Online Appendix to: Online Planning for Large Markov Decision Processes with Hierarchical Decomposition

AIJUN BAI, FENG WU, and XIAOPING CHEN, University of Science and Technology of China

A. INTRODUCTION TO ROBOCUP 2D

In RoboCup 2D, a central *server* simulates a 2D soccer field, where two teams of fully autonomous agents connecting to the server via network sockets will be playing a simulated soccer game in real time. A game is normally 6,000 timesteps (also known as *cycles*). Each cycle lasts for 100ms. Each team consists of 11 player agents, each of which interacts independently with the server by (1) receiving perceptional information, (2) making a decision, and (3) sending actions back to the server. Perceptional information for each agent contains only noisy geometric information, such as the distances and angles to the ball, other players, and landmarks within the agent's field of view. Actions are atomic commands defined by the server, such as turning the body (or neck) to an angle, dashing in a given direction with certain power, and kicking the ball with certain angle and power. All primitive actions contain random uncertainties. The key challenge lies in the fact that RoboCup 2D is a fully distributed, multiagent stochastic domain with continuous state, action, and observation spaces [Stone 2000]. More information about RoboCup 2D is available at the official Wiki page.[4] A snapshot of the RoboCup 2D 2013 final competition is also shown in Figure 10. In the figure, the left team is WrightEagle (our team—and the winner), and the right team is HELIOS from Japan.

A.1. RoboCup 2D as a MDP

Generally, RoboCup 2D is a partially observable multiagent domain with continuous state, action, and observation spaces. To model it as a fully observable single-agent Markov decision process (MDP), we define the MDP model with the state space, the action space, the transition function, and the reward function as follows.

State space. The joint state of the RoboCup 2D domain is represented as a vector $\boldsymbol{s} = (s_0, s_2, \ldots, s_{22})$, containing state variables that totally cover 23 objects, including the agent itself (s_u, $u \in [1, 11]$), 10 teammates ($\{s_1, \ldots, s_{11}\} \backslash s_u$), 11 opponents ($\{s_{12}, \ldots, s_{22}\}$), and the ball ($s_0$). The state variable for each player is $s_i = (x, y, \dot{x}, \dot{y}, \alpha, \beta)$, and the state variable for the ball is $s_0 = (x, y, \dot{x}, \dot{y})$, where (x, y), $(\dot{x}, \dot{y})$, α, and β are positions, velocities, body angles, and neck angles, respectively. Some other state information of players, such as stamina and view angle, are ignored in the state representation. As aforementioned, the resulting full state vector has a dimensionality of 136, leading to a state space with 10^{408} states even if we discretize each state variable into only 10^3 values, which is extremely larger than domains usually studied in the literature.

Action space. All primitive actions are completely defined by the 2D server, including dash, kick, tackle, turn, and turn_neck. They all have continuous parameters, leading to a continuous action space. The dash action moves the agent with a certain power in a given direction; the kick action kicks the ball by accelerating it in a specified direction

[4]http://wiki.robocup.org/wiki/Soccer_Simulation_League.

© 2015 ACM 2157-6904/2015/07-ART45 $15.00
DOI: http://dx.doi.org/10.1145/2717316

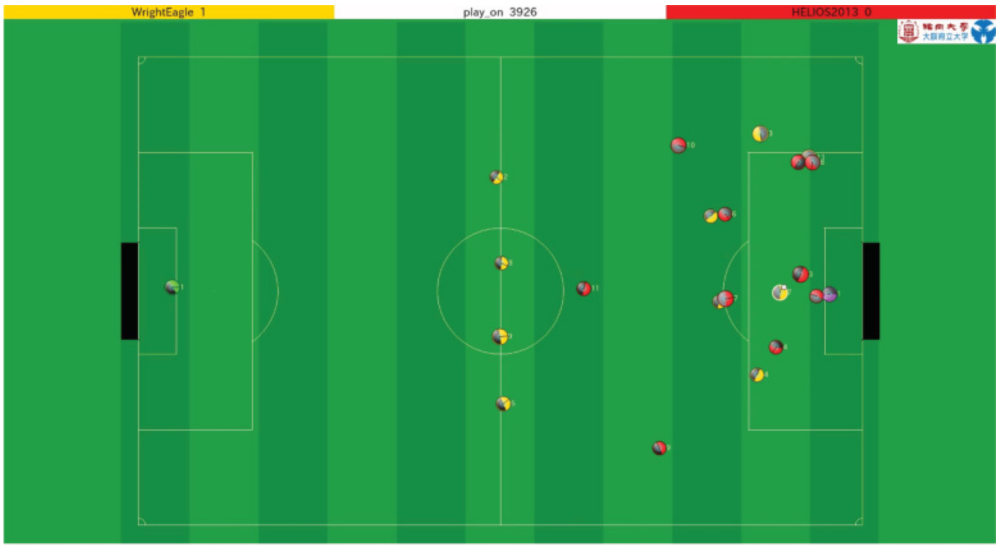

Fig. 10. A snapshot of RoboCup 2D 2013 final competition. ©The RoboCup Federation.

if the ball if kickable for the agent; the tackle action is similar to kick, but the agent can not move for 10 cycles after tackling the ball; and the turn and turn_neck actions change the agent's body and neck angles, respectively.

Transition function. The underlying transition models for all primitive actions are fully determined by the server as a set of *generative* models. Take the dash action for example. It has two parameters: $power \in [0, 1]$ and $angle \in [0, 2\pi)$. Suppose that $power = p$ and $angle = \theta$; the acceleration is then calculated as $(\ddot{x}, \ddot{y}) = (pA\cos\theta, pA\sin\theta) + r_a$, where A is the maximal acceleration and r_a is a small noisy term. The physical state is then updated as $(x, y) \leftarrow (x, y) + (\dot{x}, \dot{y}) + (\ddot{x}, \ddot{y})$ and $(\dot{x}, \dot{y}) \leftarrow (\dot{x}, \dot{y})\omega + (\ddot{x}, \ddot{y})\omega$, where $\omega = 0.4$ is the *speed decay*. In the planning phase, we assume the other players have a greedy behavior model in terms of action evaluation, which means that they will perform the greedy action that maximizes an evaluation function. For state transition, the agent is assumed to be able to observe the whole field. So we do not need to consider the observing action (e.g., turn_neck) in planning. Observing actions are specifically considered in an observation decision-making system, which will not be covered in this article.

Reward function. According to the game rules in RoboCup 2D, the agent receives a reward only in two cases: in one case, when the agent's team scores a goal, the agent receives a reward of $+1$; in the other case, when the opponent team scores a goal, the agent receives a reward of -1. This reward function is extremely sparse: the agent may earn zero rewards for thousands of cycles before the ball is finally scored or conceded. A set of local-reward functions and/or heuristic functions is developed for each subtask to make a more informative online search when implementing MAXQ-OP. For example, for the navigation task, we introduce a reward of -1 for each dash or turn action performed to ensure that the agent will find a way to move to the target as soon as possible; for passing and dribbling tasks, we introduce a heuristic function to evaluate terminal states in the search tree to let the agent try to attack in a faster way.

A.2. State Estimation for RoboCup 2D

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. In our team, the agent estimates the current state from its *belief* [Kaelbling et al. 1998]. A belief b is a probability distribution over the state space, with $b(\boldsymbol{s})$ denoting the probability that the environment is in state $\boldsymbol{s}$.

For state estimation, we assume conditional independence between individual objects, and thus the belief $b(\boldsymbol{s})$ can be expressed as

$$b(\boldsymbol{s}) = \prod_{0 \le i \le 22} b_i(\boldsymbol{s}[i]), \tag{40}$$

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) is then used to approximate b_i as

$$b_i(\boldsymbol{s}[i]) \approx \{\boldsymbol{x}_{ij}, w_{ij}\}_{j=1\dots m_i}, \tag{41}$$

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

At each decision cycle, samples are updated by Monte Carlo procedures following the domain's *motion model* and *sensor model* [Dellaert et al. 2001]. The motion model is completely defined by the transition function under the assumption that other players are behaving according to a prior model. For each particle of the ball, if the particle can be kicked by any particles of any players, it is assumed to be kicked at a random direction; otherwise, it just continues the last motion status, such as moving or staying still. For each particle of a player, if any particle of the ball is within its kickable area, it is assumed to perform a kick action; otherwise, it is simply assumed to take a random dash action. For the agent, the known action executed last cycle is used directly. Once the motion statuses for each object have been determined, the particles are separately predicated according to the respective motion models defined by the server. The sensor model is implemented according to the observation model defined by the server. We update the agent first. Suppose that the observation is $o = \{m_1, m_2, \dots\}, m_n$, where $m_i = (x, y, \dot{x}, \dot{y})$ is the observed landmark with (x, y) being the relative position and $(\dot{x}, \dot{y})$ being the relative velocity (if any). Assuming that each landmark is observed independently, then the observation function given particle $\boldsymbol{x}$ is

$$\Pr(o \mid \boldsymbol{x}) = \prod_{1 \le i \le n} \Pr(m_i \mid \boldsymbol{x}), \tag{42}$$

where $\Pr(m_i \mid \boldsymbol{x})$ gives the observation model for landmark m_i, which is defined by the server. It is worth noting that the original observation function is inflated to have a more smooth distribution. After resampling, the state of the agent is estimated as the expectation of the particles. When updating the ball and other players, we assume that the state of the agent is known as the estimated state resulting from particle filtering. Take the ball as an example. Suppose that the observed position and velocity are (x, y) and $(\dot{x}, \dot{y})$, respectively, and then given particle $\boldsymbol{x}$, the observation function reads $\Pr(x, y, \dot{x}, \dot{y} \mid \boldsymbol{x}, s_u)$, where s_u the estimated state of the agent, noting that the observation function is totally defined by the server. Finally, the environment's joint state $\boldsymbol{s}$ is estimated as

$$\boldsymbol{s}[i] = \sum_{1 \le j \le m_i} w_{ij} \boldsymbol{x}_{ij}. \tag{43}$$

Table VI. Average Error of the Agent's Estimated Self-State

x (m)	y (m)	$\dot{x}$ (m/s)	$\dot{y}$ (m/s)	α (Deg)	β (Deg)
0.047	0.046	0.0014	0.0013	0.44	1.5e-6

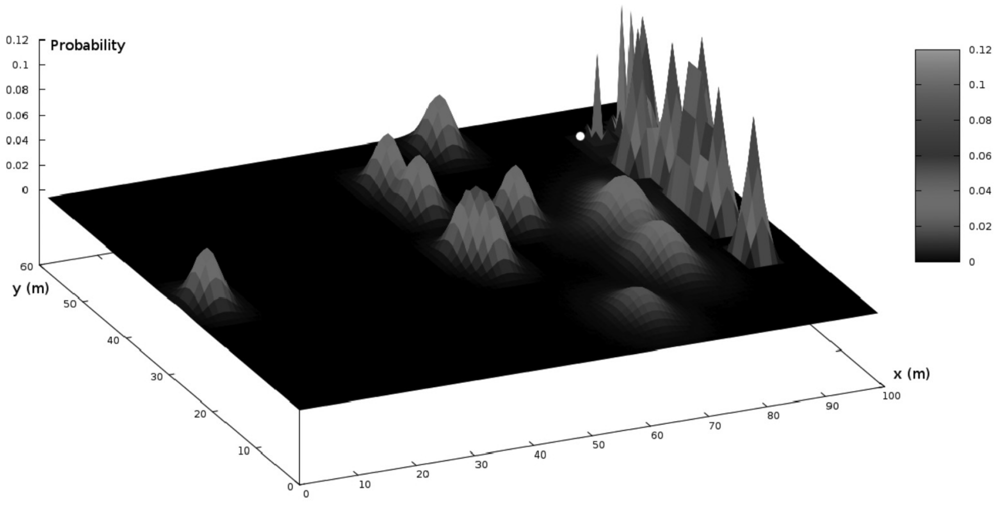

Fig. 11. Belief in terms of position distributions of other players.

Based on empirical results taken from real competitions, the estimated state is precise and robust, particularly for the agent itself and close objects. Table VI reports the average error between the agent's estimated self-state and ground-truth state (which is hidden from the agent) over one randomly generated competition. Note that the error of neck_dir is almost zero, because the turn_neck action in the RoboCup 2D domain is executed without noise. Figure 11 depicts an example of the agent's resulting belief in terms of other players' position distributions obtained from log files of a competition. In the figure, our team is on the left side. The agent, denoted by a small white circle, is holding the ball and facing the opponents' goal area. The position distributions of the agent itself and the ball are not shown in the figure, because they are relatively too narrow and high to be drawn appropriately with other players in the same figure. It can be seen from the figure that the distributions of those players that have not been observed for a number of cycles cover a relatively larger area. By estimating the current state via particle filtering, the state is assumed to be fully observable, and thus the problem can be modeled as an MDP.