The NeverMost 3D Soccer Simulation Team Desctiption 2012

Yalong Yang, Yang Yu, Haobo Ma,
Jun Song, Chang Gao, and Jianrong Wang

Computer Science Department
Software Engineering Department
Tianjin University, Tianjin, China
`http://cs.tju.edu.cn`

Abstract. This paper presents an overview of the NeverMost team. In this release we use the vision information of the field lines to enhance the accuracy of self-localization. We also establish a new mechanism of communication that transfer information between Cerebrum and Cerebellum. We develop a framework of extracting higher level information from the basic information that get from the server. In this paper, a description of our current and future works is presented. *abstract* environment.

1 Introduction

NeverMost 3D soccer simulation team, established in 2010, is the third 3D soccer simulation team from Tianjin University. Robocup 3D Simulation has been long kept as a major subject of our team. It is a young team that boasts many sedulous researchers and honored by excellent results. In August 2011, we expanded the our team by recruting new members. Thanks to the brilliant work of RoboCup 3D Simulation League, the cute humanoid robot Nao and the powerful simulation platform make a great amount of people become interested in the 3D simulation field. Compared with the other leagues, programming humanoid agents under the simspark brings several advantages such as making simplified assumptions about the world, low debugging costs, and the ability to automate experiments, especially for beginners.

This paper represents the major ideas and the establiment of NeverMost 3D soccer simulation team. Different from the first 3D soccer simulation team of Tianjin University, the TJUnited, the architecture of NeverMost is based on the NeverMore 3D soccer simulation team (the second team from Tianjin University) [1] in some degree. After a year of development, the NeverMost has grown up from a small and simple team to a strong team with sound designed and stable implemented achievements.

The whole architecture of NeverMost agent can be simply described into three levels hierarchically.

Data-processing level, which we called "WorldModel", commits two tasks. First, maintain the world states, such as the information of each perceptor; and second, pick up more complex information from the raw data offered by the sever, such as agent's global position and the situation of current game.

The strategy level, which we called "Cerebrum", is based on an HTN(Hieratical Task Network) planner. This part was written in PROLOG, and integrated into agent by the use of interface of SWI-Prolog to C++[2]. As we can imagine, the task of "Cerebrum" is to think the behaviors, which can be treated as a communication protocol that transfer information between Cerebrum and Cerebellum, like BeamBehavior, WalkBehavior and so on.

The Operation Level, which executes agent's behaviors from the Cerebrum. Mainly the instruction type behaviors, like beam; and action type behaviors, like walk, shoot and etc. We use the kinematics to implement a real time calculation of agent's action instead of the use of the static data which we use in the last version NeverMore, in the piece of biped walking, we use the preview control of ZMP to get a stable walking action. This level is written in C++.

The detailed concepts of NeverMost design are presented in the following sections. Section 2 introduces the overall agent architecture. Section 3 presents the World Model Data Processor. Section 4 illustrates the Method of agent locating. Section 5 explain the concept of Behavior, following by Section 6 Cerebellum and Conclusions and future works in Section 7.

2 Agent Architecture

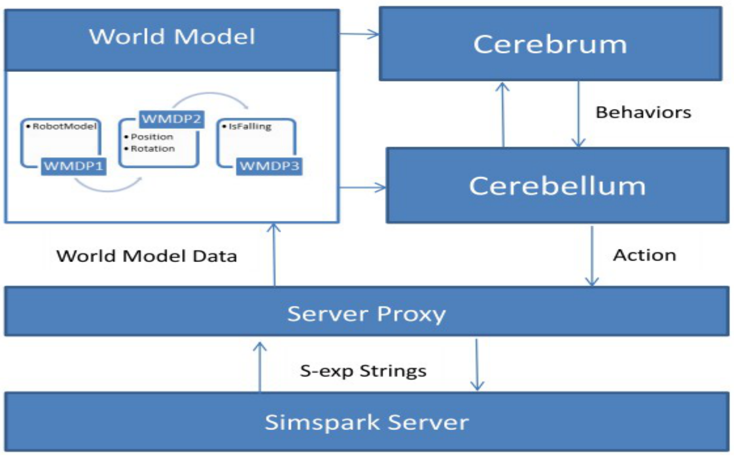

Fig. 1. Agent Architecture

Similar with the last version Nevermore, the NeverMost agent uses a flexible architecture which based on the idea of Vorsts Layered architecture[3], as demonstrated in Fig.1. Each component is explained as follow:

– **Server Proxy**: This module is responsible for communication with the server. There are network submodule, effectors submodule, perceptors submodule, agentproxy submodule in server proxy module. This module can make the conversion between s-expression and program internal strong type data easily.
– **World Model**: The world model is the abstract of simulation world in the rcssserver3D. It saves the internal representation of the environment and on the other hand, extracts useful high level abstraction from this information by using World Model Data Processor(following referred to as WMDP). WMDP is a list of processors, which can get more abstracted information like agent location. The front WMDP can be accessed by behind WMDPs. This feature can make our World Model more flexible.
– **Cerebellum**: the execution module of the agent. Cerebellum controls the joints of the agent, and receives the behavior from the Cerebrum and executes it with monitoring mechanism.
– **Cerebrum**: the thinking component of the agent. It thinks over the next best behavior for the whole team with the given world state.

3 World Model Data Processor

As the same idea of last year version of NeverMore, World Model Data Processor (following referred to as WMDP) is a mechanism designed for processing the primitive data from the World Model. Each data processor focus on one aspect of analysis of the world state.[1] As the information from the server are all basic data, which include data from the perceptors and the basic status of the game. While for the strategy, more abstract information is needed, like the global position of the players, whether the agent is falling down or not, and etc. Analyzing the basic data and extracting the higher information is what the framework does. At the same time, we found that the processing of the higher information may cause the interdependent problem in WMDPs. That is, one WMDP may need the result of another WMDP, for example, when we want to know the situation of the game, the players global position is needed as the input data. Therefore, under this framework, this problem can be solved by adding properties in WMD-P that mark those needed WMDPs, and use the topological sort algorithm to determine the processing order of all the WMDPs.

4 Method of agent locating

Before the rcssserver3D adds lines to vision perceptor, locating agent itself uses the flag around the field to perform. During one cycle of server, if more than three flags can be seen, the calculation of the position of agent should be applied intersection of three spheres formula, then calculate the rotation of agent with the horizontal angle, vertical angle, and the position we get before. If only two flags can be seen, an algorithm like the Robocup 2D location method[4] is best applied here. However, our team has made some improvement from assuming the z value of position vector to be zero. We use forward kinematics to calculate the height of agent, and then set z to this height. It is in this way that accurate the location.

Lines are added to vision perceptor in new version of rcssserver3D. With many fixed lines in the soccer field, agent can see a series of points including the endpoints of lines and the intersection points caused by visual limit. If the endpoints of lines can be extracted, just like there are more fixed flags on the field so as to reduce the difficulty in the calculation of locating agent.

The following several steps are the algorithm of the extraction the endpoints of lines:

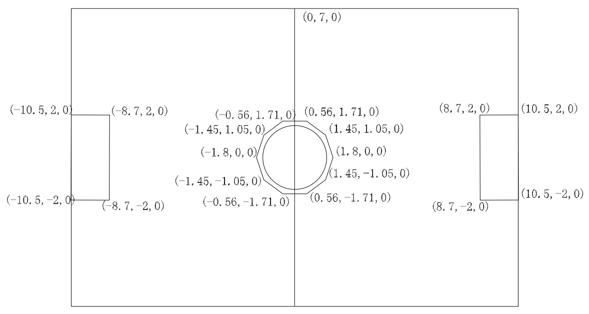

Fig. 2. Soccer Field

First of all, we locate the endpoints of lines by using traditional location method (turn off the noise and limit of vision). The result is shown in Fig. 2. Secondly, extract the endpoints of lines based on the distance between endpoints and flags (F1L, F1R, etc.) in normal situation. We can map this endpoints to the position of points on Fig. 2. It is noteworthy that this algorithm is independent from the traditional location method's result. In the end, more points with global position and vision information can be gained, i.e., more global flags are available for locating.

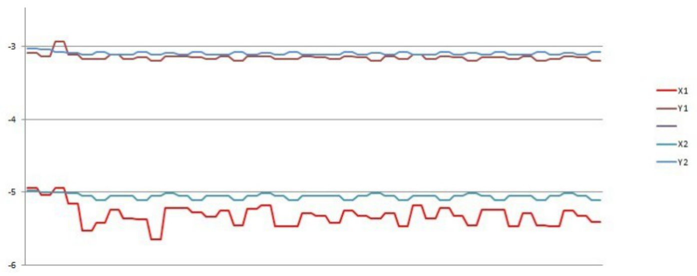

Fig. 3. Locating Results

The Fig. 3 shows the locating result after beaming to (-3, -5). The red line is the result of traditional location method, and the blue line added lines. It is obvious that the lines perceptor adding make location more accurate.

5 Behavior

Behavior works as a communication mechanism that transfers information between Cerebrum and Cerebellum, which describes the result of action planning. An appropriate structure of behavior is crucial to extensible and stable design of cerebrum and cerebellum.

As the reasons above, we have designed a new way to describe the data for communication:

1. Parameters for a behavior
2. Resources which is needed by executing the behavior
3. Priority of the behavior

At the same time some revolutionary changes have been made in the implement of the behaviors:

1. **Make the coordinate system of the walk behavior relative**
   In the Tjunited and NeverMore version, description of parameters of the walk behavior is based on the global Cartesian coordinate system. However, the dealing with the frequent change of the destination turns to be a tough one. In Nevermost, We choose the relative coordinate system. It is a system that uses the agent as the origin and the direction as the axis, and then commands such as left, right and etc. can be sent smoothly, just like authentic football playing. In this case, we will be able to get a flexible walk decision, and the parameters are also useful for the online biped walk pattern generator algorithm, like using preview control of zero-moment point[5].

2. **Implement the multiple behavior handle system**
   For handling multiple behaviors, needed resources and the priority of a behavior are added in description. When the Cerebellum generates multiple behaviors in one cycle, the system will check whether there is a conflict in the use of resource promptly. If there is no conflict, the multiple behaviors can execute together. Otherwise we use the following algorithm to solve the conflict.
   (a) Get the behavior which has the highest priority, and add to the canAct list

   (b) Delete all the behaviors that has conflict with the behavior got from step (a)

   (c) If exist leaving behaviors jump to step (a), else finish algorithm. The canAct list is the final result

With the algorithm above, it won't appear the conflicts, and it will also make as much behaviors act as possible.

For example,

(a) TurnHeadBehavior will use the resource of the head joint, and WalkBehavior will use the joints on the leg. There is no conflict, then these two behaviors can act together

(b) StandUpBehavior will use the resource of all the joints of the agent, so there will be conflicts between StandUpBehavior and the WalkBehavior. However, the StandUpBehavior has a higher priority, so the system will choose the StandUpBehavior to execute.

6 Cerebellum

Cerebellum is the core module of generating and managing motions of Nao. Since its major function is to translate behaviors received from the Cerebrum into arrays of actions which can be recognized by the agent. In addition, Cerebellum also monitors the execution of those behaviors by using the information stored in the World Model.

Behaviors are generated by Cerebrum and passed to Cerebellum, which using the current agent state, will translate them into a sequence of activities and each of activities will again be translated into a sequence of key frame. Key frames are Fig.d out by Frame Manager depending on the kind of activity it processed and the current frame, also called the current agent state. For more Frame Manager is also responsible for calculating the transition from the current frame to the target key frame.

6.1 Frame Manager

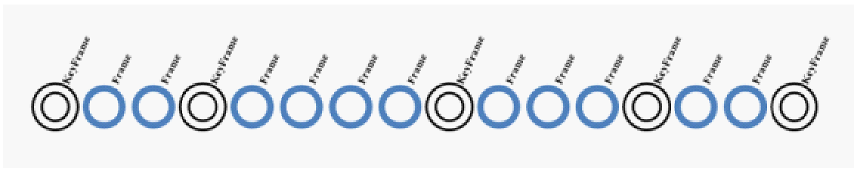

Fig. 4. Frames and Key Frames

Frame Manager takes care of the transition from current frame to the target frame.

Frame One frame is the concept of an agent state in a specific time. An array of frames can form one behavior. There are some frames among those frames that are called as key frames. For example, the turning point of a joint. Those frames between key frames can easily be computed by using proper function such as linear or trigonometric functions.

Key Frame Key frames are significant frames in a behavior. The importance of key frames lies in not only the fact that they can be used to compute other frames, but in they provide the possibilities to connect two different activities seamlessly.

It is obvious that not every frame can be treated as a key frame. A key frame must accord with the "3P" rule, saying, "proper position, proper velocity and proper torque". One key frame must satisfy each of its joints has a proper position, a proper velocity and a proper torque. The "3P" rule makes sure the stability of the Frame-KeyFrame design.

One execution of behavior is an execution of actions that make the agent reaches key frames one after another and at last reaches expected target frame. This series of the actions is also called Activity. The key frame system makes the generation and debugging of a behavior more convenient. We neednt concern about the total process any more, but only the states between key frames instead. It becomes much easier for us to find a function to process each joint's movement, as those key frames separate each behavior into short segments. What's more, it provides us the seamless connection between two behaviors and the flexibility of behavior transition.

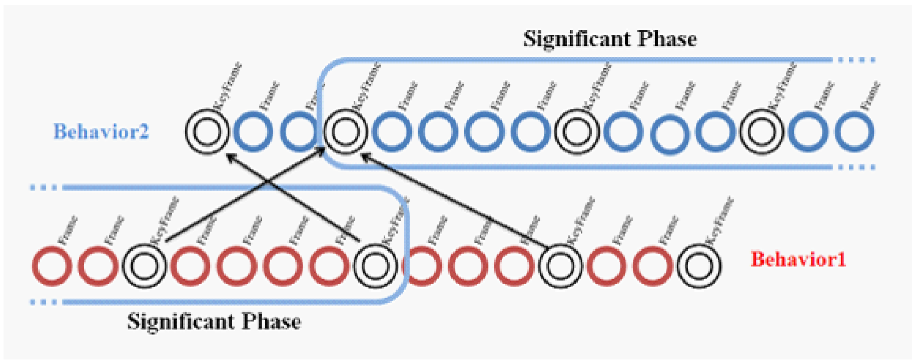

Fig. 5. Transition from Behavior 1 to Behavior 2

Fig. 5 shows how the Frame Manager implements the transition between two behaviors. Frame Manager searches the shorted path of the Key Frames from all the current behavior to the next one. We have built a function to calculate the difference between two frames. For each behavior, there is a "significant phase". The transition from the current behavior to the next behavior should be finished before the significant phase of the next one, because it would be meaningless if we just reach the tail of a behavior by transition, which may cause the agent to do some unexpected activities.

7 Conclusions and future works

This paper is an introduction of the features and implementation of our team, including the architecture of the agent, the self-localization of agent, the method to maintain the higher level information of the WorldModel, the frame system and the information transformed between the Cerebrum and Cerebellum. We use the lines in vision perceptor of the new version of rcssserver to localize the agent, and the result proves that it will make the self-localization more accurate.

The follow-up of NeverMost agent development will mainly focus on the following aspects:

- use the reinforcement learning to make the implement of agent's actions more stable and more efficient
- combine the machine learning with the preview control to improve the stability of the biped walking
- in the self-localization, make use of the intersection between the vision area and the lines of the field
- make a conclusion of the information of the fixed point of the field line and submit it to the official simspark wiki

References

1. Yaolong Huang, Chao Ma, Haobo Ma, Yalong Yang, Yang Yu,Jun Song and Jianrong Wang, The NeverMore 3D Soccer Simulation Team Description 2011. Robot Soccer World Cup, Istanbul, Turkey. July 2011.
2. Jan Wielemaker: An overview of the SWI-Prolog programming environment. In Fred Mesnard and Alexander Serebenik, editors, Proceedings of the 13th International Work shop on Logic Programming Environments, pages 116, Heverlee, Belgium.2003.
3. Philipp Vorst: Readylog agents for the robocup 3d soccer simulation league, Masters thesis, RWTH Aachen.2006.
4. Jelle Kok, Remco de Boer, Nikos Vlassis, and Frans Groen. UvA Trilearn 2002 Team Description. Robot Soccer World Cup. 2002.
5. Shuuji Kajita, Fumio Kanehiro, Kenjio Kaneko, Kiyoshi Fujiwara, Kensuke Harada, Kazuhito Yokoi and Hirohisa Hirukawa. Biped wlaking pattern generation using preview control of zero-moment point. In International Conference on Robotics and Automation; Proceedings of the 2003 IEEE, 2003.