UAV Cooperative Control with Stochastic Risk Models

Alborz Geramifard, Joshua Redding, Nicholas Roy, and Jonathan P. How

***Abstract*— Risk and reward are fundamental concepts in the cooperative control of unmanned systems. This paper focuses on a constructive relationship between a cooperative planner and a learner in order to mitigate the learning risk while boosting the asymptotic performance and safety of the behavior. Our framework is an instance of the intelligent cooperative control architecture (iCCA) with a learner initially following a "safe" policy generated by the cooperative planner. The learner incrementally improves this baseline policy through interaction, while avoiding behaviors believed to be "risky". Natural actor-critic and Sarsa algorithms are used as the learning methods in the iCCA framework, while the consensus-based bundle algorithm (CBBA) is implemented as the baseline cooperative planning algorithm. This paper extends previous work toward the coupling of learning and cooperative control strategies in real-time stochastic domains in two ways: (1) the risk analysis module supports stochastic risk models, and (2) learning schemes that do not store the policy as a separate entity are integrated with the cooperative planner extending the applicability of iCCA framework. The performance of the resulting approaches are demonstrated through simulation of limited fuel UAVs in a stochastic task assignment problem. Results show an 8% reduction in risk, while improving the performance up to 30% and maintaining a computational load well within the requirements of real-time implementation.**

I. Introduction

Accompanying the impressive advances in the flight capabilities of UAV platforms have been equally dramatic leaps in intelligent control of UAV systems. Intelligent cooperative control of teams of UAVs has been the topic of much recent research, and risk-mitigation is of particular interest [1,2]. The concept of *risk* is common among humans, robots and software agents alike. Amongst the latter, risk models combined with relevant observations are routinely used in analyzing potential actions for unintended or risky outcomes. In previous work, the basic concepts of risk and risk-analysis were integrated into a planning and learning framework called *iCCA* [4,8]. The focus of this research however, is to generalize the embedded risk model and then learn to mitigate risk more effectively in stochastic environments.

In a multi-agent setting, participating agents typically advertise their capabilities to each other and/or the planner *a priori* or in real-time. Task assignment and planning algorithms implicitly rely on the accuracy of such capabilities for any guarantees on the resulting performance. In many situations however, an agent remaining capable of its advertised range of tasks is a strong function of how much risk the agent takes while carrying out its assigned tasks. Taking high or unnecessary risks jeopardizes an agent's capabilities and thereby also the performance/effectiveness of the cooperative plan.

A. Geramifard, Ph.D. Candidate, Aerospace Controls Lab, MIT
J. Redding, Ph.D. Candidate, Aerospace Controls Lab, MIT
N. Roy, Associate Professor of Aeronautics and Astronautics, MIT
J. How, Richard C. Maclaurin Professor of Aeronautics and Astronautics, MIT
{agf,jredding,nickroy,jhow}@mit.edu

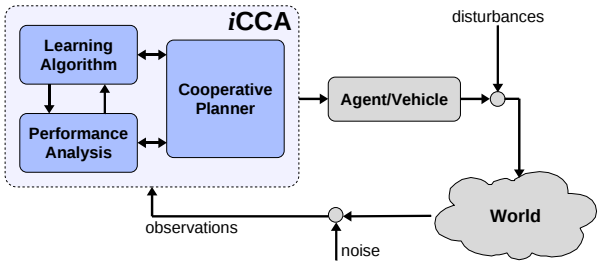


Fig. 1. **i**ntelligent **C**ooperative **C**ontrol **A**rchitecture, a framework for the integration of cooperative control algorithms and machine learning techniques [4].

The notion of risk is broad enough to also include noise, unmodeled dynamics and uncertainties in addition to the more obvious inclusions of off-limit areas. Cooperative control algorithms have been designed to address many related issues that may also represent risk, including humans-in-the-loop, imperfect situational awareness, sparse communication networks, and complex environments. While many of these approaches have been successfully demonstrated in a variety of simulations and some focused experiments, there remains a need to improve the performance of cooperative plans in real-world applications. For example, cooperative control algorithms are often based on simple, abstract models of the underlying system. Using simplified models may aid computational tractability and enable quick analysis, but at the cost of ignoring real-world complexities, thus implicitly introducing the possibility of significant risk elements into cooperative plans. The research question addressed here can then be stated as:

> *How to take advantage of the cooperative planner and the current domain knowledge to mitigate the risk involved in learning, while improving the performance and safety of the cooperative plans over time in the presence of noise and uncertainty?*

To further investigate this question, we adopt the intelligent cooperative control architecture (*iCCA* [4]) as a framework for more tightly coupling cooperative planning and learning algorithms. Fig. 1 shows the *iCCA* framework which is comprised of a cooperative planner, a learner, and a performance analyzer. Each of these modules is interconnected and plays a key role in the overall architecture. In this

research, the performance analysis module is implemented as risk analysis where actions suggested by the learner can be overridden by the baseline cooperative planner if they are deemed too risky. This synergistic planner-learner relationship yields a "safe" policy in the eyes of the planner, upon which the learner improve. Ultimately, this relationship will help to bridge the gap to successful and intelligent execution in real-world missions.

Previously, we integrated the consensus-based bundle algorithm (CBBA) [5] as the planner, natural actor-critic [6,7] as the learner, and a deterministic risk analyzer as the performance analyzer [8]. In this paper, we extend our previous work in two ways:

- The risk analyzer component is extended to handle stochastic models
- By introducing a new wrapper, learning methods with no explicit policy formulation can be integrated within the iCCA framework

The first extension allows for accurate approximation of realistic world dynamics, while the second extension broadens the applicability of iCCA framework to a larger set of learning methods.

The remainder of this paper proceeds as follows: Section II provides background information for the core concepts of this research, including Markov decision processes and Sarsa and natural actor-critic reinforcement learning algorithms. Section III then highlights the problem of interest by defining a pedagogical scenario where planning and learning algorithms are used in conjunction with stochastic risk models. Section IV outlines the proposed technical approach for learning to mitigate risk by providing the details of each element of the chosen *iCCA* framework. The performance of the chosen approach is then demonstrated in Section V through simulation of a team of limited-fuel UAVs servicing stochastic targets. We conclude the paper in Section VI.

II. Background

A. Markov Decision Processes

Markov decision processes (MDPs) provide a general formulation for sequential planning under uncertainty [**?**, 10,12]–[14]. MDPs are a natural framework for solving multi-agent planning problems as their versatility allows modeling of stochastic system dynamics as well as interdependencies between agents. An MDP is defined by tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}^a_{ss'}, \mathcal{R}^a_{ss'}, \gamma)$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of possible actions. Taking action a from state s has $\mathcal{P}^a_{ss'}$ probability of ending up in state s' and receiving reward $\mathcal{R}^a_{ss'}$. Finally $\gamma \in [0,1]$ is the discount factor used to prioritize early rewards against future rewards.[1] A trajectory of experience is defined by sequence $s_0, a_0, r_0, s_1, a_1, r_1, \cdots$, where the agent starts at state s_0, takes action a_0, receives reward r_0, transits to state s_1, and so on. A policy π is defined as a function from $\mathcal{S} \times \mathcal{A}$ to the probability space $[0,1]$, where $\pi(s,a)$ corresponds to the probability of taking action a from state s. The value of each state-action pair under policy π, $Q^\pi(s,a)$, is defined as the expected sum of discounted rewards when the agent takes action a from state s and follow policy π thereafter:

$$Q^\pi(s,a) = E_\pi \left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t \middle| s_0 = s, a_0 = a, \right].$$

The optimal policy π^* maximizes the above expectation for all state-action pairs:

$$\pi^* = \underset{a}{\operatorname{argmax}}\, Q^{\pi^*}(s,a)$$

B. Reinforcement Learning in MDPs

The underlying goal of the two reinforcement learning algorithms presented here is to improve performance of the cooperative planning system over time using observed rewards by exploring new agent behaviors that may lead to more favorable outcomes. The details of how these algorithms accomplish this goal are discussed in the following sections.

1) Sarsa: A popular approach among MDP solvers is to find an approximation to $Q^\pi(s,a)$ (policy evaluation) and update the policy with respect to the resulting values (policy improvement). Temporal Difference learning (TD) [15] is a traditional policy evaluation method in which the current $Q(s,a)$ is adjusted based on the error between the expected reward given by Q and the actual received reward. Given $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$ and the current value estimates, the temporal difference (TD) error, δ_t, is calculated as:

$$\delta_t(Q) = r_t + \gamma Q^\pi(s_{t+1}, a_{t+1}) - Q^\pi(s_t, a_t).$$

The one-step TD algorithm, also known as TD(0), updates the value estimates using:

$$Q^\pi(s_t, a_t) = Q^\pi(s_t, a_t) + \alpha \delta_t(Q), \tag{1}$$

where α is the learning rate. Sarsa (state action reward state action) [9] is basic TD for which the policy is directly derived from the Q values as:

$$\pi^{Sarsa}(s,a) = \begin{cases} 1-\epsilon & a = \operatorname{argmax}_a Q(s,a) \\ \frac{\epsilon}{|\mathcal{A}|} & \text{Otherwise} \end{cases}.$$

This policy is also known as the ϵ-greedy policy[2].

2) Natural Actor-Critic: Actor-critic methods parameterize the policy and store it as a separate entity named *actor*. In this paper, the actor is a class of policies represented as the Gibbs softmax distribution:

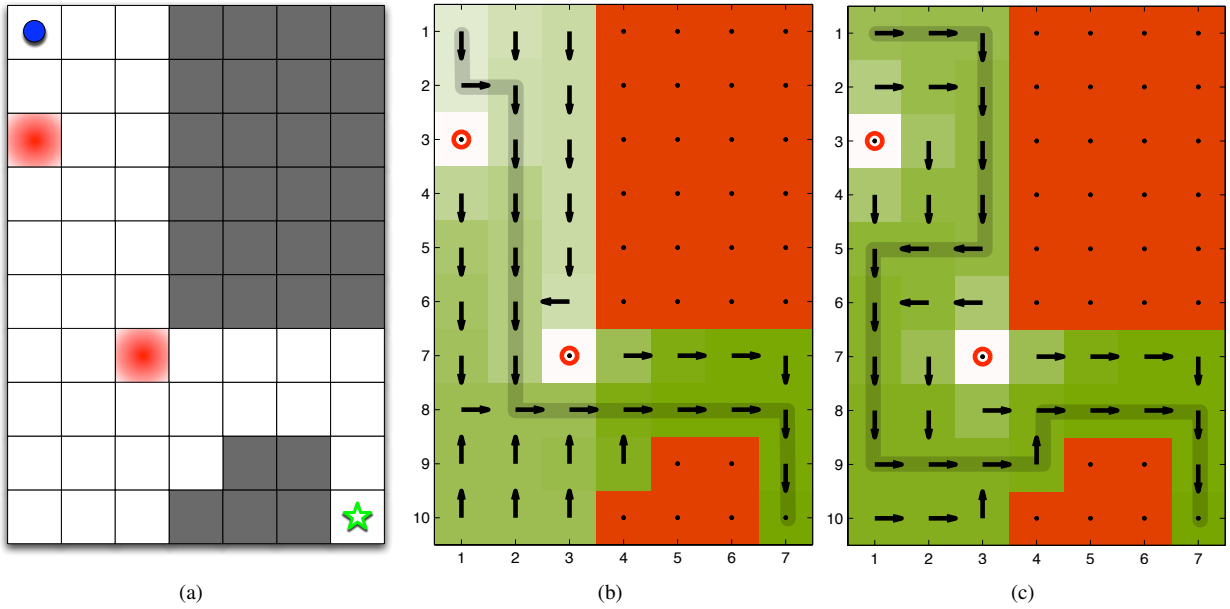
$$\pi^{AC}(s,a) = \frac{e^{P(s,a)/\tau}}{\sum_b e^{P(s,b)/\tau}},$$

in which $P(s,a) \in \mathcal{R}$ is the preference of taking action a in state s, and $\tau \in (0, \infty]$ is a knob allowing for shifts between greedy and random action selection. Since we use a tabular representation, the actor update amounts to:

$$P(s,a) \leftarrow P(s,a) + \alpha Q(s,a)$$

[1] γ can be set to 1 only for episodic tasks, where the length of trajectories are fixed.

[2] Ties are broken randomly, if more than one action maximizes $Q(s,a)$.



(a) (b) (c)

Fig. 2. The GridWorld domain (a), the corresponding policy calculated with a planner assuming deterministic movement model and its true value function (b) and the optimal policy with the perfect model and its value function (c). The task is to navigate from the top left corner highlighted as • to the right bottom corner identified as ⋆. Red regions are off-limit areas where the UAV should avoid. The movement dynamics has 30% noise of moving the UAV to a random free neighboring grid cell. Gray cells are not traversable.

following the incremental natural actor-critic framework [6]. The value of each state-action pair ($Q(s,a)$) is held by the *critic* and is calculated/updated in an identical manner to Sarsa.

III. Problem Statement

In this section, we use a pedagogical example to explain: (1) the effect of unknown noise on the planner's solution, (2) how learning methods can improve the performance and safety of the planner solution, and (3) how the approximate model and the planner solution can be used for faster and safer learning.

A. The GridWorld Domain: A Pedagogical Example

Consider a grid world scenario shown in Fig. 2-(a), in which the task is to navigate a UAV from the top-left corner (•) to the bottom-right corner (⋆). Red areas highlight the danger zones where the UAV will be eliminated upon entrance. At each step the UAV can take any action from the set $\{\uparrow,\downarrow,\leftarrow,\rightarrow\}$. However, due to wind disturbances, there is 30% chance that the UAV is pushed into any unoccupied neighboring cell while executing the selected action. The reward for reaching the goal region and off-limit regions are $+1$ and -1 respectively, while every other move results in -0.001 reward.

Fig. 2-(b) illustrates the policy (shown as arrows) calculated by a planner that is unaware of the wind together with the nominal path highlighted as a gray tube. As expected the path suggested by the planner follows the shortest path that avoids direct passing through off-limit areas. The color of each cell represents the true value of each state (*i.e.,* including the wind) under the planner's policy. Green indicates positive, white indicate zero, and red indicate negative values[3]. Lets focus on the nominal path from the start to the goal. Notice how the value function jumps suddenly each time the policy is followed from an off-limit neighbor cell (*e.g.,* $(8,3) \rightarrow (8,4)$). This drastic change highlights the involved risk in taking those actions in the presence of the wind.

The optimal policy and its corresponding value function and nominal path are shown in Fig. 2-(c). Notice how the optimal policy avoids the risk of getting close to off-limit areas by making wider turns. Moreover, the value function on the nominal path no longer goes through sudden jumps. While the new nominal path is longer, it mitigates the risk better. In fact, the new policy raises the mission success rate from 29% to 80%, while boosting the value of the initial state by a factor of ≈3. Model-free learning techniques such as Sarsa can find the optimal policy through mere interaction, although they require many training examples. More importantly, they might deliberately move the UAV towards off-limit regions to gain information about those areas. If the learner is integrated with the planner, the estimated model can be used to rule out intentional poor decisions. Furthermore, the planner's policy can be used as a starting point for the learner to bootstrap on, reducing the amount of data the learner requires to master the task.

Though simple, the preceding problem is fundamentally similar to more meaningful and practical UAV planning scenarios. The following sections present the technical approach and examines the resulting methods in this toy domain and a more complex multi-UAV planning task where the size of

[3]We set the value for blocked areas to $-\infty$, hence the intense red color

the state space exceeds 200 million state-action pairs.

IV. Technical Approach

This section provides further details of the intelligent cooperative control architecture (*iCCA*), describing the purpose and function of each element. Fig. 3 shows that the consensus-based bundle algorithm (CBBA) [5] is used as the cooperative planner to solve the multi-agent task allocation problem. The learning algorithms used are the natural actor-critic [6] and Sarsa [9] methods. These algorithms use past experience to explore and suggest promising behaviors leading to more favorable outcomes. The performance analysis block is implemented as a risk analysis tool where actions suggested by the learner can be overridden by the baseline cooperative planner if they are deemed too risky. The following sections describe each of these blocks in detail.

A. Cooperative Planner

At its fundamental level, the cooperative planner yields a solution to the multi-agent path planning, task assignment or resource allocation problem, depending on the domain. This means it seeks to fulfill the specific goals of the application in a manner that optimizes an underlying, user-defined *objective function*. Many existing cooperative control algorithms use observed performance to calculate *temporal-difference errors* which drive the objective function in the desired direction [16,17]. Regardless of how it is formulated (e.g. MILP, MDP, CBBA), the cooperative planner, or cooperative control algorithm, is the source for baseline plan generation within *iCCA*. In our work, we focus on the CBBA algorithm. CBBA is a deterministic planner and therefore cannot account for noise or stochasticity in action outcomes, but, as outlined in the following section, the algorithm has several core features that make it particular attractive as a cooperative planner.

1) Consensus-Based Bundle Algorithm: CBBA is a decentralized auction protocol that produces conflict-free assignments that are relatively robust to disparate situational awareness over the network. CBBA consists of iterations between two phases: In the first phase, each vehicle generates a single ordered bundle of tasks by sequentially selecting the task giving the largest marginal score. The second phase resolves inconsistent or conflicting assignments through local communication between neighboring agents.

In the second phase, agent i sends out to its neighboring agents two vectors of length N_t: the winning agents vector $\mathbf{z}_i \in \mathcal{I}_t^N$ and the winning bids vector $\mathbf{y}_i \in \mathbb{R}_+^{N_t}$. The j-th entries of the $\mathbf{z}_i$ and $\mathbf{y}_i$ indicate who agent i thinks is the best agent to take task j, and what is the score that agent gets from task j, respectively. The essence of CBBA is to enforce every agent to agree upon these two vectors, leading to agreement on some conflict-free assignment regardless of inconsistencies in situational awareness over the team.

There are several core features of CBBA identified in [5]. First, CBBA is a decentralized decision architecture. For a large team of autonomous agents, it would be too restrictive to assume the presence of a central planner (or server) with which every agent communicates. Instead, it is more natural for each agent to share information via local communication with its neighbors. Second, CBBA is a polynomial-time algorithm. The worst-case complexity of the bundle construction is $\mathcal{O}(N_t L_t)$ and CBBA converges within $\max\{N_t, L_t N_a\} D$ iterations, where N_t denotes the number of tasks, L_t the maximum number of tasks an agent can win, N_a the number of agents and D is the network diameter, which is always less than N_a. Thus, the CBBA methodology scales well with the size of the network and/or the number of tasks (or equivalently, the length of the planning horizon). Third, various design objectives, agent models, and constraints can be incorporated by defining appropriate scoring functions. It is shown in [5] that if the resulting scoring scheme satisfies a certain property called *diminishing marginal gain*, a provably good feasible solution is guaranteed.

While the scoring function primarily used in [5] was a time-discounted reward, a more recent version of the algorithm [18] handles the following extensions while preserving convergence properties:

- Tasks that have finite time windows of validity
- Heterogeneity in the agent capabilities
- Vehicle fuel cost

Starting with this extended version of CBBA, this research adds additional constraints on fuel supply to ensure agents cannot bid on task sequences that require more fuel than they have remaining or that would not allow them to return to base upon completion of the sequence.

B. Risk/Performance Analysis

One of the main reasons for cooperation in a cooperative control mission is to minimize some global cost, or objective function. Very often this objective involves time, risk, fuel, or other similar physically-meaningful quantities. The purpose of the performance analysis module is to accumulate observations, glean useful information buried in the noise, categorize it and use it to improve subsequent plans. In other words, the performance analysis element of iCCA attempts to improve agent behavior by diligently studying its own experiences [14] and compiling relevant signals to drive the learner and/or the planner.

The use of such feedback within a planner is of course not new. In fact, there are very few cooperative planners which do not employ some form of measured feedback. In this research, we implemented this module as a risk analysis element where candidate actions are evaluated for risk level. Actions deemed too "risky" are replaced with another action of lower risk. The next section, detail the process of overriding risky actions. It is important to note that the risk analysis and learning algorithms are coupled within an MDP formulation, as shown in Fig. 3, which implies a fully observable environment.

C. Learning Algorithm

A focus of this research is to integrate a learner into *iCCA* that suggests candidate actions to the cooperative planner that

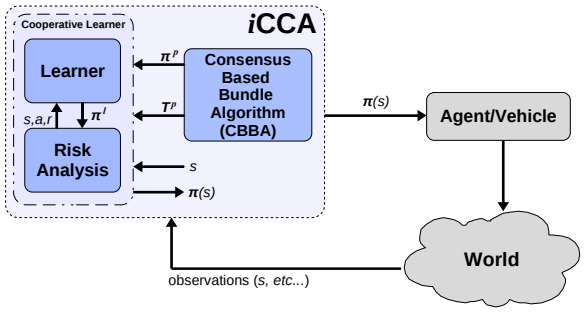


Fig. 3. iCCA framework as implemented. CBBA planner coupled with risk analysis and reinforcement learning modules, where the latter two elements are formulated within an MDP.

it sees as beneficial. Suggested actions are generated by the learning module through the learned policy. In our previous work [4], we integrated natural actor-critic through iCCA framework. We refer to this algorithm as Cooperative Natural Actor-Critic (CNAC). As a reinforcement learning algorithm, CNAC introduces the key concept of *bounded exploration* such that the learner can explore the parts of the world that may lead to better system performance while ensuring that the agent remains safe within its operational envelope and away from states that are known to be undesirable. In order to facilitate this bound, the risk analysis module inspects all suggested actions of the actor, and replaces the risky ones with the baseline CBBA policy. This process guides the learning away from catastrophic errors. In essence, the baseline cooperative control solution provides a form of "prior" over the learner's policy space while acting as a backup policy in the case of an emergency.

Algorithm 1 illustrates CNAC in more detail. In order to encourage the policy to initially explore solutions similar to the planner solution, preferences for all state-action pairs, $P(s,a)$, on the nominal trajectory calculated by the planner are initialized to a fixed number $\xi \in \Re^+$. All other preferences are initialized to zero. As actions are pulled from the policy for implementation, they are evaluated for their safety by the risk analysis element. If they are deemed unsafe (e.g. may result in a UAV running out of fuel), they are replaced with the action suggested by the planner (π^p). Furthermore, the preference of taking the risky action in that state is reduced by parameter ξ, therefore dissuading the learner from suggesting that action again, reducing the number of "emergency overrides" in the future. Finally, both the critic and actor parameters are updated.

Previously, we employed a risk analysis component which had access to the exact world model dynamics. Moreover, we assumed that the transition model related to the risk calculation was deterministic (*e.g.,* movement and fuel burn did not involve uncertainty). In this paper, we introduce a new risk analysis scheme which uses the planner's inner model, which can be stochastic, to mitigate risk. Algorithm 2, explains this new risk analysis process. We assume the existence of the $constrained$ function: $\mathcal{S} \rightarrow \{0,1\}$, which indicates if being in a particular state is allowed or not. Risk

Algorithm 1: Cooperative Natural Actor-Critic (CNAC)

Input: π^p, ξ
Output: a
$a \sim \pi^{AC}(s,a)$
if not *safe*(s,a) **then**
  $P(s,a) \leftarrow P(s,a) - \xi$
  $a \leftarrow \pi^p$
$Q(s,a) \leftarrow Q(s,a) + \alpha\delta_t(Q)$
$P(s,a) \leftarrow P(s,a) + \alpha Q(s,a)$

Algorithm 2: safe

Input: s, a
Output: $isSafe$
$risk \leftarrow 0$
for $i \leftarrow 1$ **to** M **do**
  $t \leftarrow 1$
  $s_t \sim T^p(s,a)$
  while not $constrained(s_t)$ **and not** $isTerminal(s_t)$ **and** $t < H$ **do**
    $s_{t+1} \sim T^p(s_t, \pi^p(s_t))$
    $t \leftarrow t+1$
  $risk \leftarrow risk + \frac{1}{i}(constrained(s_t) - risk)$
$isSafe \leftarrow (risk < \psi)$

is defined as the probability of visiting any of the constrained states. The core idea is to use Monte-Carlo sampling to estimate the risk level associated with the given state-action pair if planner's policy is applied thereafter. This is done by simulating M trajectories from the current state s. The first action is the suggested action a, and the rest of actions come from the planner policy, π^p. The planner's inner transition model, T^p, is utilized to sample successive states. Each trajectory is bounded to a fixed horizon H and the risk of taking action a from state s is estimated by the probability of a simulated trajectory reaching a risky state within horizon H. If this risk is below a given threshold, ψ, the action is deemed to be safe.

The initial policy of actor-critic type learners is biased quite simply as they parameterize the policy explicitly. For learning schemes that do not represent the policy as a separate entity, such as Sarsa, integration within iCCA framework is not immediately obvious. In this paper, we present a new approach for integrating learning approaches without an explicit actor component. Our idea was motivated by the concept of the R_{max} algorithm [20]. We illustrate our approach through the parent-child analogy, where the planner takes the role of the parent and the learner takes the role of the child. In the beginning, the child does not know much about the world, hence, for the most part s/he takes actions advised by the parent. While learning from such actions, after a while, the child feels comfortable about taking a self-motivated actions as s/he has been through the same situation many times. Seeking permission from the parent, the child could take the action if the parent thinks the action

Algorithm 3: Cooperative Learning

Input: $N, \pi^p, s, learner$
Output: a
$a \leftarrow \pi^p(s)$
$\pi^l \leftarrow learner.\pi$
$knownness \leftarrow \min\{1, \frac{count(s,a)}{N}\}$
if $rand() < knownness$ **then**
  $a' \sim \pi^l(s, a)$
  if $safe(s, a')$ **then**
    $a \leftarrow a'$
else
  $count(s, a) \leftarrow count(s, a) + 1$
$learner.update()$

is not unsafe. Otherwise the child should follow the action suggested by the parent.

Algorithm 3 details the process. On every step, the learner inspects the suggested action by the planner and estimates the knownness of the state-action pair by considering the number of times that state-action pair has been experienced. The N parameter controls the shift speed from following the planner's policy to the learner's policy. Given the knownness of the state-action pair, the learner probabilistically decides to select an action from its own policy. If the action is deemed to be safe, it is executed. Otherwise, the planner's policy overrides the learner's choice. If the planner's action is selected, the knownness count of the corresponding state-action pair is incremented. Finally the learner updates its parameter depending on the choice of the learning algorithm. What this means, however, is that state-action pairs explicitly forbidden by the baseline planner will not be intentionally visited. Also, notice that any control RL algorithm, even the actor-critic family of methods, can be used as the input to Algorithm 3.

V. Experimental Results

This section compares the empirical performance of cooperative-NAC and cooperative-Sarsa with pure learning and pure planning methods in the GridWorld example mentioned in Section III, and a multi-UAV mission planning scenario where both dynamics and reward models are stochastic. For the GridWorld domain, the optimal solution was calculated using dynamic programming.[4] As for the planning, the CBBA algorithm was executed online given the expected deterministic version of both domains. Pure planning results are averaged over $10,000$ Monte Carlo simulations. For all learning methods, the best learning rates were calculated by:

$$\alpha_t = \alpha_0 \frac{N_0 + 1}{N_0 + \text{Episode\#}^{1.1}}.$$

The best α_0 and N_0 were selected through experimental search of the sets of $\alpha_0 \in \{0.01, 0.1, 1\}$ and $N_0 \in \{100, 1000, 10^6\}$ for each algorithm and scenario. The best preference parameter, ξ, for NAC and CNAC were empirically found from the set $\{1, 10, 100\}$. τ was set to 1. Similarly, the knownness parameter, N, for CSarsa was selected out of $\{10, 20, 50\}$. The exploration rate (ϵ) for Sarsa and CSarsa was set to 0.1. All learning method results were averaged over 60 runs. Error bars represents the 95% confidence intervals on each side.

A. GridWorld Domain

Fig. 4-(a) compares the performance of CNAC, NAC, the baseline planner (hand-coded) policy, and the expected optimal solution in the pedagogical GridWorld domain shown in Fig. 2. The X-axis shows the number of steps the agent executed an action, while the Y-axis highlights the cumulative rewards of each method after each $1,000$ steps. Notice how CNAC achieves better performance after $6,000$ steps by navigating farther from the danger zones. NAC, on the other hand, could not outperform the planner by the end of $10,000$ steps.

In Fig. 4-(b), the NAC algorithm was substituted with Sarsa. Moreover, the interaction between the learner and the planner follows Algorithm 3. The same pattern of behavior can be observed, although both CSarsa and Sarsa have a better overall performance compared to CNAC and NAC respectively. We conjecture that Sarsa learned faster than NAC because Sarsa's policy is embedded in the $Q-$value function, whereas NAC's policy requires another level of learning for the policy on the top of learning the $Q-$value function. While, in this domain, the performance of Sarsa was on par with CSarsa at the end of training horizon, we will see that this observation does not hold for large domains indicating the importance of cooperative learners for more realistic problems.

B. Multi-UAV Planning Scenario

Fig. 5 depicts of the mission scenario of interest where a team of two fuel-limited UAVs cooperate to maximize their total reward by visiting valuable target nodes in the network. The base is highlighted as node 1 (green circle), targets are shown as blue circles and agents as triangles. The total amount of fuel for each agent is highlighted by the number inside each triangle. For those targets with an associated reward it is given as a positive number nearby. The constraints on the allowable times when the target can be visited are given in square brackets and the probability of receiving the known reward when the target is visited is given in the white cloud nearest the node.[5] Each reward can be obtained only once and traversing each edge takes one fuel cell and one time step.

UAVs are allowed to loiter at any node. The fuel burn for loitering action is also one unit, except for any UAV at the base, where they are assumed to be stationary and thus there is no fuel depletion. The mission horizon was set to 10 time steps. If UAVs are not at base at the end of the mission

[4] Unfortunately, calculating an optimal solution for the UAV scenario was estimated to take about 20 days. We are currently optimizing the code in order to prepare the optimal solution for the final paper submission.

[5] If two agents visit a node at the same time, the probability of visiting the node would increase accordingly.

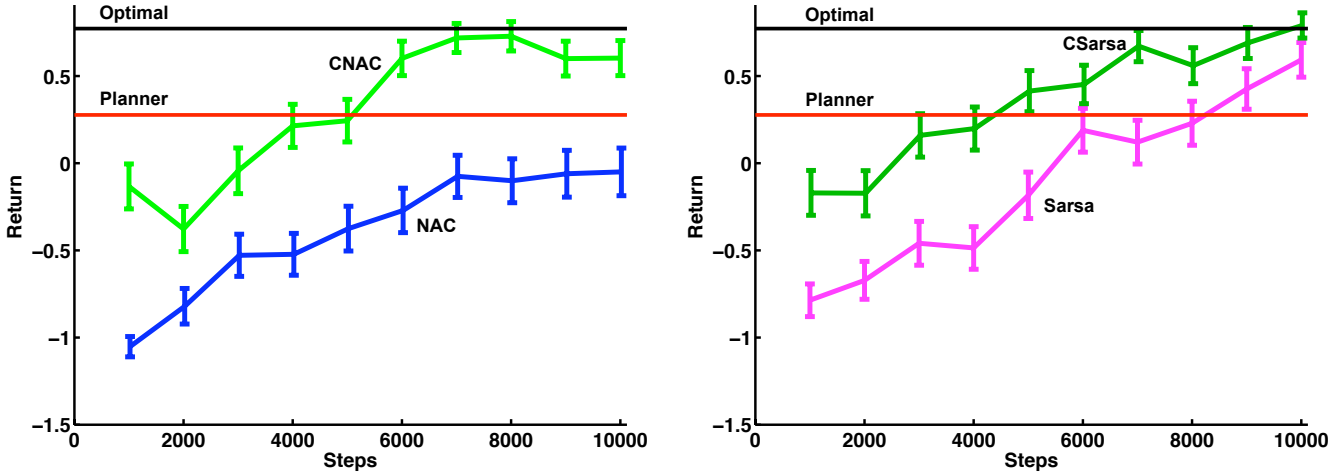


Fig. 4. In the pedagogical GridWorld domain, the performance of the optimal solution is given in black. The solution generated by the deterministic planner is shown in red. In addition, the performance of NAC, CNAC (left) and Sarsa and CSarsa (right) are shown. It is clear that the cooperative learning algorithms (CNAC and CSarsa) outperform their non-cooperative counterparts and eventually outperform the baseline planner when given a sufficient number of interactions. This result motivates the application of the cooperative learning algorithms to more complex domains, such as the Multi-UAV planning scenario.

horizon, or crash due to fuel depletion, a penalty of -800 is added for that trial. In order to test our new risk mitigation approach, we added uncertainty to the movement of each UAV by adding 5% chance of edge traverse failure. Notice that our previous risk analyzer [4,8] could not handle such scenarios, as it assumed the existence of an oracle knowing catastrophic actions in each state.

Figs 6-(a),(b) show the results of the same battery of algorithms used in the GridWorld domain applied to the UAV mission planning scenario. In this domain, we substituted the hand-coded policy with the CBBA algorithm. Fig. 6-(a) represents the performance of each method after 10^5 steps of interactions, while Fig. 6-(b) depicts the risk of executing the corresponding policy. At the end of training, both NAC and Sarsa failed to avoid the crashing scenarios, thus yielding low performance and more than a 90% probability of failure. This observation coincides with our previous experiments with this domain where the movement model was noise free [4], highlighting the importance of biasing the policy of learners in large domains. On average, CNAC and CSarsa improved the performance of CBBA by about 15% and 30% respectively. At the same time they reduced the probability of failure by 6% and 8%.

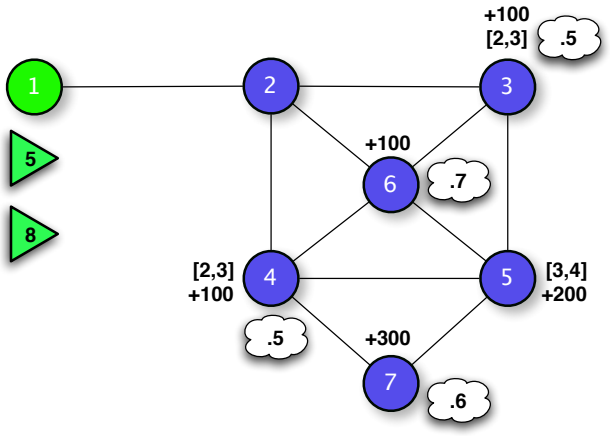
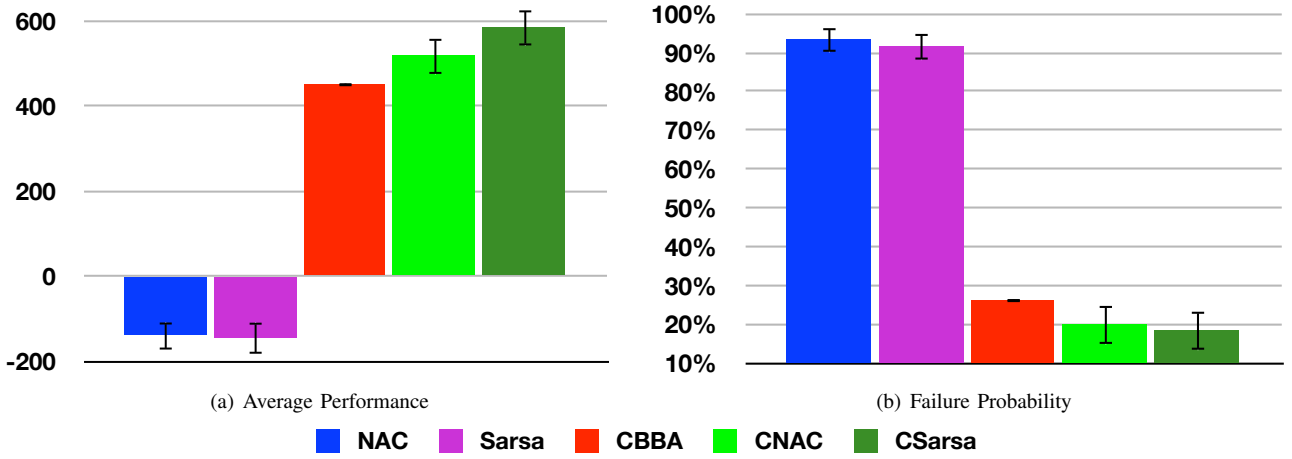


Fig. 5. The mission scenarios of interest: A team of two UAVs plan to maximize their cumulative reward along the mission by cooperating to visit targets. Target nodes are shown as circles with rewards noted as positive values and the probability of receiving the reward shown in the accompanying cloud. Note that some target nodes have no value. Constraints on the allowable visit time of a target are shown in square brackets.

VI. Conclusions

This paper extended the capability of the previous work on merging learning and cooperative planning techniques through two innovations: (1) the risk mitigation approach has been extended to stochastic system dynamics where the exact world model is not known, and (2) learners without a separate policy parameterization can be integrated in the iCCA framework through the cooperative learning algorithm. Using a pedagogical GridWorld example, we explained how the emerging algorithms can improve the performance of existing planners. Simulation results verified our hypothesis in the GridWorld example. We finally tested our algorithms in a multi-UAV planning scenario including stochastic transition and rewards models, where none of the uncertainties were known a priori. On average, the new cooperative learning methods boosted the performance of CBBA up to 30%, while reducing the risk of failure up to 8%.

For future work, we are interested in increasing the learning speed of cooperative learners by taking advantage of function approximators. Function approximation allows generalization among the values of similar states often boosting the learning speed. However, finding a proper function approximator for a problem is still an active area of research, as poor approximations can render the task unsolvable, even with infinite amount of data. While in this work, we assumed



(a) Average Performance (b) Failure Probability

Fig. 6. Results of NAC, Sarsa, CBBA, CNAC, and CSarsa algorithms at the end of the training session in the UAV mission planning scenario. Cooperative learners (CNAC, CSarsa) perform very well with respect to overall reward and risk levels when compared with the baseline CBBA planner and the non-cooperative learning algorithms.

a static model for the planner, a natural extension is to adapt the model with the observed data. We foresee that this extension will lead to a more effective risk mitigation approach as the transition model used for Monte-Carlo sampling resembles the actual underlying dynamics with more observed data.

VII. Acknowledgments

This research was supported in part by AFOSR (FA9550-09-1-0522) and Boeing Research & Technology.

References

[1] J. Kim and J. Hespanha, "Discrete approximations to continuous shortest-path: Application to minimum-risk path planning for groups of UAVs," in *42nd IEEE Conference on Decision and Control, 2003. Proceedings*, vol. 2, 2003.

[2] R. Weibel and R. Hansman, "An integrated approach to evaluating risk mitigation measures for UAV operational concepts in the NAS," *AIAA-2005-6957*, 2005.

[3] J. Redding, A. Geramifard, H.-L. Choi, and J. P. HowS, "Actor-critic policy learning in cooperative planning," in *AIAA Guidance, Navigation, and Control Conference (GNC)*, 2010 (AIAA-2010-7586).

[4] J. Redding, A. Geramifard, A. Undurti, H. Choi, and J. How, "An intelligent cooperative control architecture," in *American Control Conference*, 2010.

[5] H.-L. Choi, L. Brunet, and J. P. How, "Consensus-based decentralized auctions for robust task allocation," *IEEE Trans. on Robotics*, vol. 25 (4), pp. 912 – 926, 2009.

[6] S. Bhatnagar, R. S. Sutton, M. Ghavamzadeh, and M. Lee, "Incremental natural actor-critic algorithms." in *NIPS*, J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, Eds. MIT Press, 2007, pp. 105–112. [Online]. Available: http://dblp.uni-trier.de/db/conf/nips/nips2007.html#BhatnagarSGL07

[7] J. Peters and S. Schaal, "Natural actor-critic," *Neurocomput.*, vol. 71, pp. 1180–1190, March 2008. [Online]. Available: http://portal.acm.org/citation.cfm?id=1352927.1352986

[8] J. Redding, A. Geramifard, H. Choi, and J. How, "Actor-critic policy learning in cooperative planning," in *AIAA Guidance, Navigation and Control Conference (to appear)*, 2010.

[9] G. A. Rummery and M. Niranjan, "Online Q-learning using connectionist systems (tech. rep. no. cued/f-infeng/tr 166)," *Cambridge University Engineering Department*, 1994.

[10] R. A. Howard, "Dynamic programming and markov processes," 1960.

[11] M. L. Puterman, "Markov decision processes," 1994.

[12] M. L. Littman, T. L. Dean, and L. P. Kaelbling, "On the complexity of solving markov decision problems," in *In Proc. of the Eleventh International Conference on Uncertainty in Artificial Intelligence*, 1995, pp. 394–402.

[13] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, pp. 99–134, 1998.

[14] S. Russell and P. Norvig, "Artificial Intelligence, A Modern Approach," 2003.

[15] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning*, vol. 3, pp. 9–44, 1988. [Online]. Available: citeseer.csail.mit.edu/sutton88learning.html

[16] R. Murphey and P. Pardalos, *Cooperative control and optimization*. Kluwer Academic Pub, 2002.

[17] D. Bertsekas and J. Tsitsiklis, *Neuro-Dynamic Programming*. Belmont, MA: Athena Scientific, 1996.

[18] S. Ponda, J. Redding, H.-L. Choi, J. P. How, M. A. Vavrina, and J. Vian, "Decentralized planning for complex missions with dynamic communication constraints," in *American Control Conference (ACC)*, July 2010.

[19] J. Redding, A. Geramifard, A. Undurti, H. Choi, and J. How, "An intelligent cooperative control architecture," in *American Control Conference (ACC) (to appear)*, 2010.

[20] R. I. Brafman and M. Tennenholtz, "R-max - a general polynomial time algorithm for near-optimal reinforcement learning," *Journal of Machine Learning*, vol. 3, pp. 213–231, 2001.