Actor-Critic Policy Learning in Cooperative Planning

Actor-Critic Policy Learning in Cooperative Planning

Josh Redding[*], Alborz Geramifard[†], Han-Lim Choi[‡], Jonathan P. How[§]

Aerospace Controls Laboratory

Massachusetts Institute of Technology

Cambridge, MA 02139

In this paper, we introduce a method for learning and adapting cooperative control strategies in real-time stochastic domains. Our framework is an instance of the intelligent cooperative control architecture (iCCA)[1]. The agent starts by following the "safe" plan calculated by the planning module and incrementally adapting its policy to maximize the cumulative rewards. Actor-critic and consensus-based bundle algorithm (CBBA) were employed as the building blocks of the iCCA framework. We demonstrate the performance of our approach by simulating limited fuel unmanned aerial vehicles aiming for stochastic targets. In one experiment where the optimal solution can be calculated, the integrated framework boosted the optimality of the solution by an average of %10, when compared to running each of the modules individually, while keeping the computational load within the requirements for real-time implementation.

I. Introduction

Planning for heterogeneous teams of mobile, autonomous, health-aware agents in uncertain and dynamic environments is a challenging problem. In such a setting, the agents are simultaneously engaged and continuously interact with each other, their surroundings and with potential threats. They may encounter evasive targets and need to reason through adversarial actions with insufficient data. Or, agents may receive delayed, lossy and contaminated communications or experience sensor and actuator failures. On top of these challenges, autonomous agents must be robust to unmodeled dynamics or parametric uncertainties while remaining capable of performing their advertised range of tasks.

Although much work has been done in the area of multi-agent planning in uncertain environments[2–5], key gaps in the current literature include:

[*]Ph.D. candidate, Aerospace Controls Laboratory, Massachusetts Institute of Technology jredding@mit.edu

[†]Ph.D. candidate, Aerospace Controls Laboratory, Massachusetts Institute of Technology agf@mit.edu

[‡]Postdoctoral Fellow, Aerospace Controls Laboratory, Massachusetts Institute of Technology hanlimc@mit.edu

[§]Professor, MIT Dept. of Aeronautics and Astronautics, Associate Fellow AIAA jhow@mit.edu

- *How to improve planner performance over time in the face of uncertainty and a dynamic world?*
- *How to use current knowledge and past observations to become both robust to likely failures and intelligent with respect to unforeseen future events?*

In this research, we focus primarily on the former question. That is, we are interested specifically in improving the performance of the system over time in an uncertain world. To facilitate this, we adopt the intelligent cooperative control architecture (iCCA) previously introduced by the authors[1].

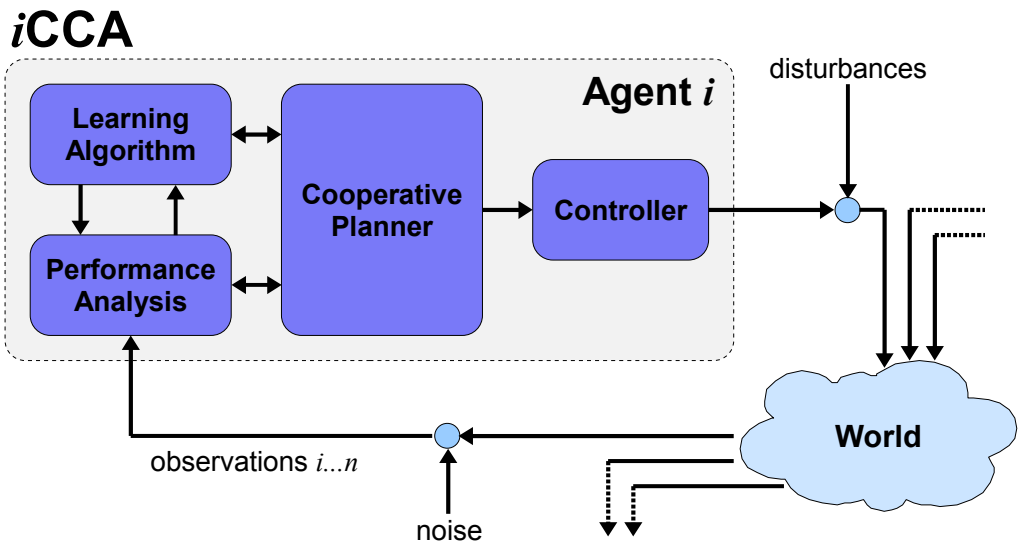


Figure 1: An **i**ntelligent **C**ooperative **C**ontrol **A**rchitecture, a framework for the integration of cooperative control algorithms and machine learning techniques.

As seen in Figure 1, iCCA is comprised of a cooperative planner, a learner, a metric for performance-to-date. Each of these elements is interconnected and plays a key role in the overall architecture. For this research, we use the consensus-based bundle algorithm (CBBA)[6] as the cooperative planner to solve the multi-agent task allocation problem. For the learning algorithm, we implemented an actor-critic reinforcement learner which uses information regarding performance to explore and suggest new behaviors that would likely lead to more favorable outcomes than the current behavior would produce. 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. This synergistic planner-learner relationship yields a "safe" policy in the eyes of the planner, upon which the learner can only improve. Ultimately, this relationship will help to bridge the gap to successful and intelligent execution in real-world missions.

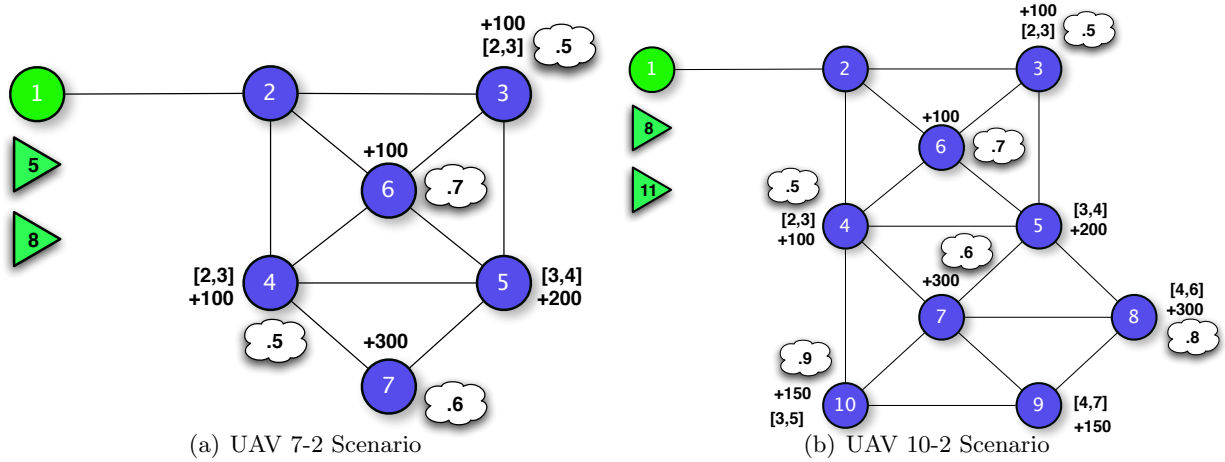
In the remainder of this paper, we detail the integration of learning with cooperative control and show how the marriage of these two fields can result in an intelligent, adaptable planning scheme in the context of teams of autonomous agents in uncertain environments. We first formulate an instance of the iCCA framework and introduce a method for learning and adapting cooperative

control strategies in real-time stochastic domains. An agent starts by following a "safe" plan calculated by its planning module and then incrementally adapts the associated policy in order to maximize the cumulative rewards via actor-critic reinforcement learning. We then demonstrate the performance of our approach by simulating limited fuel unmanned aerial vehicles aiming for stochastic targets. We proceed as follows: We motivate and formally state the multi-agent planning problem in Section II. The specifics of the proposed architecture are then detailed in Section III and followed by a discussion of simulation results in Section IV and lastly by a summary of conclusions.

II. Problem Statement

Having introduced the general problem, outlined a few key research gaps, and proposed our solution, we now prepare to dive a little deeper by formally presenting the problem we aim to solve and by giving some useful background information. We first outline a small, yet difficult, multi-agent scenario in Section II.A where traditional cooperative control techniques tend to struggle, due to the presence of significant uncertainties. Some relevant background information is given in Section **??**, followed by the formulation of the planning problem associated with the scenario of interest in Section II.C along with further details of the proposed solution approach, including how it addresses the research gap of interest.

II.A. Problem Scenario



(a) UAV 7-2 Scenario (b) UAV 10-2 Scenario

Figure 2: 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.

Here we outline the scenarios in which we developed and tested each of the modules in the iCCA framework. Referring to Figures 2, we see a depiction of the mission scenarios 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.[a] Each reward can be obtained only once and traversing each edge takes one fuel cell and one time step. We also allow UAVs to loiter at any of the nodes indefinitely if, for some reason, they believe that to be the "optimal" action. The fuel burn for loitering action is also one unit, except for any UAVs at the base, where they are assumed to be stationary and their fuel level is therefore not depleted. The mission horizon was set to 8 time steps for UAV 7-2 scenario and 11 for the UAV 10-2 scenario.

II.B. Markov Decision Processes

As the scenarios above are modeled each as a multi-agent Markov Decision Process (MDP)[7–9], we now provide some relevant background. The MDP framework provides a general formulation for sequential planning under uncertainty. 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.[b] 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, transit 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:

$$\pi^* = argmax_a Q^{\pi^*}(s, a)$$

II.C. MDP Formulation

Here, we formulate the scenarios of interest into MDP framework, as described above.

II.C.1. State Space $\mathcal{S}$

We formulated the state space as $[N_1, F_1, \ldots, N_n, F_n, V_1, \ldots, V_m, t]^T$, where N_i and F_i are integer values highlighting the location and the remaining fuel respectively for UAV i ($i \in 1 \ldots n$). V_j is a

[a] If two agents visit a node at the same time, the probability of visiting the node would increase accordingly.
[b] γ can be set to 1 only for episodic tasks, where the length of trajectories are fixed.

single bit signaling if node j has been visited before, where $(j \in 1 \dots m)$, and t is the current time step. There are n UAVs and m nodes participating in the scenario.

II.C.2. Action Space $\mathcal{A}$

Action space is $[N_1^+, \dots, N_n^+]$ where N_i^+ is the node to which the agent is traveling, or where it will be at the next time interval.

II.C.3. Transition Function $\mathcal{P}_{ss'}^a$

The transition function is deterministic for the UAV position, fuel consumption, and time variables of the state space, while it is stochastic for the visited list of targets. The detailed derivation of the complete transition function should be trivial following the corresponding graph in Figure 2. That is, transitions are allowed between nodes for which there is an edge on the graph.

II.C.4. Reward Function $\mathcal{R}_{ss'}^a$

The reward on each time step is stochastic and calculated as the sum of rewards from visiting new desired targets minus the total burnt fuel cells on the last move. Notice that a UAV receives the target reward only if it lands on an unvisited rewarding node and lucky enough to obtain the reward. In that case, the corresponding visibility bit will turn on, and the agent receive the reward. The crash penalty equals to the negative sum of rewards at all nodes for both scenarios in order to prioritize safety over visiting targets. It occurs when any UAV runs out of fuel or is not at the base by the end of the mission horizon.

III. iCCA

In this section, we detail our instance of the intelligent cooperative control architecture (iCCA), describing the purpose and function of each element and how the framework as a whole fits together with the MDP formulated in the previous section.

As seen in Figure 3, we use the consensus-based bundle algorithm (CBBA)[6] as the cooperative planner to solve the multi-agent task allocation problem. For the learning algorithm, we implemented an actor-critic reinforcement learner which uses information regarding performance to explore and suggest new behaviors that would likely lead to more favorable outcomes than the current behavior would produce. 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. In the sections that follow, each of these blocks is described in a bit more detail.

III.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

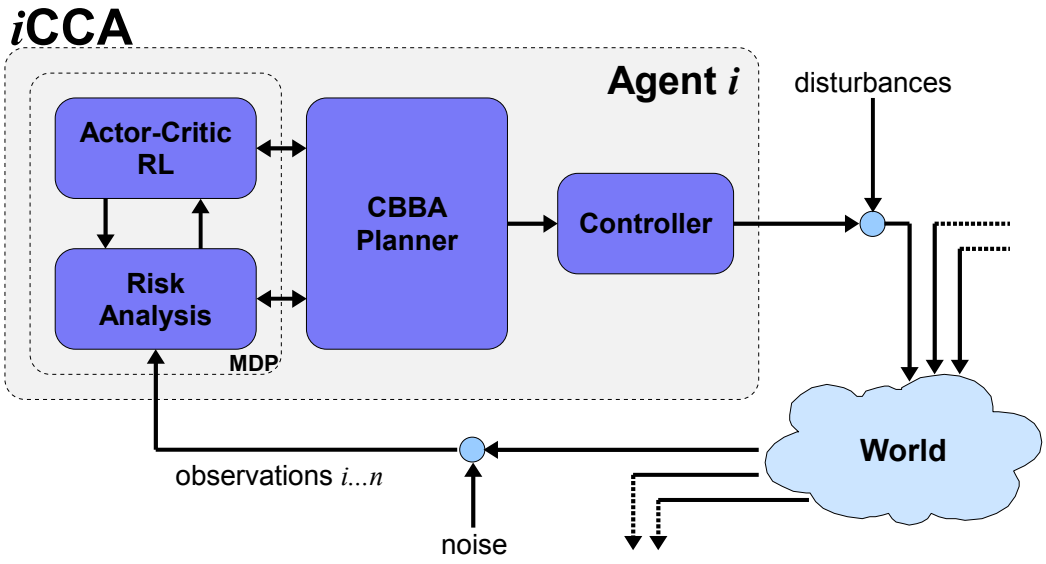


Figure 3: iCCA framework as implemented. CBBA planner and risk analysis and the actor-critic learner formulated within an MDP.

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[5,10]. Regardless of how it is formulated (e.g. MILP, MDP, CBBA, etc...), the cooperative planner, or cooperative control algorithm, is the source for baseline plan generation within *iCCA*.

In this research, we implemented a decentralized auction protocol called the consensus-based bundle algorithm as the cooperative planner. The following section details this approach.

III.A.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 local communication round, some 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 [6]. 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 [6] 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 [6] was a time-discounted reward, a more recent version of the algorithm is due to Ponda[11] and handles the following extensions while preserving convergence properties:

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

This research uses this extended version of CBBA as the cooperative planner and adds to it additional constraints on fuel supply to ensure agents cannot bid on task sequences that require more fuel than they have remaining.

III.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[3] 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 of lower risk. The details of this feature are given in the following section.

III.C. Learning Algorithm

Although learning has many forms, iCCA provides a minimally restrictive framework where the contributions of the learner to fall into either of two categories:

1. Assist the cooperative planner by adapting to parametric uncertainty of internal models

2. Suggesting candidate actions to the cooperative planner that the learner sees as beneficial

The focus of our research is on the latter category, where suggested actions are generated by the learning module through the learned policy. 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)[12] is a traditional policy evaluation method in which the current $Q(s,a)$ is adjusted based on the temporal difference error akin to the gradient vector. Given $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$ and the current value estimates, the 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:

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

where α is the learning rate. We chose to formulate the learner as an actor-critic type, where TD method used for the critic section. As for the actor, the policy was represented using Gibbs softmax method:

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

in which $P(s,a)$ is the preference of taking action a in state s, and $\tau \in (0, \infty]$ is the temperature parameter acting as a knob shifting from greedy towards random action selection. Since we use a tabular representation the actor update amounts to:

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

following the incremental natural actor-critic framework[13]. We initialized the actor's policy with the action preferences generated by the baseline CBBA planner. As actions are pulled from the policy for implementation, they are evaluated for risk level and can be overridden by CBBA if the action would lead the agent into an undesirable configuration, such as crashing or running out of fuel. As the agent implements the policy, the critic receives rewards for the actor's actions.

As a reinforcement learning algorithm, the actor-critic element of iCCA 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 inspect all suggestive 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.

A canonical failure of learning algorithms in general, is that negative information is extremely useful in terms of the value of information it provides. We therefore introduce the notion of a "virtual reward". In this research, the virtual reward is a large negative value delivered by the risk analysis module to the learner for risky actions suggested by the actor and pruned by the risk analysis module. When this virtual reward is delivered, the learner associates it with the previously suggested action, therefore dissuading the learner from suggesting that action again, reducing the number of "emergency overrides" in the future.

IV. Experimental Results

In this section, we present and discuss simulation results for the scenarios described in Section II.A. For the UAV 7-2 scenario, we were able to solve the problem using backward dynamic programing in order to use this solution as the baseline for the optimality.[c] Unfortunately, calculating an optimal solution was not feasible for the UAV 10-2 case, with about 9 billion state action pairs. However, we ran the CBBA algorithm online on the expected deterministic version of both scenarios on each step for 10,000 episodes. Finally, we empirically searched for the best learning rates for the Actor-Critic and iCCA methods where the learning rate was 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. Table 1 shows the best performing values. For all experiments, we set the preference of the advised CBBA state-action pairs to 100. Risky actions resulted in -100 preference for the state-actions pairs in the actor. τ was set to 1 for the actor.

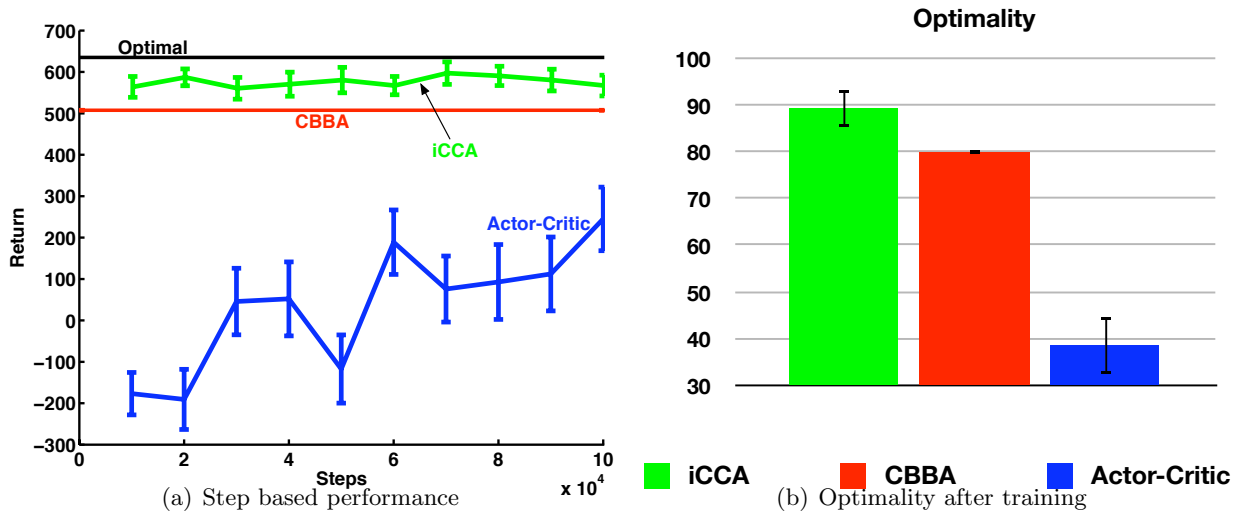
	UAV 7-2		UAV 10-2	
Algorithm	α_0	N_0	α_0	N_0
Actor-Critic	0.1	$1,000,000$	0.1	$1,000$
iCCA	0.01	100	0.01	100

Table 1: The best α_0 and N_0 found empirically out of 9 configurations for each algorithm and scenario.

Figure 4-(a) depicts the performance of iCCA and Actor-Critic averaged over 60 runs in the UAV 7-2 scenario. The Y-axis shows the cumulative reward, while the X-axis represents the number of interactions. Each point on the graph is the result of running the greedy policy with respect to the existing preferences of the actor. For iCCA, risky moves again were replaced by the CBBA baseline solution. Error bars represent the standard error with %90 confidence interval. In order to show the relative performance of these methods with offline techniques, both the optimal and CBBA solutions are highlighted as horizontal lines. It is clear that the actor-critic performs much

[c] This computation took about a day to calculate all expected values over more than 100 million state action pairs. Hence this approach can not be easily scaled for larger sizes of the problem.

better when wrapped into the iCCA framework and performs better than CBBA alone. The reason is that CBBA provides a good starting point for the actor-critic to explore the state space, while the risk analyzer filters risky actions of the actor leading into catastrophic situations. Figure 4-(b) shows the optimality of iCCA and Actor-Critic after 10^5 steps of interaction with the domain and the averaged optimality of CBBA through $10,000$ trials. Notice how the integrated algorithm could on average boost the best individual optimality performance of both individual components by %10.



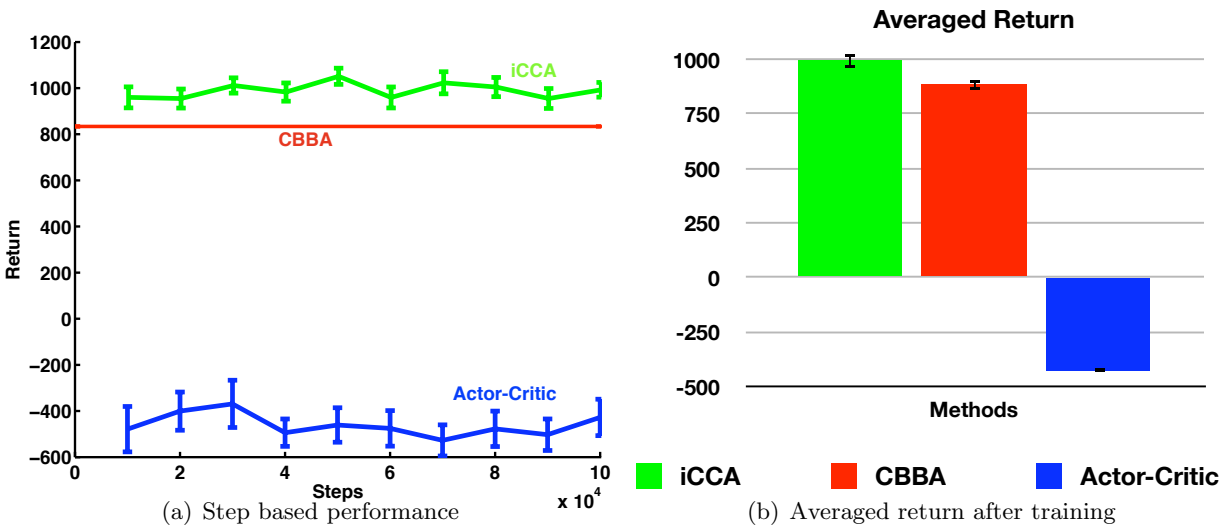
(a) Step based performance (b) Optimality after training

Figure 4: A comparison of the collective rewards received in UAV 7-2 scenario when strictly following plans generated by CBBA alone, actor-critic reinforcement learning outside of the iCCA environment, i.e. without initialization and guidance from CBBA, and the result when these are coupled via the iCCA framework are all compared against the optimal performance as calculated via dynamic programming

For the second set of results, we performed the same set of runs in the UAV 10-2 scenario. Figure 5-(a) depicts the performance of actor-critic, CBBA, and iCCA algorithms in this domain averaged over 30 runs. Since the size of the state-action pairs for this domain is about 9 billion, we could not take advantage of the dynamic programing approach to calculate the optimal solution. Since the state space scaled by a huge factor compared to UAV 7-2 scenario, actor-critic method had a hard time to find a sensible policy even after 10^5 steps. Online CBBA still could find a good policy to the approximated problem. When both CBBA and actor-critic put together through iCCA framework, the agent could achieve a better performance even early on after 10^4 steps. Figure 5-(b) shows the averaged performance of each method at the end of the learning phase. Notice that iCCA again could boost the performance of CBBA solution.

V. Conclusions

In conclusion, we introduced a method for learning and adapting cooperative control strategies in real-time stochastic domains. Our framework of choice was an instance of the intelligent cooper-



(a) Step based performance

(b) Averaged return after training

Figure 5: A comparison of the collective rewards received in UAV 10-2 scenario when strictly following plans generated by CBBA alone, actor-critic reinforcement learning outside of the iCCA environment, i.e. without initialization and guidance from CBBA, and the result when these are coupled via the iCCA framework.

ative control architecture (iCCA) presented in [1]. A "safe" plan was generated by the Consensus-Based Bundle Algorithm[6], which initialized a policy which was then incrementally adapted by a natural actor-critic learning algorithm to increase planner performance over time. We successfully demonstrated the performance of our approach by simulating limited-fuel UAVs aiming for stochastic targets in two scenarios involving uncertainty.

VI. Future Work

Perhaps one of the main drawbacks of our proposed realization of the iCCA framework was the use of hash tables for representing the policy and the value function. While this approach could improve the performance of the CBBA algorithm, we suspect that by taking advantage of linear function approximation, the learning process can take less time as learned values are generalized among related states. While from the algorithmic side both TD and actor-critic methods were successfully extended to the linear approximation case, both approaches assume the existence of the "right" basis functions a priori. We are currently working on novel ways to expand the linear basis in an automated fashion so that it can capture interesting correlations among features. In the future, we are interested to integrate our *feature discovery* method with natural actor-critic in order to boost the learning speed and push the scalability of iCCA to even larger domains.

Acknowledgments

This research was generously supported by Boeing Research & Technology, Seattle, WA and by AFOSR grant FA9550-08-1-0086.

References

[1] Redding, J., Undurti, A., Choi, H., and How, J., "An Intelligent Cooperative Control Architecture," *American Control Conference*, to appear 2010.

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

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

[4] H. Brendan McMahan, Geoffrey J. Gordon, A. B., "Planning in the Presence of Cost Functions Controlled by an Adversary," *Proceedings of the Twentieth International Conference on Machine Learning*, 2003.

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

[6] Choi, H.-L., Brunet, L., and How, J. P., "Consensus-Based Decentralized Auctions for Robust Task Allocation," *IEEE Trans. on Robotics*, Vol. 25 (4), 2009, pp. 912 – 926.

[7] Howard, R. A., "Dynamic Programming and Markov Processes," 1960.

[8] Puterman, M. L., "Markov Decision Processes," 1994.

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

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

[11] Ponda, S., Redding, J., Choi, H.-L., Bethke, B., How, J. P., Vavrina, M., and Vian, J. L., "Decentralized Planning for Complex Missions with Dynamic Communication Constraints," *American Control Conference*, to appear 2010.

[12] Sutton, R. S., "Learning to Predict by the Methods of Temporal Differences," *Machine Learning*, Vol. 3, 1988, pp. 9–44.

[13] Bhatnagar, S., Sutton, R. S., Ghavamzadeh, M., and Lee, M., "Incremental Natural Actor-Critic Algorithms." *NIPS*, edited by J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, MIT Press, 2007, pp. 105–112.