POMDP to the Rescue: Boosting Performance for Robocup Rescue

Kegui Wu Wee Sun Lee David Hsu

Abstract— **Disaster response is one of the most critical social issues and introduces quite a few research themes for the AI planning area. Robocup Rescue provides a platform to simulate the rescue process in a city when an earthquake happens. Existing methods consist of multi-agent methods that use greedy heuristics. These methods scale to large maps but suffer from volatile performance under different scenarios. In this work, we propose a planning framework to boost the performance on Robocup Rescue given several policies from the competition to be used as components. More specifically, we use an online POMDP algorithm with macro-actions and restrict it to plan within the space of tasks performed by the agents in the component policies at each time instance. Since the action space contains macro-actions of the component policies, the method is guaranteed to perform at least as well as the best component policy, and possibly better, if sufficient computation is provided. On the other hand, the restriction of the tasks to those suggested by component policies reduces the computational complexity of planning and allows the planning method to be practically applied. Experiment results show that our planner generates better performance than the best component policy for some scenarios and gives performance comparable to the best component policy for the rest.**

I. INTRODUCTION

To promote research and development in techniques of robotic agents in the socially significant domain of disaster management, the Robocup Rescue Simulation project was initiated in 2001. The system simulates the rescue process by heterogeneous agents, such as fire brigades and ambulance teams, in a city when an earthquake happens.

Maps from competitions of recent years contain up to thousands of buildings and hundreds of civilians. In order to scale to such a large space and still respond in real time, participant teams usually use heuristic algorithms. These algorithms build model based on the observation history, to predict the development of the environment or to estimate the importance of targets. In addition, instead of planning, they usually select actions based on greedy heuristics. Hence, the outcome depends heavily on the quality and coverage of the model. Usually, the models of the participating teams are not comprehensive, and performance is volatile under different scenarios.

An alternative method for developing policies for disaster management is to do planning under uncertainty. Partially observable Markov decision process (POMDP) provides a principled planning framework under uncertain and partially observable environment and is powerful enough to model the disaster management problems. However, POMDP is notorious for its computational complexity. For Robocup Rescue, the size of state space is exponential in the number of buildings and the size of action space is exponential in the number of agents. Both are extremely large, even for state-of-the-art POMDP solvers.

The authors are with School of Computing, National University of Singapore, Singapore. {wu-kegui,leews,dyhsu}@comp.nus.edu.sg

In this work, we seek a middle ground of using planning under uncertainty techniques to boost the performance of a collection of heuristic policies – to exploit the information contained in the policies to constrain the computational complexity, and to go beyond the component policies by using the planning technique to search over their action combinations of the component policies. More specifically, we combine online POMDP algorithm with macro-action techniques and plan in a macro-action space suggested by the component policies.

We first note that it is straightforward to approach the performance of the best policy for each city – simply simulate each policy on each city many times, and use the policy with the best average performance in simulation for each city. In this work, we would like to outperform the best policy whenever the combination of the actions suggested by the component policies are able to do so. Specifically, we constrain the possible actions of each agent to those suggested by the component policies and search over those actions. Since the action space contains the joint actions of each of the component policies, this method will perform at least as well as the best component policy and potentially better, given enough computation time. At the same time, the information in the component policies constrains the action space to be much smaller than the complete action space, allowing the search method to return good results in reasonable time.

Various challenges need to be overcome in order to apply the idea in practice.

- The planning horizon of the problem is long, making planning difficult. To overcome that, we extend a state of the art online POMDP planner, POMCP [1], to plan over macro-actions instead of actions. A macro-action is an extended sub-policy that runs over multiple steps, as opposed to a single step in the case of an action. To construct macro-actions, we divide the map up into regions and treat the allocation of agents to regions as macro-actions.
- Each policy used in the previous Robocup competition is designed as a stand-alone policy. As a consequence, it may not be possible for a component policy to operate when the previous actions come from different policies rather than being generated by the policy itself. To overcome this problem, we approximate each component policy as a function of the history to action. We use a

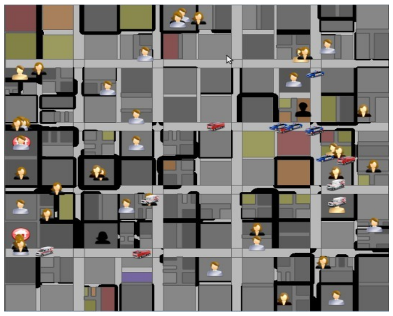

Fig. 1. An example of a map (from [3])

structured support vector machine [2] in order to learn each component policy from the training data generated by running the policies in simulations.

In the rest of the paper, we will first introduce the Robocup Rescue problem in Section II. In Section III, we describe our method of combining online POMDP algorithm with macro-actions and constraining the size of planning space. Finally, we will show and discuss the experiment results in Section IV.

II. Robocup Rescue

A. Problem Description

The Robocup Rescue project simulates heterogeneous agents collaborating to execute the rescue process when an earthquake happens in a city. Map entities include buildings, roads and road blockades (see Figure 1). Civilians are scattered in buildings and roads. Controllable entities, or the so-called platoons consist of three kinds of agents: fire engine, ambulance team and police force. Fire engines are able to extinguish fire, ambulance teams are able to rescue people, while the police force is able to clear road blockades. Another common action for civilians and agents is to move. A move action is a legal path consisting of a sequence of buildings and roads which are connected. An agent's observation is generated by a line-of-sight model. The aim is to save as many civilians as possible while minimizing the damage to buildings.

The Robocup Rescue simulation system is built upon modules communicating with each other. Changes of environment are all simulated by component simulators, such as building collapsing, blockade generation and clearance, damage generation and development, fire development and traffic condition. Each simulator is independent and designed to operate separately.

Among the test cases in the competition, there are some cases that only test the performance of fire engines – no civilians to be rescued and no road blockades to be cleared. In this work, we will also just focus on planning for actions of fire engines. Nevertheless, the framework can easily be extended to situations with heterogeneous agents. For scenarios where only fire engines are present, the score is just the percentage of undamaged buildings.

B. Competition Policies

Competition policies typically build simplified models to approximate the environment. For example, the RI-ONE policy [4], which won the 2012 competition, builds the fire model by assuming that heat is transmitted in the concentric sphere centered on the burning building, capturing only a part of the actual fire simulator. Furthermore, actions are chosen based mostly on greedy heuristics, instead of thorough planning. For example, the RI-ONE policy gives a nearby fire a higher priority and uses the heat flowing in and out to choose the fire to extinguish.

Simplified models and useful heuristics allow competition methods to delivery satisfactory scores. However, even the champion rarely produces best score and performance is hard to predict. Without using principled planning methods, these methods usually deliver volatile performance.

III. PomRescue:Boosting Performance for Robocup Rescue with POMDP

A. POMDP Model

Partially Observable Markov Decision Process is a principled framework for planning under uncertainty. It is general enough to model a variety of real-world sequential decision problems. It is defined as a tuple $< S, A, T, R, O, Z, \gamma >$, where S, A, O denote state space, action space and observation space. T defines the transition model or the uncertainty of taking action. Z is the observation function, modelling the noise of sensors. And R is the reward function, which defines the benefit or penalty that is obtained for performing an action at a particular state.

In particular, we can model the Robocup Rescue problem as POMDP as follows. A state is composed of state variables $\{f_i\}$ which represents the fire level of building i as well as state variables representing the position p_a and water power w_a of each agent a. An action u^i that an agent can take includes moving to a legal area or extinguishing a building fire. Since we are considering a centralized POMDP, a primitive action in the POMDP tuple would be a joint action $a = \{u^1, u^2, \dots\}$. And the reward function computes how much percentage the city will be burnt during a time step given the state and action. The observation consists of the same variables as the state. And the transition and observation functions are defined by the Robocup Rescue simulators.

Unfortunately, POMDP is notorious for its computational difficulty. Point-based algorithms have made dramatic progress in computing approximation solutions in recent years [5], [6], [7]. State-of-the-art solvers can scale to hundreds of thousands of states. In addition, algorithms that tackle continuous space such as MCVI [8] also extend the applicability of POMDP model. However, due to the curse of dimensionality and the curse of history, these solvers are still unable to solve many complex real-world problems.

POMCP (Partially Observable Monte-Carlo Planning) [1] is one of the state-of-the-art online POMDP solvers. It achieves high scalability and performance, by combining

Monte-Carlo tree search and the UCT (Upper Confidence bounds for tree) algorithm.

B. Macro-action

Due to the curse of history, a POMDP solver's performance tends to degrade when the planning horizon becomes longer. Macro-actions can be used to tackle this issue. Macro-action is a temporal abstraction of primitive actions, and can be adopted to mitigate the effect of long horizon. Generally, a macro-action could be an arbitrary mapping from belief to an action. However, macro-actions need to be carefully constructed to retain some favourable properties and achieve good result.

Macro-action has been used to speed up MDP and POMDP algorithms [9], [10]. For POMDP algorithm, earlier works rely on specific representation form of belief and value function, making it difficult to scale to very large space [11]. Recently, macro-action has been adopted to state-of-the-art offline POMDP solver that uses Monte Carlo simulation [12].

Similar to action, for Robocup Rescue, we define the joint macro-action m as a set of individual macro-actions $\{m^i\}$. Each m^i is a pair of $< agent, task >$ that specifies the task that agent i would executes. That is, the macro-action is a task allocation.

We partition the map into regions $r_1, r_2, \ldots, r_n$ and define the task as one of the following three types.

- extinguish(i): each agent can go to a target region r_i and execute the low level task there.
- search: an agent can perform a search routine which searches for fire locations.
- head2refuge: an agent can go to a refuge to refill its water supply.

Besides the high-level macro-action, there will be an engine to generate the low-level primitive actions. The low-level action would be the sequence of specific actions to execute the task. We focus on the high-level actions, that is, to plan in the macro-action space and identify the best task allocation for agents. For the low-level planning, we use heuristic algorithms from the competition methods.

Since we focus on the macro-actions, the complexity of the planning will be reduced significantly. However, at every time step, combination of tasks to be allocated is still exponential to the number of the agents. The idea of constructing a small policy space based on the policies of given competition planners helps to reduce the complexity to a manageable one.

C. POMCP with macro-action

Before introducing the idea of restricting planning space, we first discuss the framework of the algorithm. By combining the power of online POMDP and macro-actions, we can solve large POMDP problems with large action space and long horizon. Routines that extend POMCP with macro-actions are shown in Algorithm 1, Algorithm 2 and Algorithm 3.

Algorithm 1 shows the main framework of planning. Each time step, the planner receives an observation from the Robocup Rescue simulator. The *NewObs* routine decides whether the observation contains any new information, that is, the stopping criteria of macro-action. For example, a new fire is found, or a fire has been put out. If *NewObs* returns true, then the planner will update the root node of the search tree based on the last macro-action and observation (belief update) and search for a new macro-action. Finally, the *ExtractAction* routine calls the low-level planning engine and returns a primitive action given the macro-action.

Algorithm 1 POMCP with macro-action: Plan

```
function PLAN
    for $i = 1$, $i \leq$ HORIZON, $i++$ do
        $obs \leftarrow$ ReceiveObs()
        if NewObs($obs$) then
            UpdateRoot($lastMacro$, $obs$)
            $m \leftarrow$ Search($root$, $i$)
            $lastMacro \leftarrow m$
        end if
        $a \leftarrow$ ExtractAction($m$)
        SendAction($a$)
    end for
end function
```

Algorithm 2 POMCP with macro-action: Search

```
function SEARCH($node$, $depth$)
    for $i = 0$, $i <$ SIMNUM, $i++$ do
        if $node = empty$ then
            $s \sim \mathcal{I}$
        else
            $s \sim \mathcal{B}(node)$
        end if
        Simulate($s$,$node$,$depth$)
    end for
    return $\arg\max_m V(hm)$
end function
```

Algorithm 2 describes the search step. The search will run *SIMNUM* simulations. Each simulation starts from sampling a state s from the belief $\mathcal{B}(node)$ corresponding to the tree node or from the initial belief $\mathcal{I}$ when time step is 0 and tree root is empty. Then the routine *Simulate* will run the simulation from s and update the corresponding nodes.

Algorithm 3 describes the *Simulate* routine. $node_m$ represents the AND-node in the m branch of $node$, and $node_{mo}$ represents the OR-node with m action branch and o observation branch from $node$. The process is similar to that of the original POMCP. It will keep choosing the macro-action branch with best upper confidence bound, sampling the next state s', observation o and reward r and updating the statistics of the tree node, until reach the end of the tree. Then a new tree node will be created, with macro-actions produced with routine *GenerateMacros*. Finally, the routine will execute the rollout policy.

Algorithm 3 POMCP with macro-action: Simulate

```
function SIMULATE($s$,$node$,$depth$)
    if $depth > HORIZON$ then
        return 0
    end if
    if $node \notin T$ then
        $macros \leftarrow$ GenerateMacros($node$)
        for all $m \in macros$ do
            $T(node_m) \leftarrow (0, 0, \emptyset)$
        end for
        return Rollout($s$,$node$,$depth$)
    end if
    $m \leftarrow \arg\max_b V(node_b) + c\sqrt{\frac{\log N(node)}{N(node_b)}}$
    $(s', o, r, step) \sim \mathcal{G}(s, m)$
    $R \leftarrow r + \lambda$ Simulate($s'$,$node_{mo}$,$depth + step$)
    $N(node) \leftarrow N(node) + 1$
    $N(node_m) \leftarrow N(node_m) + 1$
    $V(node_m) \leftarrow V(node_m) + \frac{R - V(node_m)}{N(node_m)}$
    return $R$
end function
```

D. Selecting Macro-Actions to Plan

Our combination of online POMDP and macro-action significantly reduces the size of planning space. Normally, we have to plan in the whole macro-action space for each step *Search* routine is called in the framework of Section III-C. However, the number of macro-actions to plan is exponential to the number of agents, which is still too large for the POMCP solver.

On the other hand, the competition policies can suggest a small policy space to search. We call a competition policy as component policy $\pi : b \rightarrow m$, which maps from a belief to a macro action. In addition, $\pi(b, a)$ defines the task the agent a is assigned to in macro-action $\pi(b)$. Given a belief b, two component policies π_1 and π_2 are consistent on the task of agent a if $\pi_1(b, a)$ and $\pi_2(b, a)$ are the same. Every search step, the *GenerateMacros* routine will return the set of macro-actions to search. We select the macro-actions to search based on the following criteria.

First, if all the component policies are consistent on the task of an agent, the suggested task is adopted for that agent. In another word, the agent is assigned with the same task in all the macro-actions the POMDP planner is going to search.

Second, for the n_a remaining agents for which the component policies suggest different tasks, all the suggested tasks are allowable tasks of the agents. That forms the task allocation problem with n_f tasks and n_a agents. Hence, the number of the potential macro-actions is ${n_f}^{n_a}$.

Third, to reduce the number further, we can calculate the number of votes from the component policies for each task. In particular, for a component policy π and any agent a, we add the votes by one for the task $\pi(b, a)$. So the votes for a task t would be $\sum_{b,a} [\pi(b, a) = t]$. The votes can be used to indicate the importance of the tasks. We then prioritize the agents to the tasks according to the importance of the task and the distance the agent need to travel. More specifically, among the agents to be planned, we prioritize more agents to tasks with higher importance, and the agents assigned are those nearest to the target region of the task. Finally, we restrict the total number of macro-actions at any step to a maximum number according to the priority in order to keep the planning manageable (in our experiments, we limit the number to a maximum of ten).

As can be seen, via the above method, the selected action space will contain the actions of each of the component policy. Hence the planning will perform at least as well as the best component policy and potentially better, given enough running time. Meanwhile, the information in the component policies constrain the selected action space to be much smaller than the complete action space which enables the planning to return good results in reasonable time.

E. Learning Policies

As mentioned in the last section, the *GenerateMacros* routine needs component policies from the competition that can map any belief to a macro-action, to suggest tasks for agents. Ideally, we would like to run the competition planners to suggest the appropriate macro-actions. However, this is not always possible. We describe an extreme case to illustrate the problem. A policy for a POMDP is a mapping from a history $(o_1, a_1, o_2, a_2, \ldots, o_{n-1}, a_{n-1}, o_n)$ (or equivalently a belief) to an action. Consider a policy that accepts only histories that contain a subset of allowable actions, $A' \subset A$, and outputs an action from the same subset A'. This policy may not be defined for inputs that contain actions outside of A' but can be used without any problem in the competition, as the histories it receives are generated by the policy itself and contains only actions in A'. When this policy is used within the planner, it may encounter histories that contain actions generated by the planner that are outside of A', and hence may not be able to return an output.

To solve this problem, we need some way to extrapolate the policy to all inputs. We propose to use machine learning to do the extrapolation. Given a set of examples consisting of history-action pairs generated from a competition policy, we extrapolate the policy to all possible inputs by learning from the examples. One simple way to learn a competition policy would be to learn an independent mapping from history to action for each agent. However, the actions of agents are usually correlated in some way and we would like to exploit the correlations. In this work we use structured support vector machine (SVM) [2], in order to learn the competition policies. Structured SVM is a machine learning method used to predict labels on parts of a structure. We use a binary tree(shown in Figure 2) as the structure, where each node is associated with a fixed agent and labeled with the action taken by the agent. The structured SVM is then used to predict the actions of all the agents, corresponding to the labels of the nodes of the tree. Using a tree as the structure allows us to capture some of the hierarchical relationships in teams of agents (e.g. a parent can be viewed as a leader of the team consisting of itself and its children). Learning and

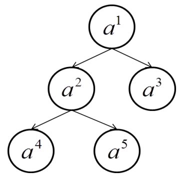

Fig. 2. An example of the tree model which is the output of Structured SVM. Each node is associated with a fixed agent i and labeled with its action a^i. We define a feature function $\Psi(x, y)$ mapping x and y to a feature vector where x is the world state and y is the tree structure. The Structured SVM will learn the weight vector w and predict the tree y that maximizes $w \cdot \Psi(x, y)$.

prediction can also be done efficiently with structured SVM when the structure is a tree.

We used features of the history, like observed fires, unknown areas, and agents' positions as input to the structured SVM. The features for every tree node include the following:

- fire intensity for each region
- unknown area of each region
- position of the agent corresponding to the tree node
- observation of fire intensity of the agent
- previous action of the agent
- water power of the agent
- the agent's distances to other agents

In addition, features for tree edges are indicator functions of the pair of macro-actions corresponding to the pair of agents associated with the edge. This captures the correlation in the tasks of the two agents. Finally, based on the learned model, we can get a policy mapping agents to tasks given the action-observation history.

F. Miscellaneous

1) Simplifying Simulators: POMCP runs many simulations to back up values for a belief and the performance depends on the number of simulations being executed. The simulation overhead hence becomes a major factor of efficiency of the algorithm. Since original Robocup Rescue simulators are very complicated, they are simplified to reduce the overhead.

2) Rollout Policy: One of the important factors in POMCP that affect its performance is the rollout policy. A rollout policy should ideally be an optimal policy or near-optimal one. We use policy of competition method RI-ONE as rollout policy and assume it will provide good rewards in most cases.

3) Macro-action Stopping Criteria: Macro-action is designed to stop when some reasonable conditions are met. For our planner, a task execution should be stopped when it is finished or interrupted by other conditions that require a reallocation of tasks. We try to loosen the criteria to reduce the stopping frequency of macro-action, since replanning for macro-actions is quite expensive. For example, generally replanning for macro-action is needed when a new fire is discovered. However, when the fire is a nearby fire caused by a fire the agents are extinguishing, or a very severe fire that has almost burnt out the building and does not need to be extinguished, the macro-action will not be interrupted.

TABLE I

ACCURACY OF LEARNING POLICIES ON MAP BERLIN5.

	training set(2500 samples)	test set(1000 samples)
ZJUBase	0.877	0.786
RI-ONE	0.913	0.749
SOS	0.892	0.773

IV. EXPERIMENTS

In this section, we present the results of running simulations on the competition maps.

As mentioned in Section II, we only consider the performance of fire engines. There are no other agents like ambulance, police force, or civilians on the map. Building collapsing and road blocking are also disabled. The agents' task is to put out the fires and keep as many buildings unburned as possible. So, the primitive actions the agent can take are moving and fire extinguishing. The agent may also need to go to a refuge to refill itself if its water is used up. Initially, there is no fire on the map. The ignition rate, which is the average number of buildings that will be ignited randomly at each time step, will also be adjusted so that the task is not too easy or too difficult for the agents. The total number of agents is 10 for each map. The total time steps for simulation on each map is up to 300. The maps used are from Robocup Rescue 2012. And the results are compared with the policies from the competition teams ZJUBase, RI-ONE and SOS [13].

We first show the results of learning policies of RI-ONE, ZJUBase, and SOS using Structured SVM in Table I. The training accuracy, which is the percentage of correct actions predicted for agents, is fairly high, although the test set accuracy is somewhat lower. When the classifiers are used as policy in real simulation runs, the score of the resulting policies generally lower than that of the original policy it was trained on. Despite this, our results show that these classifiers can be successfully used in conjunction with PomRescue.

One simpler method for combining the recommendation of the classifiers is to use the three recommendations from the three classifiers as macro-actions with the online POMDP algorithm. We call this method SimpleMacro. The policy space of SimpleMacro includes the best of the component policies, hence it can perform as well as the best component policy, if given enough search time. However, note that our learned component policies do not perform as well as the competition policies they are approximating.

The results of scores on the competition maps are seen on Table II. The SOS policy crashed on our computer on some of the runs and we compute the average performance based on the successful runs.

We ran 100 simulations for each map. The score represents the percentage of buildings that are not burnt out at the final time step. As can be seen, performance of competition planners are uneven. For example, RI-ONE's average score is much better than ZJUBase's for Istanbul3 but much worse for Paris. In addition, SimpleMacro, which can theoretically be as good or better than the component policies, generally

TABLE II

SCORES ON COMPETITION MAPS

	PomRescue	ZJUBase	RI-ONE	SOS	SimpleMacro
Berlin5	38.82	33.20	31.13	29.62	31.98
Mexico3	35.73	32.14	26.02	28.11	31.83
VC4	49.18	43.83	45.32	39.82	43.10
Eindhoven4	54.23	51.78	46.78	48.34	50.31
Eindhoven5	55.18	53.17	54.20	48.56	51.34
Istanbul3	38.12	33.10	40.92	31.69	35.12
Kobe4	48.31	49.32	43.18	47.32	48.12
Paris4	57.37	59.83	47.39	51.33	53.38

TABLE III

AVERAGE RUNTIME FOR MAP BERLIN5.

Method	Runtime(s)
PomRescue	2926.7
ZJUBase	861.8
RI-ONE	789.2
SOS	842.2
SimpleMacro	1269.8

obtains scores that are better than the poorest competition policy, but poorer that the best competition policy. It appears that a simple combination of component policies fail to outperform the competition policies for these problems. On the other hand, PomRescue performs comparably or better on the 8 maps. The performance is expected to be comparable since the policy space of PomRescue contains policies approximated from the competition policies. The better performance indicates that the policy space of PomRescue is rich enough to contain better policies, and yet remains computationally practical.

The average time for simulation on map Berlin5 for different methods are presented in Table III. As it shows, PomRescue takes about 3 times longer than other planners since it performs search. The time, however, is still reasonable for real-time performance in a disaster situation.

In Figure 3, we show the actual number of agents and macro-actions used to plan in each search step for a sample simulation. Since we use macro-actions, planning is required only for some of the time steps; otherwise the macro-action is just executed until it stops. As shown in the figure, in the early stage (before time step 50), all component policies suggest the search macro-action for all agents, so there is no need to plan for any agent. With more and more fires appearing, there is more and more inconsistency in

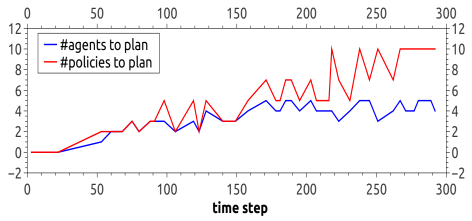

Fig. 3. Number of agents and number of policies to plan

the component policies with more and more target regions containing fires. So the number of agents to plan and the number of selected policies increase accordingly. However, they are still within a manageable range.

V. CONCLUSIONS

We have proposed a method of using several heuristic policies to help constrain the search space of an online POMDP algorithm in multi-agent scenarios. As the search space still contains the original policies, the POMDP planning algorithm is guaranteed to perform as well as the best component heuristic policies if it is run for a long enough time. Experiments on a Robocup Rescue task shows that the method is able to outperform the best component policy on many of the problem instances. It would be interesting to see if the method can be used to improve the performance of other multi-agent planning problems.

VI. ACKNOWLEDGEMENT

This research was supported by the National Research Foundation Singapore through the Singapore MIT Alliance for Research and Technology's IRG research program (R-252-000-571-592).

REFERENCES

[1] D. Silver and J. Veness, "Monte-Carlo planning in large POMDPs," in *Advances in Neural Information Processing Systems*, pp. 2164–2172, 2010.

[2] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun, "Support vector machine learning for interdependent and structured output spaces," in *Proceedings of the twenty-first international conference on Machine learning*, p. 104, 2004.

[3] A. B. da Silva, L. G. Nardin, and J. S. Sichman, "RoboCup Rescue simulator tutorial,"

[4] S. Okazaki, T. Nakagawa, K. Miyake, S. Oguri, and M. Takashita, "RoboCupRescue 2013-Rescue Simulation League team description Ri-one (Japan),"

[5] J. Pineau, G. Gordon, S. Thrun, *et al.*, "Point-based value iteration: An anytime algorithm for POMDPs," in *Proc. Int. Jnt. Conf. on Artificial Intelligence*, vol. 3, pp. 1025–1032, 2003.

[6] T. Smith and R. Simmons, "Heuristic search value iteration for POMDPs," in *Proc. Uncertainty in Artificial Intelligence*, pp. 520–527, 2004.

[7] H. Kurniawati, D. Hsu, and W. S. Lee, "SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces," in *Proc. Robotics: Science and Systems*, 2008.

[8] H. Bai, D. Hsu, W. S. Lee, and V. Ngo, "Monte Carlo value iteration for continuous-state POMDPs," in *Proc. Int. Workshop on the Algorithmic Foundations of Robotics (WAFR)*, pp. 175–191, 2011.

[9] M. Hauskrecht, N. Meuleau, L. P. Kaelbling, T. Dean, and C. Boutilier, "Hierarchical solution of Markov decision processes using macro-actions," in *Proc. Uncertainty in Artificial Intelligence*, pp. 220–229, 1998.

[10] G. Theocharous and L. P. Kaelbling, "Approximate planning in POMDPs with macro-actions," in *Advances in Neural Information Processing Systems*, 2003.

[11] J. Pineau, N. Roy, and S. Thrun, "A hierarchical approach to POMDP planning and execution," in *Workshop on hierarchy and memory in reinforcement learning (ICML)*, vol. 65, p. 51, 2001.

[12] Z. W. Lim, W. S. Lee, and D. Hsu, "Monte Carlo value iteration with macro-actions," in *Advances in Neural Information Processing Systems*, pp. 1287–1295, 2011.

[13] F. Amigoni, A. Visser, and M. Tsushima, "Robocup 2012 Rescue Simulation League winners," in *RoboCup 2012: Robot Soccer World Cup XVI*, pp. 20–35, Springer, 2013.