To appear in *Proceedings of the RoboCup International Symposium 2011* (RoboCup 2011),
Istanbul, Turkey, July 2011.

WrightEagle and UT Austin Villa: RoboCup 2011 Simulation League Champions

Aijun Bai[†], Xiaoping Chen[†],
Patrick MacAlpine[‡], Daniel Urieli[‡], Samuel Barrett[‡], and Peter Stone[‡]

[†]Department of Computer Science, University of Science and Technology of China
xpchen@ustc.edu.cn
[‡]Department of Computer Science, The University of Texas at Austin
pstone@cs.utexas.edu

Abstract. The RoboCup simulation league is traditionally the league with the largest number of teams participating, both at the international competitions and worldwide. 2011 was no exception, with a total of 39 teams entering the 2D and 3D simulation competitions. This paper presents the champions of the competitions, WrightEagle from the University of Science and Technology of China in the 2D competition, and UT Austin Villa from the University of Texas at Austin in the 3D competition.

1 Introduction

The RoboCup simulation league has always been an important part of the RoboCup initiative. The distinguishing feature of the league is that the soccer matches are run in software, with no physical robot involved. As such, some of the real-world challenges that arise in the physical robot leagues can be abstracted away, such as image processing and wear and tear on physical gears. In exchange, it becomes possible in the simulation league to explore strategies with larger teams or robots (up to full 11 vs. 11 games), and to leverage the possibility of automating large numbers of games, for example for the purpose of machine learning or to establish statistically significant effects of strategy changes.

In 2011, as in recent past years, there were two separate simulation competitions held at RoboCup. In both cases, a *server* simulates the world including the dynamics and kinematics of the players and ball. Participants develop a fully autonomous team of client *agents* that each interact separately with the server by i) receiving sensations representing the view from its current location; ii) deciding what actions to execute; and iii) sending the actions back to the server for execution in the simulated world. The sensations are abstract and noisy in that they indicate the approximate distance and angle to objects (players, ball, and field markings) that are in the direction that the agent is currently looking. The server proceeds in real time, without waiting for agent processes to send their actions: it is up to each agent to manage its deliberation time so as to keep up with the server's pace. Furthermore, each agent must be controlled by a completely separate process, with no file sharing or inter-process communication (simulated low-bandwidth verbal communication is available via the server).

Though similar in all of the above respects, the 2D and 3D simulators also differ in some important ways. As their names suggest, the 2D simulator models only the (x, y) positions of objects, while the 3D simulator includes the third dimension. In the 2D simulator, the players and the ball are modeled as circles. In addition to its (x, y) location, each player has a direction that its body is facing, which affects the directions it can move; and a separate direction in which it is looking, which affects its sensations. Actions are abstract commands such as turning the body or neck by a specified angle, dashing forwards or backwards with a specified power, kicking at a specified angle with a specified power (when the ball is near), or slide tackling in a given direction. Teams consist of 11 players, including a goalie with special capabilities such as catching the ball when it is near. In particular, the 2D simulator does not model the motion of any particular physical robot, but does capture realistic team-level strategic interactions.

In contrast, the 3D simulator implements a physically realistic world model and an action interface that is reflective of that experienced by real robots. The simulator uses the Open Dynamics Engine[1] (ODE) library for its realistic simulation of rigid body dynamics with collision detection and friction. ODE also provides support for the modeling of advanced motorized hinge joints used in the humanoid agents. The agents are modeled after the Aldebaran Nao robot,[2] which has a height of about 57 cm, and a mass of 4.5 kg. The agents interact with the simulator by sending actuation commands to each of the robot's joints. Each robot has 22 degrees of freedom: six in each leg, four in each arm, and two in the neck. In order to monitor and control its hinge joints, an agent is equipped with joint perceptors and effectors. Joint perceptors provide the agent with noise-free angular measurements every simulation cycle (20 ms), while joint effectors allow the agent to specify the direction and speed (torque) in which to move a joint. Although there is no intentional noise in actuation, there is slight actuation noise that results from approximations in the physics engine and the need to constrain computations to be performed in real-time. The 3D simulator thus presents motion challenges similar to that of the humanoid soccer leagues, especially the standard platform league (SPL) which uses physical Nao robots. In particular, it is a non-trivial challenge to enable the robots to walk and kick without falling over. The 3D simulation league teams consist of 9 (homogeneous) players, rather than 4 as in the SPL.

In 2011, the 2D and 3D simulation competitions included 17 and 22 teams, respectively, from around the world. This paper briefly describes and compares the two champion teams, WrightEagle from USTC in the 2D simulation league, and UT Austin Villa from UT Austin in the 3D simulation league.

The remainder of the paper is organized as follows. Section 2 introduces the WrightEagle team, particularly emphasizing its heuristic approximate online planning for large-scale and sparse-reward MDPs. Section 3 introduces the UT

[1] http://www.ode.org/

[2] http://www.aldebaran-robotics.com/eng/

Austin Villa Team, focussing especially on its learning-based walk. Section 4 concludes.

2 WrightEagle: 2D Simulation League Champions

This section describes the RoboCup 2011 2D competition champion team, WrightEagle. First the overall system structure, based on hierarchical MDPs, is introduced, which then leads into an overview of the team's main research focus: heuristic approximate planning in such MDPs.

2.1 System Structure of WrightEagle

The team WrightEagle, including the latest version, has been developed based on the Markov decision processes (MDPs) framework [1], with the MAXQ hierarchical structure [2] and heuristic approximate online planning techniques strengthened particularly in the past year.

MDP Framework Formally, an MDP is defined as a 4-tuple $\langle S, A, T, R \rangle$, where

- S is the set of possible states of the environment,
- A is the set of available actions of the agent,
- T is the transition function with $T(s'|s, a)$ denoting the next state probability distribution by performing action a in state s,
- R is the reward function with $R(s, a)$ denoting the immediate reward received by the agent after performing action a in state s.

A set of standard algorithms exist for solving MDP problems, including linear programming, value iteration, and policy iteration [1]. However, in large-scale and sparse-reward domains such as the 2D simulator, solving the MDP problem directly is to some degree intractable. In WrightEagle, some techniques including the MAXQ hierarchy and heuristic approximate online planning were applied to overcome this difficulty.

MAXQ Hierarchical Decomposition The MAXQ framework decomposes a given MDP into a hierarchy of sub MDPs (known as subtasks or behaviors) $\{M_0, M_1, \cdots, M_n\}$, where M_0 is the root subtask which means solving M_0 solves the entire original MDP [2]. Each of the subtasks is defined with a subgoal, and it terminates when its subgoal is achieved.

Over the hierarchical structure, a *hierarchical policy* π is defined as a set of policies for each of the subtasks $\pi = \{\pi_0, \pi_1, \cdots, \pi_n\}$. Each subtask policy π_i is a mapping from states to actions $\pi_i : S_i \rightarrow A_i$, where S_i is the set of relevant states of subtask M_i and A_i is the set of available primitive actions or composite actions (i.e. its subtasks) of that subtask.

A hierarchical policy is *recursively optimal* if the local policy for each subtask is optimal given that all its subtasks are in turn recursively optimal [3].

Dietterich [4] has shown that a recursively optimal policy can be found by computing its recursively optimal V function, which satisfies:

$$Q(i, s, a) = V(a, s) + C(i, s, a) \ , \tag{1}$$

$$V(i, s) = \begin{cases} \max_a Q(i, s, a) & \text{if } i \text{ is composite} \\ R(s, i) & \text{otherwise} \end{cases} \ , \tag{2}$$

and

$$C(i, s, a) = \sum_{s', N} \gamma^N T(s', N|s, a) V(i, s') \ , \tag{3}$$

where $T(s', N|s, a)$ is the probability that the system terminates in state s' with a number of steps N after action a is invoked.

To solve $V(i, s)$, a complete search of all paths through the MAXQ hierarchy starting from subtask M_i and ending at primitive actions should be performed, in a depth-first search way. If $V(i, s)$ for subtask M_i is known, then its recursively optimal policy can be generated from $\pi_i(s) = \text{argmax}_a Q(i, s, a)$.

Hierarchical Structure of WrightEagle The main hierarchical structure of WrightEagle is shown in Fig. 1. The two subtasks of root task WrightEagle are Attack and Defense. Attack has its subtasks including Shoot, Dribble, Pass, Position and Intercept, while Defense has its subtasks including Block, Trap, Mark and Formation. In WrightEagle, these subtasks are called *behaviors*.

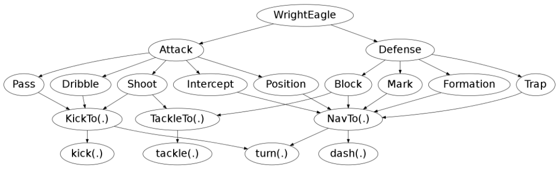

Fig. 1: MAXQ task graph for WrightEagle

Behaviors share their same subtasks, including KickTo, TackleTo and NavTo. Note that, the parentheses after these subtasks in Fig. 1 indicate that they are parameterized. In turn, these subtasks also share their same subtasks consisting of kick, turn, tackle and dash, which are parameterized primitive actions originally defined by the 2D domain.

2.2 Heuristic Approximate Online Planning

Theoretically, the full recursively optimal V function can be solved by some standard algorithms. But in practice this is intractable in 2D domain, because:

1. the state and action space is huge, even if discretization methods are used;
2. the reward function is very sparse, as the ball is usually running for thousands of cycles without any goals being scored;
3. the environment is unpredictable as teammates and opponents are all autonomous agents having the ability to make their own decisions, which prevents the original MDP problem to be solved completely *offline*.

For these reasons, the WrightEagle team focuses on approximate but not exact solutions. Our method is to compute the recursively optimal V function by heuristic *online* planning techniques under the consideration of simplification.

Current State Estimation To model the RoboCup 2D domain as an MDP which assumes that the environment's state is fully observable, the agent must overcome the difficulty that it can only receive local and noisy observations, to obtain a precise enough estimation of the environment's current state. In our team, the agent estimates the current state from its *belief* [5]. A belief b is a probability distribution over state space, with $b(s)$ denoting the probability that the environment is actually in state s. We assume conditional independence between individual objects, then the belief $b(\boldsymbol{s})$ can be expressed as

$$b(\boldsymbol{s}) = \prod_{0 \le i \le 22} b_i(\boldsymbol{s}[i]), \tag{4}$$

where $\boldsymbol{s}$ is the full state vector, $\boldsymbol{s}[i]$ is the partial state vector for object i, and $b_i(\boldsymbol{s}[i])$ is the marginal distribution for $\boldsymbol{s}[i]$. A set of m_i weighted samples (also known as particles) are then used to approximate b_i as:

$$b_i(\boldsymbol{s}[i]) \approx \{\boldsymbol{x}_{ij}, w_{ij}\}_{j=1 \ldots m_i}, \tag{5}$$

where $\boldsymbol{x}_{ij}$ is a sampled state for object i, and w_{ij} represents the approximated probability that object i is in state $\boldsymbol{x}_{ij}$ (obviously $\sum_{1 \le j \le m_i} w_{ij} = 1$).

In the beginning of each step, these samples are updated by Monte Carlo procedures using the domain's *motion model* and *sensor model* [6]. It is worth noting that, the agent can not observe the actions performed by other players, so it always assume that they will do a random kick if the ball is kickable for them, or a random walk otherwise. Finally, the environment's current state $\boldsymbol{s}$ is estimated as:

$$\boldsymbol{s}[i] = \sum_{1 \le j \le m_i} w_{ij} \boldsymbol{x}_{ij}. \tag{6}$$

Transition Model Simplification Recall that, computing Equations 3, 1, and 2 recursively can find the recursive optimal policy over the MAXQ hierarchy. However, to completely represent either the transition function T or the completion function C in the 2D domain is intractable. Some approximate methods are applied to overcome this difficulty, as described next.

In WrightEagle, based on some pre-defined rules, the entire space of possible state-steps pairs (s', N) for each subtask is dynamically split into two classes: the success class and the failure class, denoted as $sucess(s, a)$ and $failure(s, a)$ respectively.

After splitting, the expected state-steps pair (called pre-state in WrightEagle) of each class is used to represent it, which can be calculated either by Monte Carlo methods or approximately theoretical analysis, denoted as s_s and s_f respectively. Then the completion function can be approximately represented as

$$C(i, s, a) \approx pV(i, s_s) + (1 - p)V(i, s_f) \ , \tag{7}$$

where $p = \sum_{(s',N) \in sucess(s,a)} T(s',N|s,a)$.

To efficiently calculate p in Equation 7, a series of approximate methods were developed in WrightEagle. They are classified into two groups: subjective probability based on some heuristic functions, and objective probability based on some statistical models.

Heuristic Approximation of Value Function Some heuristic evaluation functions (denoted as $H(i, s_s|s,a)$ and $H(i, s_f|s,a)$ respectively) were developed to approximate $V(i, s_s)$ and $V(i, s_f)$, because:

1. the pre-states s_s and s_f are hard to estimate due to the unpredictable property of the environment, especially when they are far in the future;
2. completely recursively computing costs too much to satisfy the real-time constraints in the 2D domain.

For each subtask, different H functions were developed, according to different subgoals. For low level subtasks and primitive actions, the reward functions $R(s,a)$ are too sparse to be used directly. As a substitute, some pseudo-reward functions are developed in WrightEagle. Take the KickTo subtask for an example, the maximum speed that the ball can be kicked in a range of given cycles plays a key role in the local heuristic function.

Heuristic Search in Action Space By now, we have almost solved the MAXQ hierarchy used in WrightEagle by heuristic approximate online planning techniques, but there's still one difficulty remaining in Equation 2: as for the parameterized actions (including KickTo, kick, etc.), the action space is too huge to be searched directly by some brute force algorithms.

Some heuristic search methods are introduced to deal with this issue, e.g. an A* search algorithm with some special pruning strategy is used in the NavTo subtask when searching the action space of dash and turn, a hill-climbing method is used in the Pass subtask when searching the action space of KickTo, etc [7,8].

Particularly, for the Defense subtask and its subtasks, which are more involved with cooperation between agents, theoretical analysis is more difficult. Some work has been done on this based on the decentralized partially observable Markov decision processes (DEC-POMDPs) [9,10].

2.3 RoboCup 2011 Soccer 2D Simulation League Results

In RoboCup 2011, the WrightEagle team won the champion with no lost games, and achieved an average goal difference of 12.33 in total 12 games.[3] The Helios team, a united team from Fukuoka University, Osaka Prefecture University, and the National Institute of Advanced Industrial Science and Technology of Japan, won second place, and the MarliK team from University of Guilan of Iran won third place.

[3] The detailed competition results can be found at: `http://sourceforge.net/apps/mediawiki/sserver/index.php?title=RoboCup2011/Competition`

3 UT Austin Villa: 3D Simulation League Champions

This section describes the RoboCup 2011 3D competition champion team, UT Austin Villa, with particular emphasis on the main key to the team's success: an optimized omnidirectional walk engine. Further details about the team, including an inverse kinematics based kicking architecture and a dynamic role assignment and positioning system, can be found in [11].

3.1 Agent Architecture

The UT Austin Villa agent receives visual sensory information from the environment which provides distances and angles to different objects on the field. It is relatively straightforward to build a world model by converting this information about the objects into Cartesian coordinates. This of course requires the robot to be able to localize itself for which the agent uses a particle filter. In addition to the vision perceptor, the agent also uses its accelerometer readings to determine if it has fallen and employs its auditory channels for communication.

Once a world model is built, the agent's control module is invoked. Figure 2 provides a schematic view of the control architecture of the UT Austin Villa humanoid soccer agent.

At the lowest level, the humanoid is controlled by specifying torques to each of its joints. This is implemented through PID controllers for each joint, which take as input the desired angle of the joint and compute the appropriate torque. Further, the agent uses routines describing inverse kinematics for the arms and legs. Given a target position and pose for the foot or the hand, the inverse kinematics routine uses trigonometry to calculate the angles for the different joints along the arm or the leg to achieve the specified target, if at all possible.

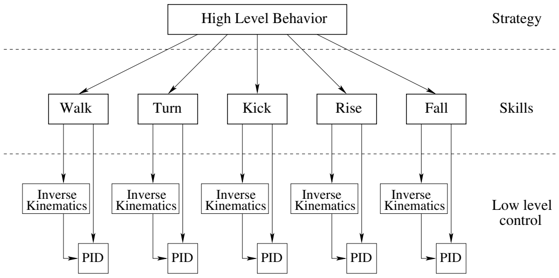

Fig. 2: Schematic view of UT Austin Villa agent control architecture.

The PID control and inverse kinematics routines are used as primitives to describe the agent's skills. In order to determine the appropriate joint angle sequences for walking and turning, the agent utilizes an omnidirectional walk engine which is described in subsection 3.2. When invoking the kicking skill, the agent uses inverse kinematics to control the kicking foot such that it follows an appropriate trajectory through the ball. This trajectory is defined by set waypoints, ascertained through machine learning, relative to the ball along a cubic Hermite spline. Two other useful skills for the robot are falling (for instance, by the goalie to block a ball) and rising from a fallen position. Both falling and rising are accomplished through a programmed sequence of poses and specified joint angles.

Because the team's emphasis was mainly on learning robust and stable low-level skills, the high-level strategy to coordinate the skills of the individual agents is relatively straightforward. The player closest to the ball is instructed to go to it while other field player agents dynamically choose target positions on the field based on predefined formations that are dependent on the current state of the game. For example, if a teammate is dribbling the ball, one agent positions itself slightly behind the dribbler so that it is ready to continue with the ball if its teammate falls over. The goalie is instructed to stand a little in front of its goal and, using a Kalman filter to track the ball, attempts to dive and stop the ball if it comes near.

3.2 Omnidirectional Walk Engine and Optimization

The primary key to Austin Villa's success in the 2011 RoboCup 3D simulation competition was its development and optimization of a stable and robust fully omnidirectional walk. The team used an omnidirectional walk engine based on the research performed by Graf et al. [12]. The main advantage of an omnidirectional walk is that it allows the robot to request continuous velocities in the forward, side, and turn directions, permitting it to approach its destination more quickly. In addition, the robustness of this engine allowed the robots to quickly change directions, adapting to the changing situations encountered during soccer games.

Walk Engine Implementation The walk engine uses a simple set of sinusoidal functions to create the motions of the limbs with limited feedback control. The walk engine processes desired walk velocities given as input, chooses destinations for the feet and torso, and then inverse kinematics are used to determine the joint positions required. Finally, PID controllers for each joint convert these positions into commands that are sent to the joints.

The walk first selects a trajectory for the torso to follow, and then determines where the feet should be with respect to the torso location. The trajectory is chosen using a double linear inverted pendulum, where the center of mass is swinging over the stance foot. In addition, as in Graf et al.'s work [12], the simplifying assumption that there is no double support phase is used, so that the velocities and positions of the center of mass must match when switching between the inverted pendulums formed by the respective stance feet.

The walk engine is parameterized using more than 40 parameters, ranging from intuitive quantities, like the step size and height, to less intuitive quantities like the maximum acceptable center of mass error. These parameters are initialized based on an understanding of the system and also testing them out on an actual Nao robot. This initialization resulted in a stable walk. However, the walk was extremely slow compared to speeds required during a competition. We refer to the agent that uses this walk as the *Initial* agent.

Walk Engine Parameter Optimization The slow speed of the *Initial* agent calls for using machine learning to obtain better walk parameter values. Parameters are optimized using the CMA-ES algorithm [13], which has been successfully applied in [14]. CMA-ES is a policy search algorithm that successively generates and evaluates sets of candidates. Once CMA-ES generates a group of candidates, each candidate is evaluated with respect to a *fitness* measure. When all the candidates in the group are evaluated, the next set of candidates is generated by sampling with probability that is biased towards directions of previously successful search steps.

As optimizing 40 real-valued parameters, can be impractical, a carefully chosen subset of 14 parameters was selected for optimization while keeping all the other parameters fixed. The chosen parameters are those that have the highest potential impact on the speed and stability of the robot, for instance: the maximum step sizes, rotation, and height; the robot's center of mass height, shift amount, and default position; the fraction of time a leg is on the ground and the time allocated for one step phase; the step size PID controller; center of mass normal error and maximum acceptable errors; and the robot's forward offset.

Similarly to a conclusion from [14], Austin Villa has found that optimization works better when the robot's fitness measure is its performance on tasks that are *executed during a real game*. This stands in contrast to evaluating it on a general task such as the speed walking straight. Therefore, the robot's in-game behavior is broken down into a set of smaller tasks, and the parameters for each one of these tasks is sequentially optimized. When optimizing for a specific task, the performance of the robot on the task is used as CMA-ES's fitness value for the current candidate parameter set values.[4]

In order to simulate common situations encountered in gameplay, the walk engine parameters for a goToTarget subtask are optimized. This consists of an obstacle course in which the agent tries to navigate to a variety of target positions on the field. The goToTarget optimization includes quick changes of target/direction for focusing on the reaction speed of the agent as well as holding targets for longer durations to improve the straight line speed of the agent. Additionally the agent is instructed to stop at different times during the optimization to ensure that is stable and doesn't fall over when doing so. In order to encourage both quick turning behavior and a fast forward walk, the agent always walks and turns toward its designated target at the same time. This allows for the agent to swiftly adjust and switch its orientation to face its target, thereby emphasizing the amount of time during the optimization that it is walking forward. Optimizing the walk engine parameters in this way resulted in a significant improvement in performance with the *GoToTarget* agent able to quickly turn and walk in any direction without falling over. This improvement also showed itself in actual game performance as when the *GoToTarget* agent played 100 games

[4] Videos of the agent performing optimization tasks can be found online at `http://www.cs.utexas.edu/~AustinVilla/sim/3dsimulation/AustinVilla3DSimulationFiles/2011/html/walk.html`

against the *Initial* agent, the *GoToTarget* agent won on average by 8.82 goals with a standard error of .11.

To further improve the forward speed of the agent, a second walk engine parameter set was optimized for walking straight forward. This was accomplished by running the goToTarget subtask optimization again, but this time the *goToTarget* parameter set was fixed while a new parameter set, called the *sprint* parameter set, was learned. The *sprint* parameter set is used when the agent's orientation is within 15° of its target. By learning the *sprint* parameter set in conjunction with the *goToTarget* parameter set, the new *Sprint* agent was stable switching between the two parameter sets and also increased the agent's speed from .64 m/s to .71 m/s as timed when walking forward for ten seconds after starting from a standstill.

Although adding the *goToTarget* and *sprint* walk engine parameter sets improved the stability, speed, and game performance of the agent, the agent was still a bit slow when positioning to dribble the ball. This slowness makes sense because the goToTarget subtask optimization emphasizes quick turns and forward walking speed while positioning around the ball involves more side-stepping to circle the ball. To account for this discrepancy, the agent learned a third parameter set called the *positioning* parameter set. To learn this new parameter set a driveBallToGoal2 optimization was created in which the agent is evaluated on how far it is able to dribble the ball over 15 seconds when starting from a variety of positions and orientations from the ball. The *positioning* parameter set is used when the agent is within .8 meters of the ball. Both *goToTarget* and *sprint* parameter sets are fixed and the optimization naturally includes transitions between all three parameter sets, which constrains them to be compatible with each other. Adding both the *positioning* and *sprint* parameter sets further improved the agent's performance such that it, the *Final* agent, was able to beat the *GoToTarget* agent by an average of .24 goals with a standard error of .08 across 100 games. A summary of the progression in optimizing the three different walk parameter sets can be seen in Figure 3.

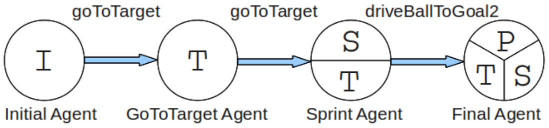

Fig. 3: UT Austin Villa walk parameter optimization progression. Circles represent the set(s) of parameters used by each agent during the optimization progression while the arrows and associated labels above them indicate the optimization tasks used in learning. Parameter sets are the following: I = *initial*, T = *goToTarget*, S = *sprint*, P = *positioning*.

3.3 RoboCup 2011 Soccer 3D Simulation League Results

The UT Austin Villa team won the 2011 RoboCup 3D simulation competition in convincing fashion by winning all 24 matches it played, scoring 136 goals and conceding none. The CIT3D team, from Changzhou Institute of Technology of China, came in second place while the Apollo3D team, from Nanjing University of Posts and Telecommunications of China, finished third. The success UT Austin Villa experienced during the competition was no fluke as when playing 100 games against each of the other 21 teams' released binaries from the competition, the

UT Austin Villa team won by at least an average goal difference of 1.45 against every team. Furthermore, of these 2100 games played, UT Austin Villa won all but 21 of them which ended in ties (no losses). The few ties were all against three of the better teams: Apollo3D, Bold Hearts, and RoboCanes. We can therefore conclude that UT Austin Villa was the rightful champion of the competition.

While there were multiple factors and components that contributed to the success of the UT Austin Villa team in winning the competition, its omnidirectional walk was the one which proved to be the most crucial. When switching out the omnidirectional walk developed for the 2011 competition for a fixed skill based walk used in the 2010 competition, and described in [14], the team did not fare nearly as well. The agent with the previous year's walk had a negative average goal differential against nine of the teams from the 2011 competition, suggesting a probable tenth place finish. Also this agent lost to our *Final* agent by an average of 6.32 goals across 100 games with a standard error of .13. One factor that did not come into play in UT Austin Villa winning the competition, however, was its goalie. The team's walk allowed it to dominate possession of the ball and keep it away from the opposing team such that the opponent never had a chance to shoot on the goal, and thus the goalie never touched the ball during the course of gameplay.

4 Conclusion

This paper introduced the champions of the RoboCup 2011 simulation leagues.

First, we described the MAXQ hierarchical structure of WrightEagle, and the online planning method combining with heuristic and appropriate techniques over this hierarchy. Based on this effort, The WrightEagle team has a very flexible and adaptive strategy, particularly for unfamiliar teams. It has won 3 championships and 4 runner-ups in the past 7 years of RoboCup competitions.[5]

Second, we described the learning-based omnidirectional walk of UT Austin Villa, and the series of fitness functions that were employed during learning. The resulting walk was quick, agile, and stable enough to dribble around most of the other teams in the competition. 2011 was the first victory for UT Austin Villa in the 3D simulation league.[6]

Acknowledgments

The authors would like to thank the additional contributing members of WrightEagle (Feng Wu, Zongzhang Zhang, Haochong Zhang and Guanghui Lu) and UT Austin Villa (Shivaram Kalyanakrishnan, Frank Barrera, Nick Collins, Adrian Lopez-Mobilia, Art Richards, Nicu Stiurca, and Victor Vu).

[5] More information about the WrightEagle team can be found at the team's website: `http://www.wrighteagle.org/2d`

[6] More information about the UT Austin Villa team, as well as video highlights from the competition, can be found at the team's website: `http://www.cs.utexas.edu/~AustinVilla/sim/3dsimulation/`

The work of WrightEagle is supported by the Natural Science Foundations of China under Grant No. 60745002, and No. 61175057, as well as the National Hi-Tech Project of China under Grant No. 2008AA01Z150.

The work of UT Austin villa took place in the Learning Agents Research Group (LARG) at the Artificial Intelligence Laboratory, The University of Texas at Austin. LARG research is supported in part by grants from the National Science Foundation (IIS-0917122), ONR (N00014-09-1-0658), and the Federal Highway Administration (DTFH61-07-H-00030).

References

1. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley & Sons, Inc. (1994)
2. Dietterich, T.G.: The MAXQ method for hierarchical reinforcement learning. Proceedings of the fifteenth international conference on machine learning (1998) Volume: 8, Issue: c, Publisher: Morgan Kaufmann, Pages: 118-126 (1999)
3. Sutton, R.S., Precup, D., Singh, S.: Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. Artificial Intelligence **112** (1999) 181–211
4. Dietterich, T.G.: Hierarchical reinforcement learning with the MAXQ value function decomposition. Journal of Machine Learning Research **13** (1999) 63
5. Kaelbling, L.P., Littman, M.L., Cassandra, A.R.: Planning and acting in partially observable stochastic domains. Artificial Intelligence **101** (1998) 99–134
6. Dellaert, F., Fox, D., Burgard, W., Thrun, S.: Monte carlo localization for mobile robots. In: Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C). Volume 2., IEEE (2001) 1322–1328
7. Fan, C., Chen, X.: Bounded incremental real-time dynamic programming. In: Frontiers in the Convergence of Bioscience and Information Technologies. (2007) 637–644
8. Shi, K., Chen, X.: Action-driven markov decision process and the application in robocup. Journal of Chinese Computer Systems **32** (2011) 511 – 515
9. Wu, F., Chen, X.: Solving large-scale and sparse-reward dec-pomdps with correlation-mdps. Lecture Notes in Computer Science **5001** (2008) 208219
10. Wu, F., Zilberstein, S., Chen, X.: Online planning for multi-agent systems with bounded communication. Artificial Intelligence **175** (2011) 487–511
11. MacAlpine, P., Urieli, D., Barrett, S., Kalyanakrishnan, S., Barrera, F., Lopez-Mobilia, A., Ştiurcă, N., Vu, V., Stone, P.: UT Austin Villa 2011: A champion agent in the RoboCup 3D soccer simulation competition. In: Proc. of 11th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2012). (2012)
12. Graf, C., Härtl, A., Röfer, T., Laue, T.: A robust closed-loop gait for the standard platform league humanoid. In: Proc. of the 4th Workshop on Humanoid Soccer Robots in conjunction with the 2009 IEEE-RAS Int. Conf. on Humanoid Robots. (2009) 30 – 37
13. Hansen, N.: The CMA Evolution Strategy: A Tutorial. (2009) `http://www.lri.fr/~hansen/cmatutorial.pdf`.
14. Urieli, D., MacAlpine, P., Kalyanakrishnan, S., Bentor, Y., Stone, P.: On optimizing interdependent skills: A case study in simulated 3D humanoid robot soccer. In: Proc. of the Tenth Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2011). (2011) 769–776