Wright Eagle 3-D Simulation Team

Chen Xiaoping, Liu Jinsu, Xue Feng, Lai Xinghua

AI Center of Department of Computer Science and Technology,
University of Science and Technology of China,
Hefei, China

jslau@mail.ustc.edu.cn WWW home page: http://mas.ai.ustc.edu.cn

Abstract. This paper introduces the implementation of Wright Eagle 3-D Simulation team. Simulation3-D is a new domain of RoboCup Soccer-Simulation, which is more realistic than the old 2-D. We have tried several different approaches to resolve problems in the extension to 3D. In our application, we optimize the architecture of agent, and implement agent locating using the method of Kalman Filer.

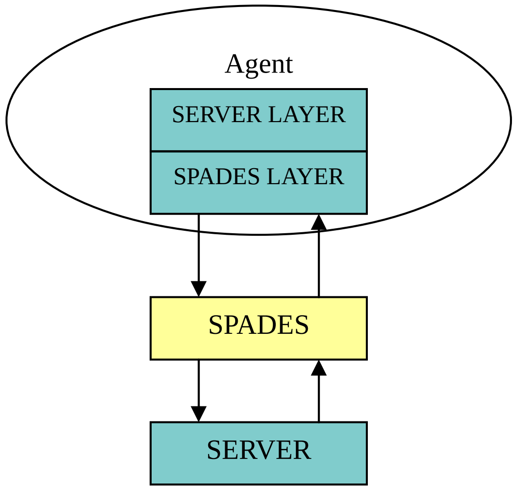
1 Overview

In this paper, we mainly talk about two subjects. The first part of this paper proposes a new architecture that improves independenatbility between different modules and is more suitable for making changes in SPADES and Server, which makes it more convenient to develop complex agent. The other part of the paper shows how the action can be made more precise with Kalman Filter in agent locating and it also helps to find a balance between reaction speed and precision of location.

2 Architecture

2.1 The old architecture

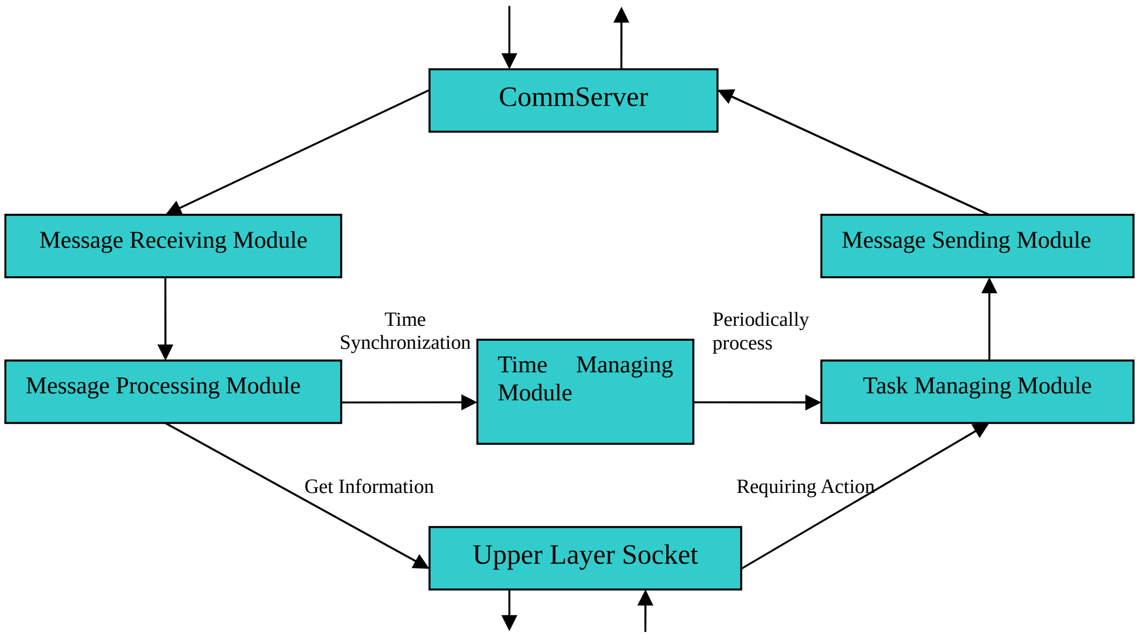
According to the analysis of the simulation system, to implement an agent in a 3D simulation system involves two parts of work, i.e. the agent should communicate well with both SPADES and Server. As the server is built above SPADES, an efficient player should firstly be an agent meeting SPADES' demand, and secondly adept in communication, that is, to act in accordance with the messages sent from the server. In this way, the two-layer structure is illustrated by the figure, one of which deals with events on SPADES layer, the other on the Server layer.



Although we have been aware the two-layer structure of SPADES and server, actually we did not completely split them up. Many components within the server can still process messages such as S, K, I, etc, from the SPADES, while within SPADES, periodical counting about server layer continue to exist. Such structure not only may cause understanding confusion on the short term, the concepts on different layers are not well distinguished. What's more, on the long term, under such structure, if the simulation system alters, much work will be involved to update the agent, which is a serious problem. Consequently, we have to make proper changes to this overall structure.

2.2 Improvement to SPADES Layer

According to the previous experience of competition, **SPADES layer** can further be abstracted to be a component exclusively responsible for communication, while SPADES is factually the detailed implementation of the component. If we manage to abstract this component and keep the interface towards higher layer invariable, the flexibility can be greatly improved. Whatever communication method is adapted, relevant communication work can solve the problem, that is, to submit the result to the higher component via the interface.



This component is responsible to mask the low-layer communication from higher layer, with interface as follows:

GetMessage: to get messages from low layer;

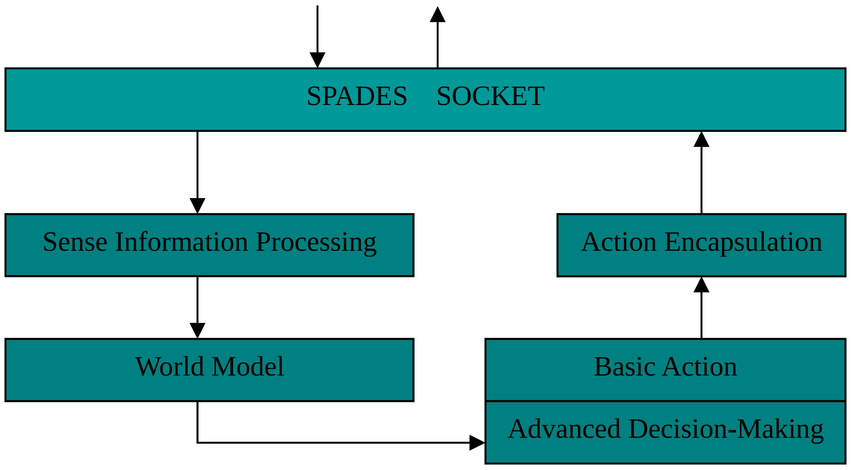
PutMessage: to send out messages from the upper layer;

GetSimTime: to get current simulation time (optional, if time stamp is included in the message from the low layer, this interface can be used.)

2.2 Improvement to Server Layer

The **Server LAYER** handles the data that **SPADES LAYER** receives according to the form defined inside the Server, make decision, pack the data, and finally send the data through **SPADES LAYER**.

Its Structure is illustrated by the figure below:



Sense Processing Model distills various information from the Perceivable Information (character strings) that Server sends to the agent, parsing strings. The contents parsed include visual info, game state and agent state. Game state and agent state can be accepted without further processing, but visual info is disrupted by the noise added manually. Whether we store the visual info directly or after processing relates to the demands of the decision. Generally we store them both and provide two kinds of interface to access both the origin info and the refined info.
World Model Component is actually a storage pool storing various kinds of information that we need. Mostly it stores the data that **Perceivable Information Handle Model** processes for the agent reasoning.
Action Encapsulation Component encodes the actions decided by the agent into command character strings defined inside the Server and passes them to **SPADES LAYER.**
Agent Thinking Component is the kernel and most complex part of the whole Agent System. Originally it must be an independent layer, but we simplify the layer structure and combine it with the **Server LAYER**. The component makes a series of actions for the agent through computing and logic system to accomplish a specified target. Here we provide two basic layers aimed at soccer simulation to finish basic action control and thinking decision.
Actually, the demands of the **Server Layer** are much more complicated than the structure discussed above. There are many more problems to consider in detail. And we'll finish them in the future step by step.

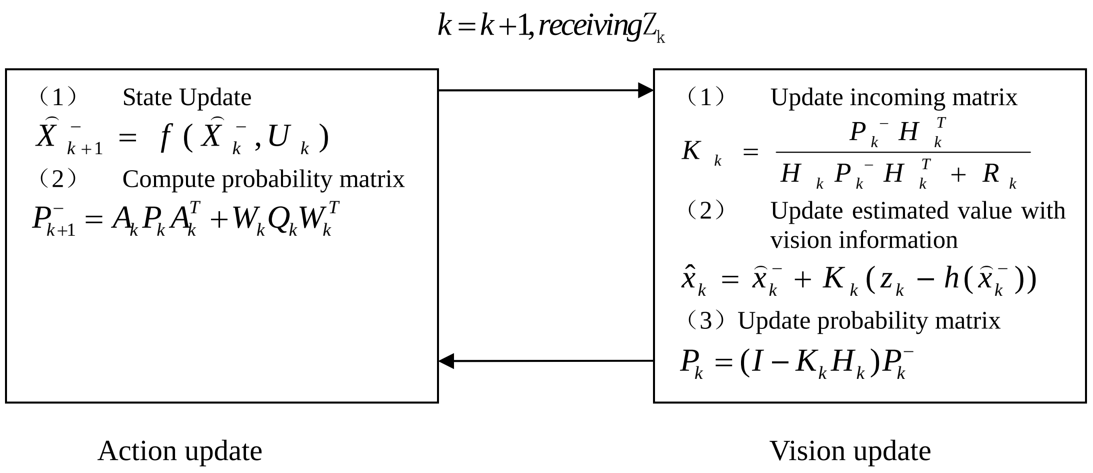
3 Location

3.1 Kalman filter

Kalman filter is used to eliminate the vision error with the purpose of location. The basic equations of the Kalman filter are as follows:

$$\hat{x}_{k+1/k} = \Phi_k \hat{x}_{k/k} + B_k u_k$$
$$P_{k+1/k} = \Phi_k P_{k/k} \Phi_k^T + \Gamma_k Q_k \Gamma_k^T$$
$$\hat{x}_{k+1/k+1} = \hat{x}_{k+1/k} + K_{k+1}(y_{k+1} - H_{k+1}\hat{x}_{k+1/k})$$
$$K_{k+1} = p_{k+1/k} H_{k+1}^T (H_{k+1} p_{k+1/k} H_{k+1}^T + R_{k+1})^{-1}$$
$$p_{k+1/k+1} = (I - K_{k+1} H_{k+1}) p_{k+1/k}$$

Since Server model system is not linear, extended Kalman filter is needed. And its framework as shown below:


$k = k+1, receiving Z_k$
（1） State Update
$\hat{X}_{k+1}^{-} = f(\hat{X}_k^{-}, U_k)$
（2） Compute probability matrix
$P_{k+1}^{-} = A_k P_k A_k^T + W_k Q_k W_k^T$
（1） Update incoming matrix
$K_k = \frac{P_k^- H_k^T}{H_k P_k^- H_k^T + R_k}$
（2） Update estimated value with vision information
$\hat{x}_k = \hat{x}_k^- + K_k(z_k - h(\hat{x}_k^-))$
（3）Update probability matrix
$P_k = (I - K_k H_k) P_k^-$
Action update
Vision update

3.2 Simulation result of Agent's location

After the realization of the filter simulation, robot's vision has been improved a lot. At the present time, Agent's location could be stabilized within about 10 periods. The filter errors basically stay below 1 centimeter after reaching the stable state. Some figures are given to show a series of simulation results.
3.21 A figure of tracking x axis in the instance of Agent's stillness (figure 7): the stochastically distributing dots in the figure are the vision values sending by the Server, while the middle curve is the result of location.

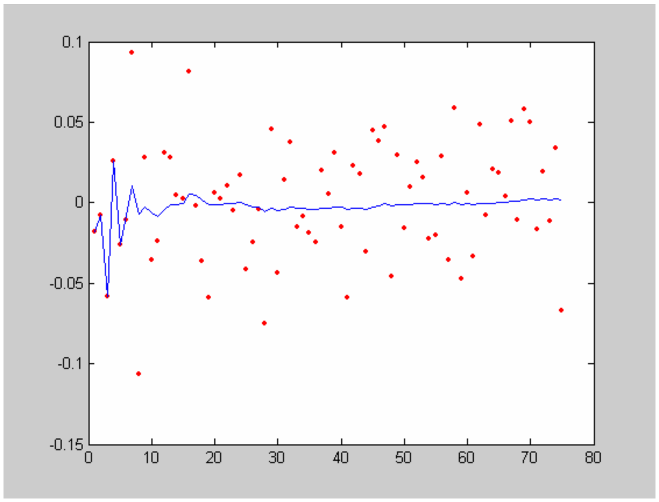


Figure 1　A picture of x axis in the instance of agent's stillness

3.2.2 A figure of still ball's location result on z axis with the help of the Agent's position. It could be concluded that the nearer the agent to the balls, the smaller the vision errors are.

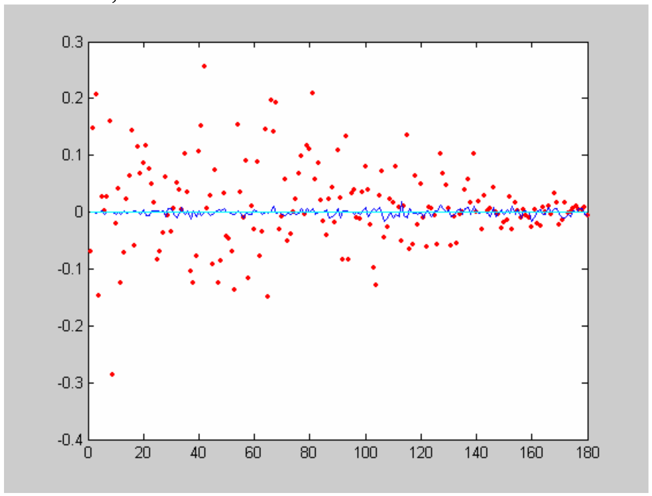


Figure 2　A figure of still ball's location result on z axis

3.2.3. A figure shows the moving ball's position on x axis. From the figure it can be concluded that the ball's position changes after being kicked for 3 times.

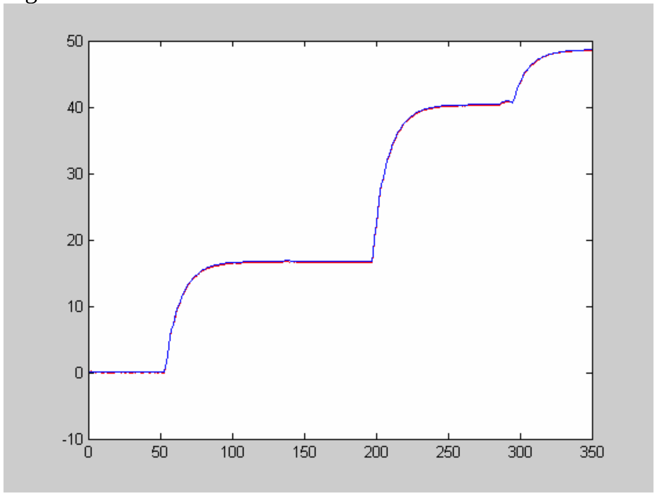


Figure 3　A figure of tracking moving ball on x axis

Reference

1.Text instead of manual. *Rcssserver3d-0.3*
2.*RoboCup Official Site, http://www.robocup.org/*
3.Welch G, and Bishop G, An Introduction To The Kalman Filter Department of Computer Science. *University of North Carolina at Chapel Hill* ,2001
4.Kalman,R.E.1960.A New Approach To Linear Filtering and Prediction Problems,*Transaction of ASME—Journal of Basic Engineering*.