Alzahra Soccer 3D Simulation Team
Team Description Paper for RoboCup 2010

Maryam Davari, Fatemeh Sadat Makki, Bita Namvar, Elmira Asadzadeh, and Seyedeh Atieh Hashemzadeh
http://robocup.alzahra.ac.ir

Alzahra University
Tehran, Iran
robocup@alzahra.ac.ir

Abstract. In this paper, Alzahra 3D soccer simulation team activities for RoboCup 2010 competitions is described. Since the number of agents in 3D soccer simulation is being increased each year, we use a path finding algorithm adapted from some Computational Geometry (CG) methods. Central Pattern Generator (CPG) provides in real time a reliable synchronization signal for periodic motions such as walking, respiration and flap. In most of the presented works the numerous CPG parameters are found using automatic techniques like Genetic Algorithms (GA).

Key words: Localization, Path Finding, Central Pattern Generator

1 Introduction

Alzahra university Robocup Team was formed in late summer 2009 in collaboration with students of mathematical sciences and computer engineering in technical college Alzahra university, with the aim of robotic science progress. In this short time our team could promote code (our code is based on Zigorat base code) and obtain useful skills for robot. Also, we optimized the skills using scientific methods and new techniques.

2 Team Architecture

The performances of humanoid robots depend on the agent's architecture. If this architecture is more accurate, the performance of the robot will be more like a human.

Here, we will briefly explain the architecture of our robots (Fig. 1).

2.1 Formation

At this part, the position and also the role of each agent (goal keeper and players) is specified with regard to the type of playing (offensive, defensive).

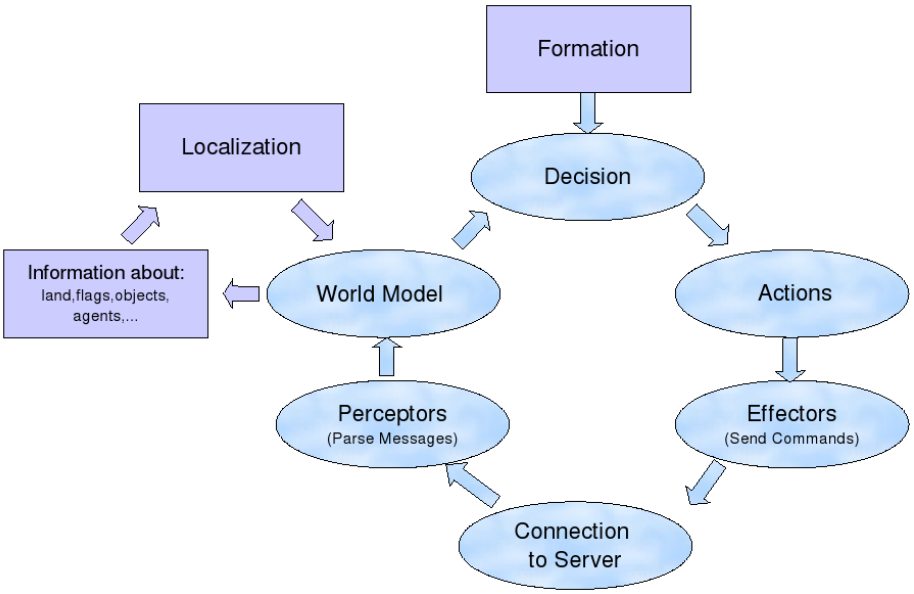

Fig. 1. Team Architecture

2.2 World Model

It includes all needed information for an agent such as the information about the field, ball and the other agents in the game. Therefore, a robot can figure out his own location and the state of other objects in the field. World model updates this information in each cycle with the messages it recieve from the simulation server. The agents can always use the information stored in the world model to make their decisions.

2.3 Decision

This part specifies the role of each agent using the Formation (in the beginning of the game) and World Model (during the game).

2.4 Actions

After the decision making process, actions of robots (like walking, kicking, turning and standing up) are sent to the server as commands.

2.5 Send Commands

It prepares a connection with the server and sends information to it.

2.6 Server Communication

This part is about communication with the server that includes network control, messages analysis and fixing the commands. This part is very important because it performs sending commands to the robot and also receiving sensor information from the robot.

2.7 Preceptors

In this part, the server sends necessary information about the game, players and other objects that were seen in the field.

3 Localization

Since robots receive important information from the server such as the position of the ball and the other agents as relative positions, localization is one of the most important and basic required abilities of a robot.

Considering the 120 degrees field of view of the robot in each dimension, this subject has become more complex compared to the previous versions of the simulator in which robots had a 360 degrees vision. Previously, robots were able to find their own location by seeing at least 3 flags with their 360 degrees vision; but with the new restricted vision sensor, this is not always possible.

Here, to remove this problem we use a new method in which the robot can localize itself using at least 2 flags. An overview of this method follows: First, we calculate the robot's height with this method:

$$\begin{aligned} Z_R = FootSize_Z + ShankSize_Z \times cos(\theta_{rlj5} + ThightSize_Z \times cos(\theta_{rlj4} - \theta_{rlj5}) \\ + (TorsoHeight + Head_Z/2) \times cos(\theta_{rlj3} - \theta_{rlj4} + \theta_{rlj5}) \end{aligned}$$

$$\begin{aligned} Z_L = FootSize_Z + ShankSize_Z \times cos(\theta_{llj5} + ThightSize_Z * cos(\theta_{llj4} - \theta_{llj5}) \\ + (TorsoHeight + Head_Z/2) * cos(\theta_{llj3} - \theta_{llj4} + \theta_{llj5}) \end{aligned}$$

Since the final height of the robot is relative to its base leg, we can use the value of the FRP sensor to determine the robot's base leg and use this height as the height of the robot. Now, since the height of the goal posts is different from the corners' height, we divided the visual states to 3 modes to simplify the flag vision control:

1. Right and left corners are visible
2. Up and down corners are visible
3. Left and right goal posts are visible

The difference between the first and the second states is the method of calculating the x coordinate in one state will result in the y coordinate in the other state. The third state is further divided into 2 substates:

– When

$$G2L_Y < y < G1L_Y$$

– When

$$(y < G2L_Y) \wedge (y > G1L_Y)$$

In these cases with 2 visible flags, robot's locus around each flag is available which is equal to a sphere with the flag in its center and the relative distance of the flag as its radius. Then we calculate the locus of the meet area of two obtained spheres and find the total radius. Since we have the robot's height, we can obtain x and y coordinates.

Until now, x, y and z coordinates of the robot are known but its angles relative to these 3 axes should be found. One simple procedure to find these 3 angles (θ, φ, ψ) is achieved by using hinges angles of robot and relative angle which is seen from one flag.

$$\theta = tan((Y_{flag} - y)/(X_{flag} - x)) - \theta_{flag} - \theta_{neck}$$

$$\phi_R = \theta_{rlj3} - |\theta_{rlj4} + \theta_{rlj5}|$$
$$\phi_L = \theta_{llj3} - |\theta_{llj4} + \theta_{llj5}|$$

$$\psi_R = |\theta_{rlj2}| + |\theta_{rlj6}|$$
$$\psi_L = |\theta_{llj2}| + |\theta_{llj6}|$$

Final φ, ψ are related to the base leg which is determined with the procedure mentioned before.

As the previous discussion, this solution is applicable only with two flags. If the robot can't see 2 flags, it can find them by rotating its head.

4 Path Finding

One of the important problems concerning mobile robots is reaching a target without hitting obstacles (players and other objects in the field). Normally, Artificial Intelligence (AI) is used to help robots avoid obstacles, but another possibility is to use Computational Geometry (CG) methods. In our team, we are working on a CG based approach to the path finding problem.

In the proposed method, the arms and feet of the standing robots and the arms, feet and the head of the other robots are observed. Then, their positions are determined and considerd as vertices of a convex polygon. Afterwards, the ball polygon is drawn and the polygons of the obstacles in the field are calculated.

In (Fig. 2), polygon O_1 represents an agent laying down on the ground and polygons O_2 and O_3 represent standing agents. Nevertheless, as the agent is not normally able to see all of the agents' points at the same time, it can draw the largest possible polygon for them and find out the *configuration-space obstacle*

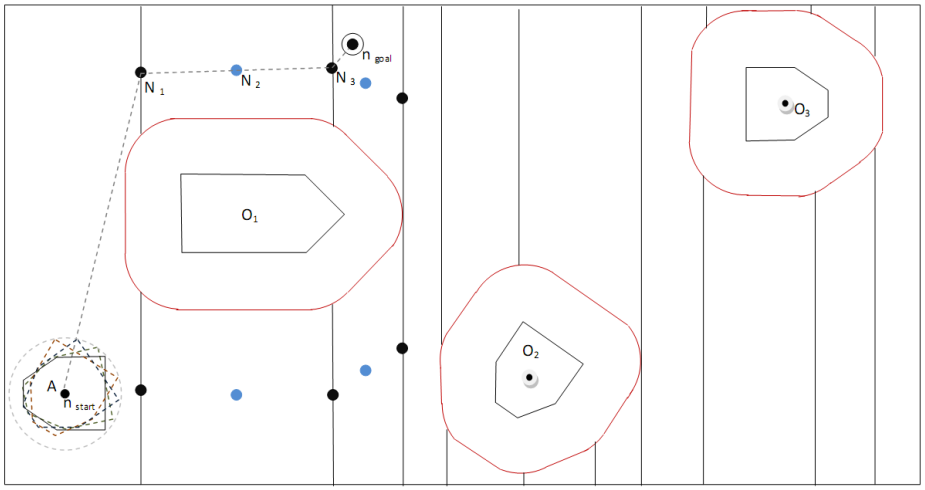

Fig. 2. Path finding

C_p by seeing the arms or feet of the standing agents and viewing at least 3 of the specified points for the laying down agents.

Needless to say, it is not easy to formulate this space because robot shall check, all the time, its own edges and vertices and other objects to avoid hitting obstacles. Taking into account the largest possible polygon for a robot and by analyzing the server's data in every n cycles (here n is set at 250, i.e. every 5 seconds), it is possible to reduce the complexity of the algorithm. However, objects may have considerable changes in position due to this 5 seconds, so we have to estimate positions of the objects in this 5 seconds.

In this stage, we call the robot's polygon "A" and the polygons of each objects "O_i" and compute the Minkowski Sum [1, 5] of A and O_i (this help the robot to prevent hitting other objects). So we have:

$$P_i = A \oplus O_i \qquad 1 \leqslant i \leqslant m$$

Where

P_i is the Minkowski sum of A and O_i
m is the number of agents that was seen

Then we subtract configuration-space obstacle Cp from free configuration space M:

$$C_{free} = M \setminus \bigcup_{i=1}^{m} P_i$$

We call it "free configuration". Now with the conventional triangulation method (Trapezoid Map) [1], we divide free configuration. Then, to arrive to

the desired node (with the minimum possibility in hitting an obstacle), we use conventional shortest path algorithms where nodes are agent's position (N_start), goal's position (N_goal), centers of trapezoids ($N_i s.t. i = 2k : l < k < m/2$) and also centers of parallel edges which are perpendicular to X axis of the field ($N_i s.t. i = 2k + 1 : l < k < m/2$). In this case all of the possible paths for reaching the agent to the goal is already defined; and with the shortest path algorithms, the best and the shortest path of agent's movement will be specified with the minimum possibility of hitting with obstacles.

One of the shortcomings of this method is that the question can be solved in two cases only:

1. When the robot is supposed to be a point, while it is a shaped object
2. When the orientation of the robot remains unchanged during the movement

Obviously, the first case is not favorable. But the second case is problematic too: the robot uses consecutive Walk and Turn actions to reach a given point and so, its orientation changes during the movement. To solve this problem, three solutions are proposed as below:

1. Using Straight Walk and Side Walk. We ignore it because our GotoPosition actions are not implemented this way.
2. Assuming that the robot is facing towards the next point (next N_i), we will keep the robot's speed constant and predict its path in the next 5 seconds. Then, we estimate the robot's position after this 5 seconds using the distance between N_is and the angle between the last N_i and the next one is calculated. Now, we give the final direction and position to the robot. Accordingly, the robot will stand in the desired direction.
3. As the robot constantly keeps changing its direction, all possible directions the robot may go in are taken into account. In fact, the robot is taken as a circle. Now, we can continue as above.

However, because of the reduction in system's speed we are simplifying the complexity of the system.

5 Walking Controller

In the initial steps we studied two type of working: static and dynamic The results of our studies are as follows:

Static walking means a stable motion in which the projection of the center of mass should be inside of a stable area in the field that cause the robot to walk slower. Dynamic walking doesn't have the limitations of static walking. In this type of walking, the agent balance depends on the whole dynamic system (speed and inertia) that cause the agent to walk faster.

We decided to use dynamic walking. Agent control in this method is divided into three parts: the first part is the control strategy, the second part is planning or other middle components. The role of these two parts is to produce reference

signals for the motion of agent's joints and the third part is tracking and stabilization using different common controls. It includes controllers that were used for controling agent arms [6].

In agent walking, the first two parts are different so we discuss the relative methods into the two parts and the results are as follows:

1. methods based on motion trajectory [3].
2. mental methods.
3. methods based on the CPG [2].

The first two methods require accurate mathematical models and the system should be defined accurately. Since achieving this accuracy is difficult and expensive (because of the simulation steps), a lot of analysis & testing is needed and each agent should be defined separately with the complete details.

So we preferred to use the third method (CPG) that doesn't need any exact dynamic information from the agent and the environment for building a mathematical model. Neural oscillators, also known as Spinal Pattern Generators (SPG) are neural networks that can produce complex muscular activation patterns [9].

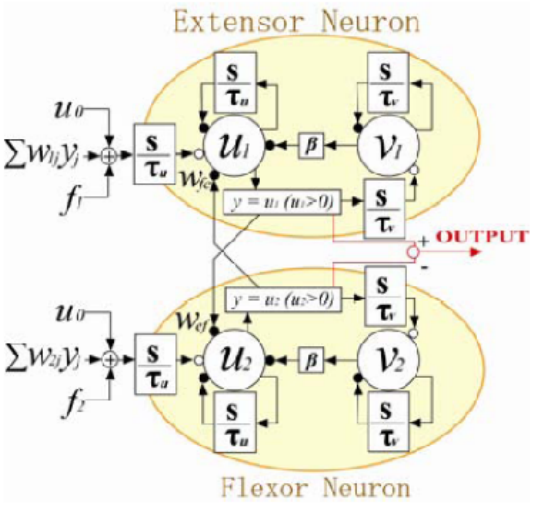

Fig. 3. The architecture of CPG [7]

Proposing a suitable oscillator model is the most important part of the CPG design. We used Matsuoka Oscillator model, the equations of which is as follows [8]:

$$\tau_u \dot{u}_i - u_i - \beta v_i + \sum_{j=1}^{n} \omega_{ij} y_i + u_0 + f_i$$

$$y_i = \max(0, u_i)$$

$$\tau_v \dot{v_i} = -v_i + y_i$$

u_i is the potential of the i^{th} neuron.
v_i is the variable that represents the degree of adaptation.
u_0 is the external with a constant rate.
f_i is the feedback signal from a sensory input.
τ_u, β, τ_v are parameters that specify the time constant for adaptation.
ω_{ij} is the weight between neurons.
n is the number of neurons.

The CPG network for bipedal walk is designed to have an oscillator with two neurons (Fig. 3), which are an *extensor* neuron and a *flexor* neuron, on each joint [7].

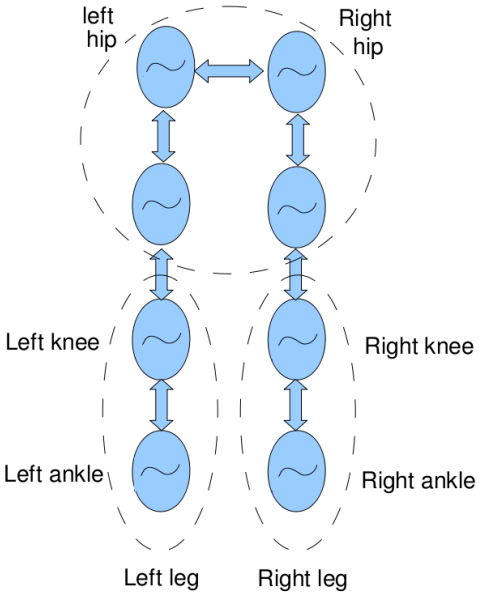

Fig. 4. The architecture of a CPG network

We just use 10 degrees of 22 degrees of freedom for walking of the robot (12 degrees of freedom of robot are related to its neck, leg and hands that don't have any effect on the robot's walking). Since each oscillator is used for the control of one degree of freedom, we use 10 oscillators for the constitution of the neuron oscillator network with 10 oscillators considering symmetry between the left side and the right side of the robot (Fig. 4).

Considering symmetrical robot's motion, the total number of connections between oscillators will decrease; and as a result we will have simpler and faster optimization, but we prefer to use more connections between oscillators with respect to the symmetry of left and right sides of the robot. It will make the robot more resistant when faced to disturbances. Using genetic algorithm, we try to optimize the CPG parameters. Here is an overview of our researches:

Because of the symmetry, we are just considering the parameters of the right side of the body and the other half of the body is completely same as the other side, just with a different time shift. Since the total number of the parameters are 22, actually each chromosome has 22 genes.

We will use the usual coding scheme for the genetic algorithm, which means using binary coding for the chromosomes. The fitness function is the distance that the robot travels from the start point till the end point (in which the balanced robot crashes and it falls); and we select the best chromosomes for reproduction with the selection operator (using binary tournament selection algorithm).

Two-point-crossover is used for the crossover operation and the probability of mutation is 0.5% [4].

References

1. De Berg, M., Cheong, O., Van Kreveld, M., Overmars, M.: Computational geometry: algorithms and applications. Springer-Verlag New York Inc (2008)
2. Geyer, H.: A model of biped walking based on muscle reflexes that encode principles of legged mechanics. IEEE Humanoids Workshop "Modeling, Simulation and Optimization of Bipedal Walking", Zurich, Germany (2009)
3. Ijspeert, A., Crespi, A.: Online trajectory generation in an amphibious snake robot using a lamprey-like central pattern generator model. In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2007). Citeseer (2007)
4. Inada, H., Ishii, K.: Bipedal walk using a central pattern generator. In: international congress Series. vol. 1269, pp. 185–188. Elsevier (2004)
5. Joseph, O.: Computational geometry in C (1998)
6. Katić, D., Vukobratović, M.: Survey of intelligent control techniques for humanoid robots. Journal of Intelligent and Robotic Systems 37(2), 117–141 (2003)
7. Matsuo, T., Ishii, K.: The Motion Control of Snake-like Robot Using CPG
8. Matsuoka, K.: Sustained oscillations generated by mutually inhibiting neurons with adaptation. Biological Cybernetics 52(6), 367–376 (1985)
9. Minassian, K., Jilge, B., Rattay, F., Pinter, M., Binder, H., Gerstenbrand, F., Dimitrijevic, M.: Stepping-like movements in humans with complete spinal cord injury induced by epidural stimulation of the lumbar cord: electromyographic study of compound muscle action potentials. Spinal Cord 42(7), 401–416 (2004)