that switch actions as shown in right of **Fig.5**. This representation scheme is equal to transforming a tree structure mentioned above into a graph structure. And using this state transition graph, it is possible to decrease operations when adding a new action and to make it easy to predict a system behavior. But most of representations like this require that a developer must decide conditions of a transition occurrence and partitioning of a state space explicitly. And a structure of the graph

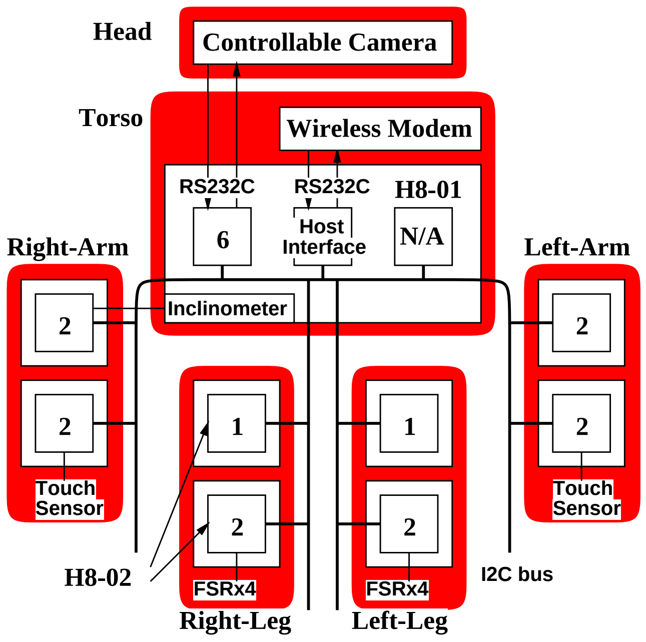


Fig. 4. Structure of Onbody Network

depends on respective applications. As a result, it is difficult to recycle and integrate graphs designed for different applications.

3.3 StateNet

In proposed representation scheme, a state transition is also used. However, the state space is defined clearly as a space which is defined by sensory information. Defining a space like this, an automatic node generation function and a quantitative evaluation function of a relation between states are available. These functions are essential for an automatic error recovery as mentioned later. There is some similar action representation schemes. CAAD[MUK$^+$97](Creating Actuator Angle Data) is used for the humanoid robot Hadary2[MSS98] and MoNet[FK97](Motion Network) is used for Mutant. In these systems, graph nodes represents postures and arcs represents motions that connects those postures. CAAD includes an error recovery function for a collision avoidance. On the other hand, in our scheme all sensory information is included in nodes and an error detection and an error recovery function are embedded in the graph representation. And extensions, for example by sensor addition, can be done easily.

4 State Transition Graph Description of Action Space

In this section, following topics are discussed, (1)a definition of a robot state space which is able to include an error recovery function, (2)functions drawing out from that definition and (3)an error detection and recovery function.

4.1 State Space of Robot

A robot state at a certain time is represented by a state variable $\boldsymbol{X}$ which has several parameters. Using $\boldsymbol{X}$, a robot action A is described as a trajectory which exists in the state space as follows.

$$A = X(t) \tag{1}$$

In order to decide a robot state $\boldsymbol{X}$ uniquely at a certain time, all joint angles, a position and an orientation of reference coordinates on the robot body at that time are required. In the case the robot can control these parameters, the robot state should be described as follows. In **Eq.(2)**, $\boldsymbol{q}$ is a vector of joint angles , $\boldsymbol{p}$ is a position vector and $\boldsymbol{r}$ is a rotation vector.

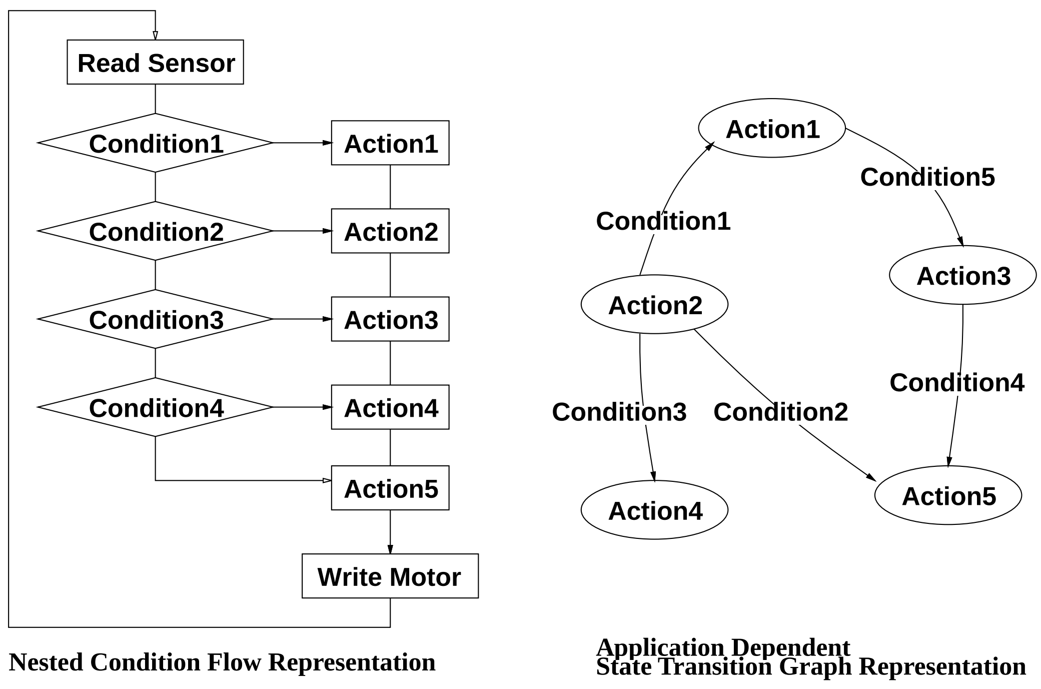


Fig. 5. Action Representations

$$\boldsymbol{X} = \begin{pmatrix} \boldsymbol{q} \\ \boldsymbol{p} \\ \boldsymbol{r} \end{pmatrix} \tag{2}$$

If the action is executed as planned, the robot state traces a pre-planned trajectory. However, because of internal or external factors, a deviation from this trajectory happens. This deviation is euqal to an error occurence and must be detected first in order to make an error recovery function. In order to detect errors, a planned state variable and a current perceived one should be compared. Therefore, the state variable must be defined only by perceptible information. If all parameters in **Eq.(2)** are measurable, **Eq.(2)** can be used as a state variable, but in many cases a direct measurement of these parameters is difficult. Therefore, in a proposed representation the robot state variable is defined using all sensory information as **Eq.(3)**. In **Eq.(3)**, $\boldsymbol{s_i}$ is a vector which has sensory information from sensor type i. Value ranges of elements in $\boldsymbol{s_i}$ is different from each sensor type. For absorbing these difference, weight coefficients w_i are used.

$$\boldsymbol{X} = \begin{pmatrix} w_1\boldsymbol{s_1} \\ w_2\boldsymbol{s_2} \\ \cdots \end{pmatrix} \tag{3}$$

Defining a state like this, if the robot has many sensors, it is expected that an accuracy of a recognition of a current state improves. Because it is possible to cover noisy data using other sensors. However, if the robot only has insufficient sensors, namely, a resolution of a state space is not sufficient, it is impossible to distinguish states that must be distinguished for an error recovery. As a result, a normal error recovery can't be expected.

Each elemental action is described as a trajectory that exists in this state space like **Eq.(1)**. When gathering several actions and some of them have junctions, they constructs a directional graph in the state space and this graph represents an action space of the robot at that time. This directional graph in this state space is called "StateNet".

4.2 Functions of StateNet

In many cases, robot action are described using a state transition. But most of those do not define a state space clearly and they are designed based on a human subjectivity. On the other hand, using a state space which is defined clearly, following function are acquired.

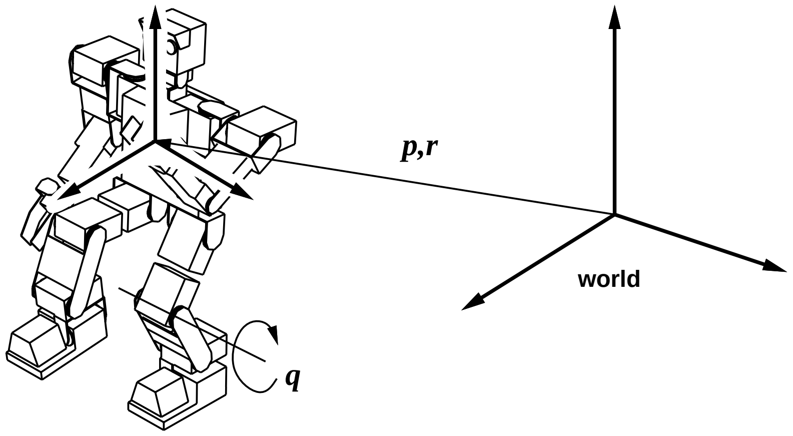


Fig. 6. State Space of Robot

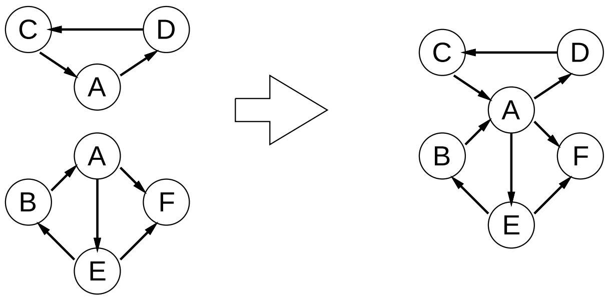


Fig. 7. Synthesis of StateNets

State Generation An automatic state generation is possible if a state is defined clearly. For example, since a state is constructed only by perceptible sensory information in StateNet, a new state can be defined by measuring those information and checking where a state has already existed or not. Defining a state space based on human subjectivity, a new state must be defined only by human.

Distance Calculation It is possible to calculate a distance between states. Because states are defined in an uniform space. A relation among states can be decided by a calculation. In the case a state is defined based on a human subjectivity, that relation is also decided only by the human subjectivity. A distance between states are calculated by the next formula.

$$distance(\boldsymbol{X_1}, \boldsymbol{X_2}) = ||\boldsymbol{X_1} - \boldsymbol{X_2}|| \tag{4}$$

Path Search Describing an action space as a directional graph and naming graph nodes for developers to distinguish them, it is possible to search action series getting to the target state from the graph structure and execute them in order only by directing a target state(node) from an application level. In the case the direction is done by the name of the action, it is required to confirm that preconditions to do the action is satisfied. Since a causal relationship between actions is described as a graph structure, preconditions of each action are automatically satisfied only by tracing a searched path to the target state.

Transition Generation If a relation between states can be calculated, there is a possibility of a transition generation. StateNet has joint angles as a part of state variables. Using those information, a joint angle sequence that connects two states straight in the state space can be generated and tried. If the transition finished successfully, the transition is registered as new one.

Space Integration Since states are defined in an uniform space, sub-graphs that are designed for different behaviors are integrated easily by a graph unification without any modification as shown in **Fig.7**. If it is described in human subjective form, states exist in difference spaces and it is difficult to integrate different sub-graphs.

5 Error Detection and Recovery

Researches on fault tolerant robots regard robots used in remote and hazardous environments such as in space, underwater and radioactive as targets especially. Most of these researches deal with manipulators[SL99], wheeled robots[Pun00] and underwater robots[PAS00]. But there is no precedent that deal with an error recovery function of a humanoid robot and its whole body actions.

It is possible for StateNet to equip following functions using characteristics mentioned above.

5.1 Error Detection

Using a state generation function and a distance calculation function, it is possible to detect an error occurence by comparing a current generated state and a pre-planned state. Ideally, a distance between a current state and a pre-planned state should be zero. If it is not zero, that means the action is not executed as pre-planned. Practically, considering there is a noise on sensory information and there is some allowable margin, for example, a stable margin while walking. Therefore, it is possible to set a tolerance for the distance and regard as an error if the distance exceed it. Namely, when the next formula is met, an error occurence is detected. In this formula, A is a pre-planned action, $\boldsymbol{X}(t)$ is a generated current state using sensory information and ε is the maximum error of distance between states.

$$distance(\boldsymbol{X}(t), A(t)) > \varepsilon \tag{5}$$

Executing this error detection function at all times while the robot is moving over StateNet, an application which control StateNet from outside need not to check whether an error occurs or not. When a state transition is described based on a developers subjectivity, any error can’t be detected if he doesn’t write an error detection routine. If there is an oversight of an error detection, that error is not able to be detected.

5.2 Error Recovery

Combining a distance calculation function and a path search function, there is possibility that errors are recovered automatically. As mentioned above, an error for the robot is equal to a gap occurence between pre-planned path and a path the robot moving actually. Therefore, an error recovery is equal to returning to StateNet from an undefined state and transit to a primary target state. But a path that exist in the state space is not always executable path for the robot. Therefore, returning path to StateNet should be the shortest path to the existing graph. The shortest path is equal to a straight line in the state space. This is because it is expected that a shorter path has a strong likelihood of its success.

For example, supposing the robot is shifting from State A to State B like (I) in **Fig.8**. But the robot leaves this transition path for a certain reason. Using **Eq.(5)** an error occurence is detected and at the same time a undefined state E is generated using sensory information. After that, the nearest node from the state E is searched to return to a existing graph and a state C is obtained. And the next, a straight path from State E to C is generated and this path is tried. If this transition is successful, moreover, a transition path from a state C to a state B is searched and executed. Finally, an error recovery completes.

Embedding an error recovery function in StateNet, an upper level application need not to describe error handlers link in case of an error detection. Of course, if a new path generated by a path generation function is not executable, an error recovery fails. In cases like this, a new transition from a generated state to the graph must be added by a developer. Incompleteness of an error recovery function causes problems in the early stages. Because a scale of StateNet is small and there is only few paths in the space in the early stages. Consequently, a length of a generated path for an error recovery becomes long and a failure probability will grow up. However, those problems will decrease gradually with extending a scale of StateNet using a space integration function and filling up the space with paths.

5.3 Relations between Type of Action and Error Detection

Robot actions are defined as paths in the state space. But they are not fixed at all times. It depends on implementation methods of actions whether the path is static or dynamic. In this point, path types are categorised into three types, a static path, a semi-dynamic path and dynamic path as shown in **Fig.9**. Relations among path types, realization methods of actions and error detection methods are as follows.

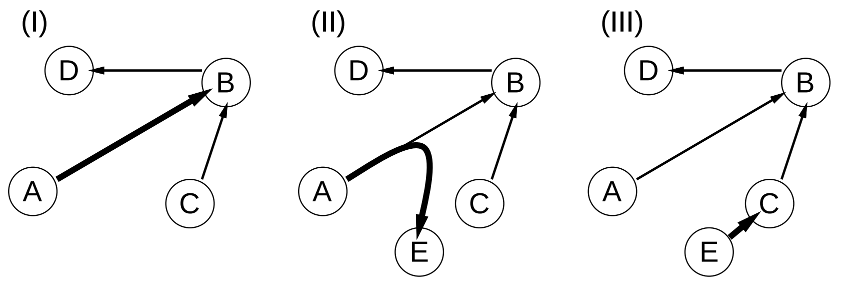


Fig. 8. Example of Error Recovery

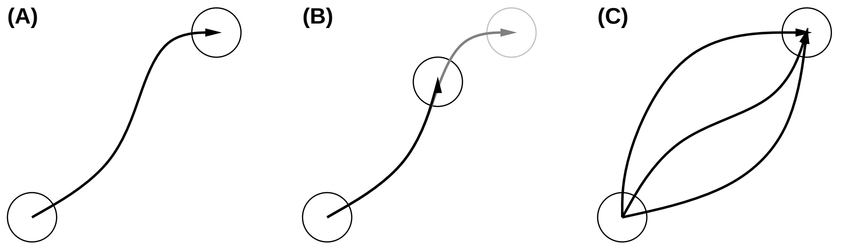


Fig. 9. Types of path, (A)Static, (B)Semi-dynamic, (C)Dynamic

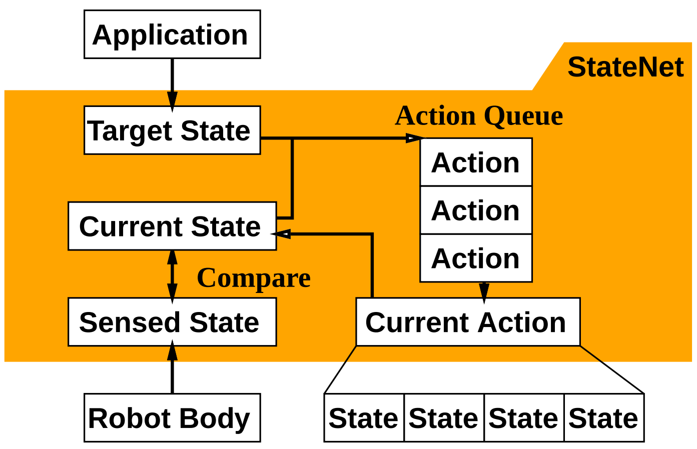
Static Path A static action means that sensory information measured while executing the action is predictable. That is, the action is realized by model based and feed-forward techniques. For example, an action acquired through learning in the simulation world is executed by playing back a simulated motion pattern. In this case, an error detection can be done by comparing a current state generated by sensory information and a predicted state.

Dynamic Path When the robot behaves adaptively to situations, many paths as many as situations exist for one named action. Actually the number of situations is countless. Therefore paths that describe actions like this are not static and they are defined dynamically when they are executed. For instance, when holding up an unknown object, the action changes according to size and weight of the object. In the case a path is generated dynamically, a pre-planned path doesn't exits. Therefore it is not possible to compare a generated path and a pre-planned path. In this case, developer must add constraint conditions explicitly and an error detection is done by checking these conditions. For instance, supposing a humanoid holds up an unknown object from the ground, of course, the humanoid must be standing while holding up. Adding this condition as a constraint, an error is detected and an error recovery function invoked when the robot falls forward or backward while holding up.

Semi-dynamic Path Actions that belong to a semi-dynamic path type are different from both types, static and dynamic, and have a moderate property. They are described by a constant trajectory, but their starting points or ending points are not fixed. For example, considering a sitting action, a humanoid moving down its hip from a standing state until its buttock contacts with a chair, a transition path until the contact is fixed for all chairs. But an ending point differs according to a height of the chair. On the other hand, considering standing up action from a sitting state to a standing state, this action has a non-fixed starting point and a fixed ending point. In this case, since a planned trajectory and a traced trajectory must be the same while moving, an error detection is done by a distance calculation function similar to a static path type.

6 Implementation of StateNet using EusLisp

StateNet is implemented using EusLisp based on specifications discussed in previous section. The whole structure of an implemented StateNet is shown in left of **Fig.10**. StateNet works using a target state($TargetState$) or a target action ($TargetAction$) directed by a higher level application. The reason these both targets are able to be directed is that it it impossible to execute recurrent actions if only a target state can be directed. The recurrent action is an action which recurs to the starting state. The recurrent action is removed by a path search function.



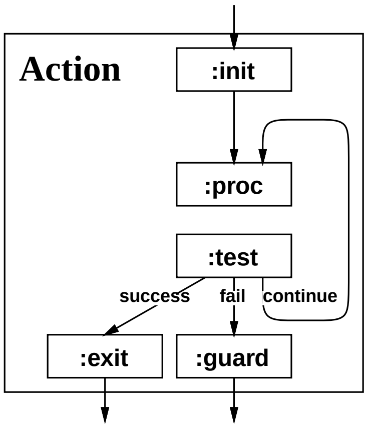


Fig. 10. Structure of implemented StateNet

The robot conforms that a state the robot shold be(*CurrentState*) is maintained while both a target state and a target action are not given. And an action starts after receiving a new direction. After receiving a new target state or action, at first StateNet checks a state of an action queue(*ActionQueue*). If that is not empty, all actions except an executing action are removed from the queue. And the next, Using a path search function, a action sequence from a current state or a ending state of an executing action to a target state is searched and it is set to an action queue. In the case a target action is passed, that is also queued.

Actions set in an action queue are implemented as instances of a class that has following five methods. A invokation relation between these methods is described in **Fig.10**.

`:init` This method is called only once when this action starts and sets up of an environment that is necessary for executing this action such as starting up autonomous low level functions.

`:proc` A main routine of this action is described in this method and invoked at every sampling time of sensors. An action is a time varied function as defined in **Eq.(1)**. In the case a path type of the action is a static, the action is described as a series of states that are sampled at every sampling time of sensors. Therefore, a method `:proc` picks up a state from this sequence in order, and a joint angles included in that state is sent as a motion command.

`:test` This method is also invoked at every sampling time and check a termination of this action and an occurence of an error. For example, in case of actions described by static paths, a distance between a picked up state and a current state(*SensedState*) is calculated by **Eq.(5)**. Using a calculated distance d and the maximum allowable error ε, a current time t and a planned ending time t_{end}, a return value is decided by the next formula.

$$return\ value = \begin{cases} continue & (t_{end} > t) \\ success & (t_{end} \leq t \ and \ d \leq \varepsilon) \\ fail & (t_{end} \leq t \ and \ d > \varepsilon) \end{cases} \quad (6)$$

If a return value is *continue*, a processing flow continues to stay in a current action. And in the case *success*, a current action terminates successfully, in the case *fail*, a processing flow switch to an error recovery mode.

`:guard` If an error is detected by method `:test`, a processing flow slips out of this action. But there is some cases that several functions must be called before slipping out. For example, stopping an autonomous function invoked by method `:init`.

`:exit` In the case this action terminates successfully, this method is called and does a clean up process and a restore process.

A processing flow of an error recovery mode is shown in **Fig.11**. At first, a current state is generated from sensory information and then the nearest state is searched from StateNet. The robot tries to transit to the state through the shortest path. If this transition succeeds, it means the robot returned to StateNet. In this case, an action queue is removed and searched actions to the target state are set and executed. If a returning to StateNet failed and a state before trying and one after trying are different, a returning to StateNet is tried again from that state. If a state before the recovery and one after that is the same, the robot stop its activity. Because that means an error recovery

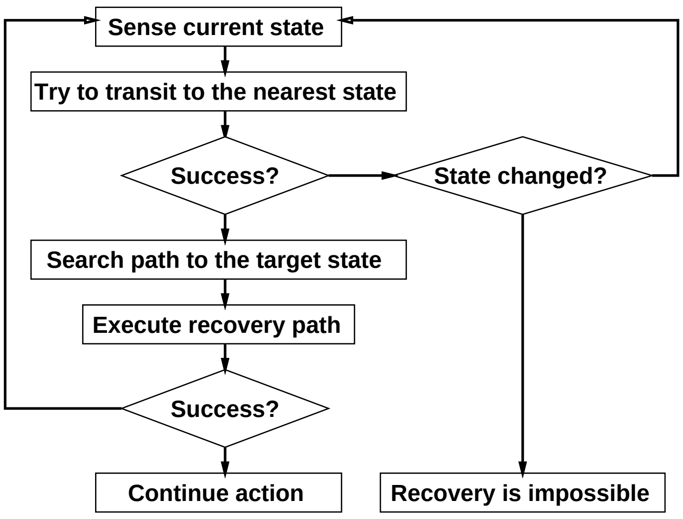


Fig. 11. Processing flow of error recovery function

from a current state is impossible only using a current StateNet. In this case, a developer must add a new action which connects between a state which is generated when the error occurs and a certain state which already exists in StateNet.

7 Error Recovery of Humanoid from Falling Down State

The humanoid robot has an ability to do various actions using many DOF. At the same time, working environments of the humanoid is also various. As a result, variations of possible errors are innumerable. StateNet is especially suitable for representing actions of robots like this. In this section, actions of the humanoid is represented using StateNet, and the humanoid stands up by itself and continue to work even if it falls down.

StateNet is applied to the humanoid robot shown in **Fig.12**. This robot is a small-size humanoid robot which is developed based on remote-brained approach[Ina93]. It has 18 DOF, 50[cm] height and 2.0[kg] weight. Remodelled RC servo modules that can measure joint torques are used as joints. Pressure sensors are attached on each corner of soles and buttocks. They are used for measuring foot forces and sitting state. A three axes accelerometer is attached on its chest for measuring a gravity direction. A camera is installed as its head. A control signal to actuators and sensory information are communicated to a remote brain software through onbody network[KMK$^+$98].

A structure of StateNet for this robot is as shown in **Fig.13**. There is 17 states and 48 actions. Among these actions, a knuckle walking, a biped walking, a holding up an object from the ground, a sitting down on the chair and a standing up action from lying on the face or the back are included.

In order to control the humanoid using StateNet, an application shown in **Fig.14** is developed. Through this graphical user interface, directions such as walk, sit, hold and release are transmitted to StateNet.

Fig.16 shows snapshots taken while an experiment of driving StateNet from a user application. At first, moving back by knuckle walking is directed. In the case of walking action, a target action is directed. Because walking direction is not able to be decided from a target state. In this knuckle walking case, two actions, moving back a left leg *Knuckle(B,L)* and moving back a right leg *Knuckle(B,R)*, are directed by turns as a target action.

And the next, moving forward by dynamic walking is directed at time (A). Internally, a dynamic walking action *March* is transmitted to StateNet as a target action. On StateNet side, for changing a walking method from knuckle walking to dynamic walking, a transition path is searched from an ending state of a executing knuckle walking *Knuckle(B,L)* to a starting state of dynamic walking March. A searched path is set to an action queue and executed sequentially. A first frame in the second row, the robot falls down on its back while doing a standing up action *Stand-up* for some reason(time (B)). At this time, an error is detected using information from an accelerometer and pressure sensors and an error recovery function is invoked automatically. At first, the nearest state from a state lying on the back is searched from **Fig.13** and a state *Lie on the back* is obtained as a result. Shifting to that state is tried and succeeded. And then, a path from a state *Lie on the back* to a state *Neutral* that is a starting state of a target action is searched and a result is set to an action queue. The humanoid is executing recovery actions, rolling over and standing up in the third and fourth row. In the fifth row, after arriving a standing state that is a starting state

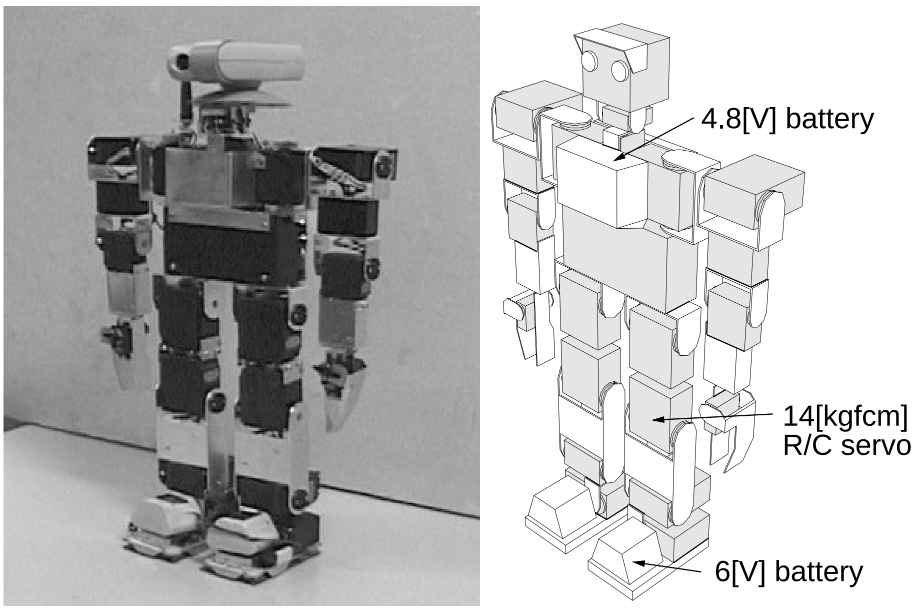


Fig. 12. Remote-Brained Humanoid Robot “Saizo”

of dynamic walk, a primary target action *March* started. While recoverying autonomously, an upper level doesn’t participate in any error detection and recovery. It is able to concentrates on an essential part of a target action.

8 Summary and Conclusions

In this paper, StateNet architecture is proposed as a developmental realization method of humanoid whole-body behaviors. This architecture represents an action space as a state transition graph which exists in a state space defined by sensory information. And it has following characteristics, (1)it is easy to extend incrementally and integrate and (2)Errors are detected and recovered autonomously. In this graph, nodes represent states shared by several actions or special states named by developers. And arcs represent actions connects those states. This representation has useful functions such as a state generation, a distance calculation, a path search, a path generation and a space integration. Combining these functions, it is possible to embed automatic error detection and recovery functions in this representation. Since an error recovery function is involved in a lower level, an upper level need not to be conscious of handling exceptions. A developer is able to devote himself on describing a main process of an target action.

Acknowledgments

This research has been partly supported by Graint-in-Aid for Research for the Future Program of the Japan Society for the Promotion of Science, “Research on Micro and Soft-Mechanics Integration for Bio-mimetic Machines(JSPS-RFTF96P00801)” project and several grants of Grant-in-Aid for Scientific Research.

References

[FK97] Masahiro Fujita and Koji Kageyama. An Open Architecture for Robot Entertainment. In *Autonomous Agents 97*, pages 435–442, 1997.

[HII98] Yukiko HOSHINO, Masayuki INABA, and Hirochika INOUE. Model and Processing of Whole-body Tactile Sensor Suit for Human-Robot Contact Interaction. In *Proc. of the 1998 IEEE International Conference on Robotics & Automation*, pages 2281–2286, 1998.

[IIKI96] M. Inaba, T. Igarashi, S. Kagami, and H. Inoue. A 35 DOF Humanoid that can Coordinate Arms and Legs in Standing up, Reaching and Grasping an Object. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 29–36, 1996.

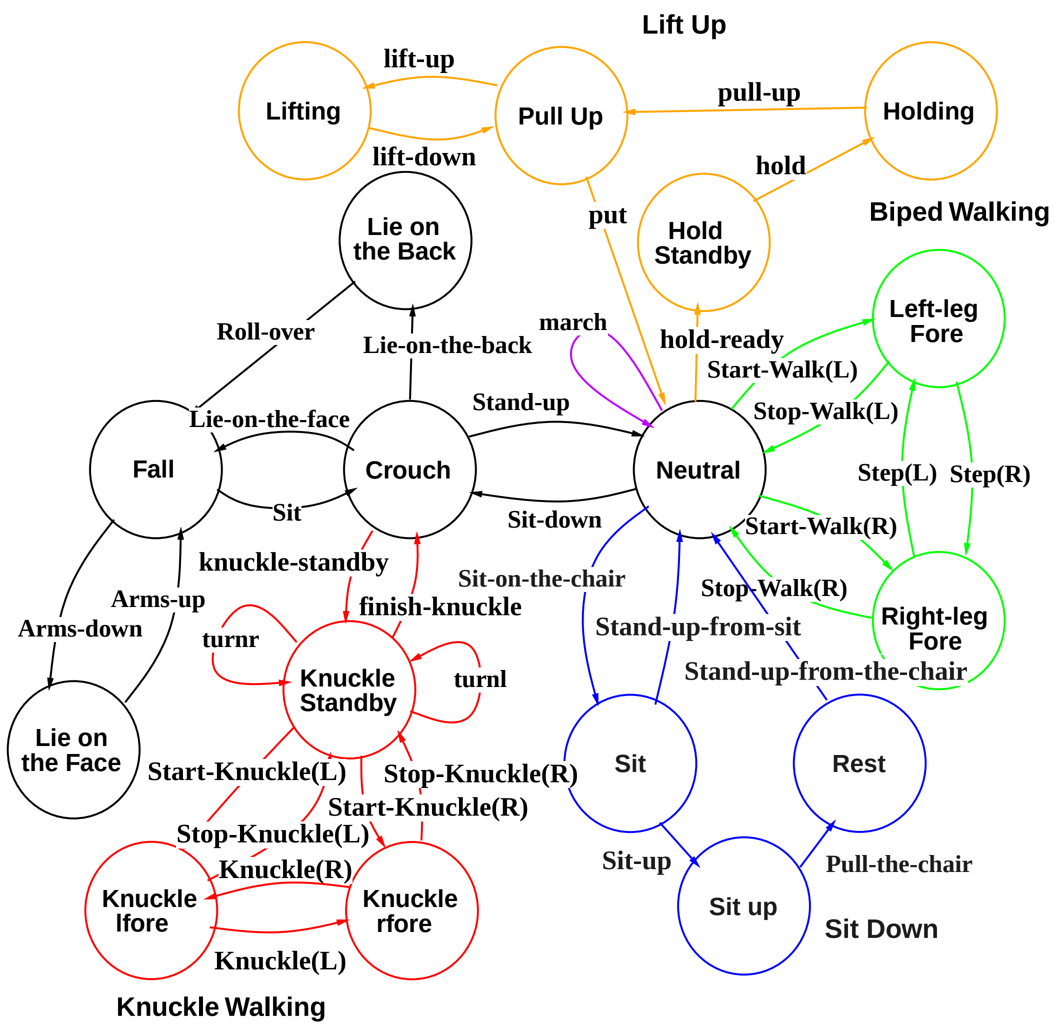


Fig. 13. StateNet of humanoid robot

[IKK^{+}96] M. Inaba, S. Kagami, F. Kanehiro, K. Takeda, T. Oka, and H. Inoue. Vision-based adaptive and interactive behaviors in mechanical animals using the remote-brained approach. *Robotics and Autonomous Systems*, 17(1–2):35–52, 1996.

[IKKI95] Masayuki Inaba, Fumio Kanehiro, Satoshi Kagami, and Hirochika Inoue. Two-Armed Bipedal Robot that can Walk, Roll-over and Stand up. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 297–302, 1995.

[Ina93] M. Inaba. Remote-Brained Robotics: Interfacing AI with Real World Behaviors. In *Proceedings of 1993 International Symposium on Robotics Research*, 1993.

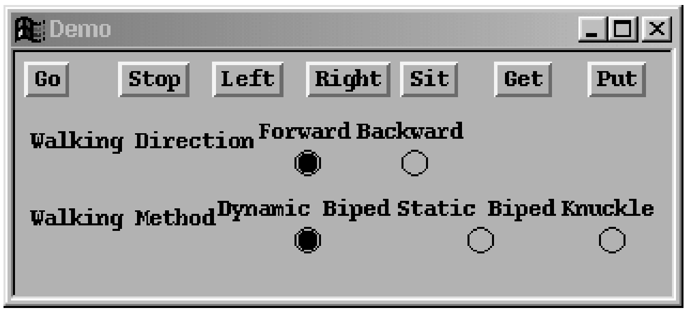
[INH^{+}97] M. Inaba, T. Ninomiya, Y. Hoshino, K. Nagasaka, S. Kagami, and H. Inoue. A Remote-Brained Full-Body Humanoid with Multisensor Imaging System of Binocular Viewer, Ears, Wrist Force and Tactile Sensor Suit. In *Proc. of the 1997 IEEE International Conference on Robotics & Automation*, pages 2497–2502, 1997.

[INO98] Hirochika INOUE. A Platform-based Humanoid Robot Project. In *IARP First International Workshop on Humanoid and Human Friendly Robotics*, pages 1–4, 1998.

[KII96] Fumio Kanehiro, Masayuki Inaba, and Hirochika Inoue. Development of a Two-Armed Bipedal Robot that can Walk and Carry Objects. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 23–28, 1996.

[KII99] Fumio Kanehiro, Masayuki Inaba, and Hirochika Inoue. Action Acquisition Framework for Humanoid Robots based on Kinematics and Dynamics Adaptation. In *Proc. of the 1999 IEEE International Conference on Robotics & Automation*, pages 1038–1043, 1999.

[KKT^{+}00] Satoshi KAGAMI, Fumio KANEHIRO, Yukiharu TAMIYA, Masa yuki INABA, and Hirochika INOUE. Autobalancer: An online dynamic balance compensation scheme for humanoid robots. In *Proc. of Fourth Intl. Workshop on Algorithmic Foundations o n Robotics (WAFR'00)*, 2000.



Walking Direction: [✓ Forward] Backward

Walking Method: [✓ Dynamic Biped] Static Biped Knuckle

Fig. 14. Controller of StateNet

[KMK+98] Fumio Kanehiro, Ikuo Mizuuchi, Kotaro Koyasako, Youhei Kakiuchi, Masayuki Inaba, and Hirochika Inoue. Development of a Remote-Brained Humanoid for Research on Whole Body Action. In *Proc. of the 1998 IEEE International Conference on Robotics & Automation*, pages 1302–1307, 1998.

[KTII99] Fumio Kanehiro, Yukiharu Tamiya, Masayuki Inaba, and Hirochika Inoue. Developmental Methodology for Building Whole Body Humanoid System. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robotics and Systems (IROS'99)*, pages 1210–1215, 1999.

[MI90] T. Matsui and M. Inaba. EusLisp: An Object-Based Implementation of Lisp. *Journal of Information Processing*, 13(3):327–338, 1990.

[MSS98] Toshio MORITA, Koiji SHIBUYA, and Shigeki SUGANO. Design and Control of Mobile Manipulation System for Human Symbiotic Humanoid: Hadaly-2. In *Proc. of the 1998 IEEE International Conference on Robotics & Automation*, pages 1315–1320, 1998.

[MUK+97] T. Morita, T. Ueda, S. Kayaba, D. Yamamoto, Y. Kunitake, and S. Sugano. A Motion Planning Strategy for a Human Symbiotic Humanoid: Hadaly-2. In *Proc. 15th Annual Conference of Robotics Society of Japan*, pages 407–408, 1997.

[NKII97] K. Nagasaka, A. Konno, M. Inaba, and H. Inoue. Acquisition of Visually Guided Swing Motion Based on Genetic Algorithms and Neural Networks in Two-Armed Bipedal Robot. In *Proc. of the 1997 IEEE International Conference on Robotics & Automation*, pages 2944–2949, 1997.

[oHFR98] Research Committee on Human Friendly Robot. Technical Targets of Human Friendly Robots. *Journal of the Robotics Society of Japan*, 16(3):288–294, 1998.

[PAS00] Tarun Kanti Podder, Gianluca Antonelli, and Nilanjan Sarkar. Fault tolerant control of an autonomous underwater vehicle under thruster redundancy: Simulations and experiments. In *Proc. of the 2000 IEEE International Conference on Robotics & Automation*, pages 1251–1256, 2000.

[Pun00] Puneet Goel and Göksel Dedeoglu and Stergios I. Roumeliotis and Gaurav S. Sukhatme. Fault detection and identification in a mobile robot using multiple model estimation and neural network. In *Proc. of the 2000 IEEE International Conference on Robotics & Automation*, pages 2302–2307, 2000.

[SL99] Jin-Ho Shin and Ju-Jang Lee. Fault detection and robust fault recovery control for robot manipulators with actuator failures. In *Proc. of the 1999 IEEE International Conference on Robotics & Automation*, pages 861–866, 1999.

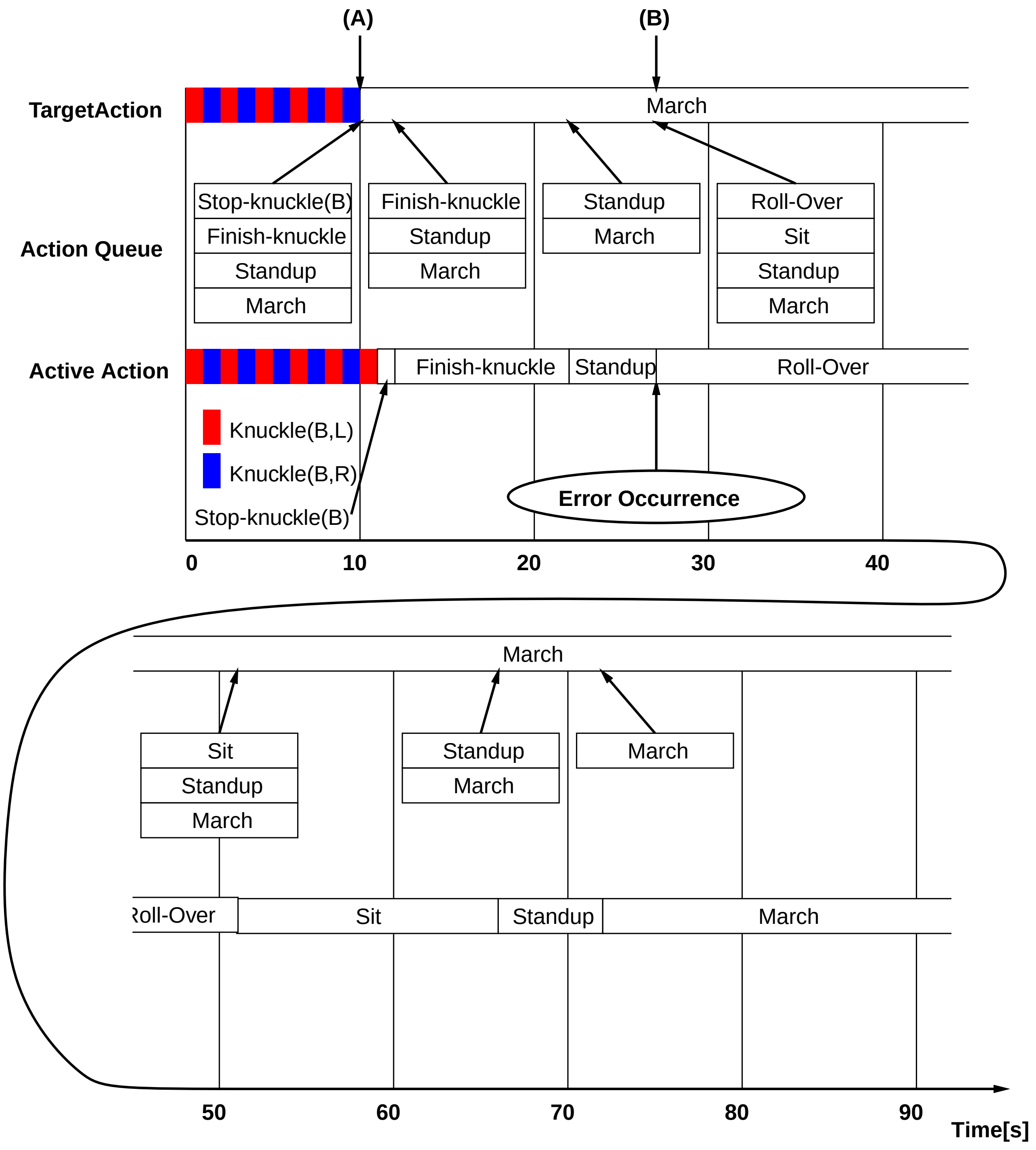


Fig. 15. Time chart of experiment

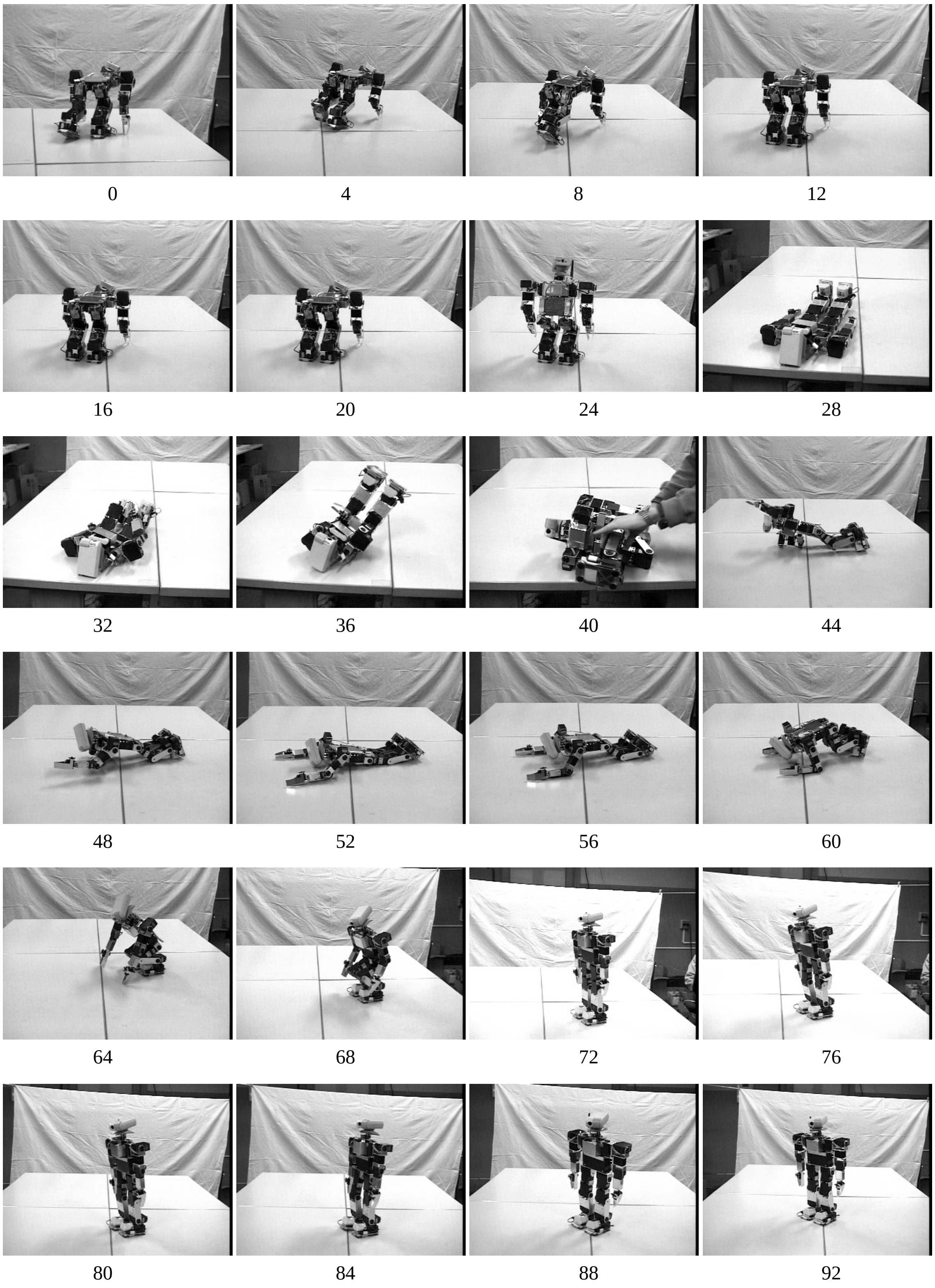


Fig. 16. Snapshots of experiment