A Human-Interactive Course of Action Planner for Aircraft Carrier Deck Operations

Bernard Michini* and Jonathan P. How†
Aerospace Controls Laboratory
Massachusetts Institute of Technology, Cambridge, MA 02139 USA

Aircraft carrier deck operations present a complex and uncertain environment in which time-critical scheduling and planning must be done, and to date all course of action planning is done solely by human operators who rely on experience and training to safely negotiate off-nominal situations. A computer decision support system could provide the operator with both a vital resource in emergency scenarios as well as suggestions to improve efficiency during normal operations. Such a decision support system would generate a schedule of coordinated deck operations for all active aircraft (taxi, refuel, take off, queue in Marshal stack, land, etc.) that is optimized for efficiency, amenable to the operator, and robust to the many types of uncertainty inherent in the aircraft carrier deck environment. This paper describes the design, implementation, and testing of a human-interactive aircraft carrier deck course of action planner. The planning problem is cast in the MDP framework such that a wide range of current literature can be used to find an optimal policy. It is designed such that human operators can specify priority aircraft and suggest scheduling orders. Inverse reinforcement learning techniques are applied that allow the planner to learn from recorded expert demonstrations. Results are presented that compare various types of human and learned policies, and show qualitative and quantitative matching between expert demonstrations and learned policies.

I. Introduction

Aircraft carrier deck operations present a complex and uncertain environment in which time-critical scheduling and planning must be done to fulfill mission requirements and ensure the safety of crew and equipment. The high-level goal is to ensure that the appropriate set of aircraft be in the air to accomplish a mission. This therefore requires the scheduled launching and landing of these aircraft within some specified timeframe. Launching and landing require a series of tasks such as refueling, re-arming, taxiing, parking, and repair. These tasks, in turn, require the allocation of a limited set of sometimes-conflicting resources such as catapults, the landing strip, deck elevators, ground vehicles and deck crews. All of this planning must be done in the presence of failures and uncertainties such as in-air emergencies (fuel or hydraulic leaks, battle damage, etc.), equipment failure (inoperative catapults, landing strip, etc.), and stochastic service times (refuel rate, re-arm time, etc.).

To date, all course of action planning in this environment is done by a hierarchy of human operators who rely on experience and training to safely negotiate off-nominal situations[9]. However, a computer decision support system could provide the operator with both a vital resource in emergency scenarios as well as suggestions to improve efficiency during normal operations. One component of such a support system is a course of action planner that generates a schedule of coordinated deck operations for all active aircraft (taxi, refuel, take off, queue in Marshal stack, land, etc.) that is optimized for efficiency, amenable to the operator, and robust to the many types of uncertainty inherent in the aircraft carrier deck environment. This paper describes the design, implementation, and testing of a human-interactive aircraft carrier deck

*Ph.D. Candidate, Aerospace Controls Laboratory, MIT, bmich@mit.edu
†Richard C. Maclaurin Professor of Aeronautics and Astronautics, Aerospace Controls Laboratory (Director), and Laboratory for Information and Decision Systems, MIT, Associate Fellow AIAA, jhow@mit.edu

course of action planner (DCAP). The planner is based on a Markov Decision Process (MDP) abstraction of the deck scenario, from which a policy is derived and used to generate operations schedules.

The planner's role as a decision support system requires that the human operator have a certain level of comfort and trust in the schedules produced by the automated system. Thus, an additional implicit requirement of the automated planner is that the plans generated be "similar to" those which the human operator would have generated, at least in nominal (non-emergency) deck operations. A natural way of achieving this similarity is to learn from demonstrations by the human expert operator and incorporate the learned behavior into the planner's policy. The process of learning via observed demonstrations is called apprenticeship learning. For problems cast in the MDP framework, a popular apprenticeship learning technique is *inverse reinforcement learning* (IRL), whereby the MDP state reward function is derived from observed expert demonstrations [6]. The reward function learned from the expert demonstrations is used to solve for the optimal policy of the MDP, which should then (presumably) be similar to the policy of the expert. A wide variety of IRL techniques in the literature have shown promising results in terms of learning and mimicking expert behavior [1,5–8,11]. A recently-published algorithm is implemented and applied to the carrier problem, and results are presented which show qualitative and quantitative matching between human operator demonstrations and the automated planner using the learned policy.

The paper is organized as follows. Section II describes the aircraft carrier deck environment in more detail, and provides an overview of the MDP abstraction and policy-based planning approach. Section III presents the challenges of specifying the MDP reward function, and the IRL techniques implemented to learn said reward function via human demonstrations. Section IV describes the exact and approximate reinforcement learning methods used to find a policy given the fully-specified MDP. Section VI presents results which compare policies derived from different solution methods (hand-picked policy, policy from hand-picked reward function, and policy from IRL-learned reward function). Finally, Section VI summarizes the findings and suggests areas of future work.

II. Carrier Environment and MDP Policy-Based Planning

This section describes the aircraft carrier deck environment, key uncertainties and planner requirements, and the MDP abstraction and policy-based planning approach.

A. Carrier Deck Environment

Figure 1 is a labeled diagram of the aircraft carrier deck layout. There are a total of four catapults. Catapults 1 and 2 are often used as deck parking areas and typically require relocation of several parked aircraft in order to be used. Catapults 3 and 4 are used more frequently, but overlap with the landing strip area. As a result, catapults 3 and 4 cannot be used while the landing strip is in active use and vice versa. The landing strip has three arresting gear wires, and landing aircraft must catch one of the three or else quickly apply power and takeoff (referred to as a "bolter"). There are four elevators that can transport aircraft to additional parking and repair areas below deck. The perimeter of the deck is lined with refueling stations.

In a typical scenario, a mission requires that a set of aircraft which start parked on deck be launched by pre-specified deadlines. A parked aircraft first taxis (towed by ground crew) to an available refueling station, where it is fueled. It then taxis to an operational catapult where it is attached and subsequently launched. After launch, each aircraft performs a mission for some amount of time, and returns to an area of airspace several miles from the carrier known as the Marshal stack. Aircraft are queued in the Marshal stack and separated by altitude. The aircraft in the lowest altitude level of the stack is cleared for landing, and starts on a approach path. If the aircraft successfully catches an arresting wire upon landing, it is disconnected from the wire and taxis to an open parking location. If the aircraft misses the wire, it takes off and is requeued in the Marshal stack for another landing attempt.

B. Key Uncertainties and Planner Requirements

There are several key uncertainties that must be accounted for by any course of action planner. First, a catapult can become inoperative due to a number of mechanical problems with the launch system. It then remains inoperative while it is being repaired, and the repair time is stochastic. Similarly, the landing strip can fail due to problems with the arresting gear (repair times are likewise stochastic) or can be obstructed by crew or taxiing vehicles causing any landing aircraft to return to the Marshal stack. Taxi and refuel times are

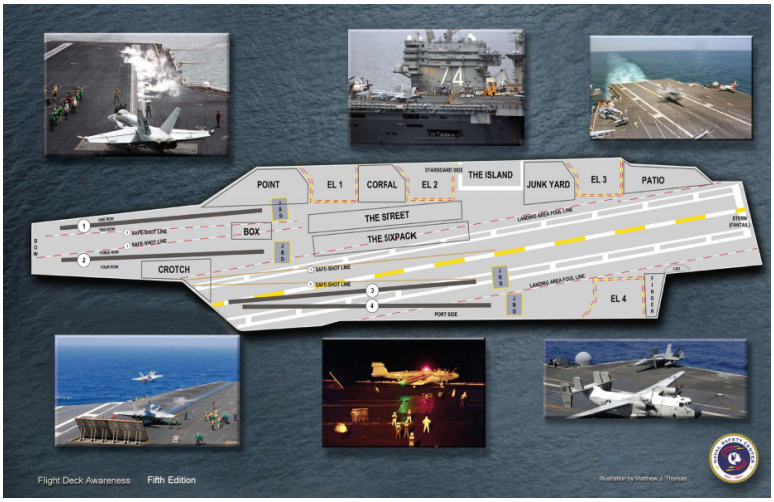

Figure 1. Diagram of aircraft carrier deck. Source: Flight Deck Awareness, Fifth Edition.

also stochastic, since aircraft must be towed and guided by ground crew and must be routed around traffic on deck. Also, aircraft may develop fuel leaks or incur damage while in the air and thus require expedited landing. Finally, landing aircraft may miss the arresting wires and need to return to the Marshal stack.

The deck course of action planner is responsible for building a feasible schedule of deck operations for each aircraft involved in the mission. This involves scheduling each aircraft's actions from the time it is parked before launch through the time it returns to parking after landing. The schedules for each aircraft must be coordinated such that aircraft are launched by the appropriate mission deadline and deck resources do not conflict (for example, Catapults 3 and 4 cannot be used while an aircraft is landing on the landing strip). The schedules must also be robust to the uncertainties listed above in terms of safety and operational efficiency (for example, aircraft should be scheduled for landing with a sufficient margin of fuel in case of a bolter).

The planner's role as a decision support system imposes additional requirements related to the human operator who will be using the information it provides. The schedules produced by the planner must be amenable to the operator, i.e. in nominal scenarios the planner's schedules should be close to what the operator would have done. This is vital for the operator to trust the automated planner and use it's suggested schedule in a potential emergency. Also, the operator should be able to provide feedback to the planner. Specifically, the operator can denote "priority aircraft" which the planner should prioritize over others (land sooner, for instance) and also suggest an ordering for certain tasks (land aircraft A before aircraft B). This feedback must be incorporated by the planner and reflected in the schedules that it generates.

Finally, the planner must generate schedules online in real-time. Continuous replanning is vital as the state of the real world changes, and schedules must be generated at the request of the human operator.

C. MDP Abstraction and Policy-Based Planning

The first step in developing a planner is finding a suitable model which captures the relevant system dynamics, actions, constraints, and uncertainties. With this in mind, the carrier deck scenario is modeled as a finite-state Markov decision process (MDP). An MDP is a tuple (S,A,T, γ,D,R), where S denotes a set of states, A is a set of actions, $T = P_{sa}$ is a set of state transition probabilities (P_{sa} is the state transition distribution upon taking action a in state s), $\gamma \in [0, 1)$ is a discount factor, D is the initial-state distribution from which the start state s_0 is drawn, and $R : S \mapsto \mathbb{R}$ is the reward function.[2] Formulation of the problem as an MDP is advantageous for several reasons:

1. Many uncertainties can be naturally encoded into MDP state transitions.

2. Once in MDP form, a host of dynamic programming and reinforcement learning literature exists for finding an optimal policy.

3. Inverse reinforcement learning (IRL) techniques can be used to learn a reward function from operator demonstrations which results in a policy that is similar to that of the human operator.

In the MDP model there are a set of aircraft, each with discrete location, fuel and priority states. The location states are Parked/Refuel, Catapult 1, Catapult 2, Catapult 3, Catapult 4, On Mission, Marshal Stack, On Approach, and Landing Strip. Each aircraft also has a boolean priority state. The catapults and the landing strip each have a boolean state indicating whether they are currently operative. Available action sets for each aircraft encode the problem constraints (e.g. it is not possible to transition to an occupied catapult, or land if the landing strip is inoperative) as well as the key uncertainties (an aircraft tries to transition from approach to landing strip, but misses the wire and thus transitions to the Marshal stack instead). Stochastic taxi times are encoded using transition probabilities between on-deck states. Transition probabilities also dictate when each catapult/landing strips becomes operative and inoperative. The fuel level of each aircraft is discretized into 6 levels. The size of the state space is:

$$(2 \text{ op/inop})^5 \times (9 \text{ locations} \cdot 6 \text{ fuel} \cdot 2 \text{ priority})^n$$

where n is the number of aircraft. The reward function R must be chosen such that reward is given for on-time aircraft launch and successful recovery. The selection of R directly shapes the optimal policy of the MDP and thus determines the behavior of the planner. Section III describes the methods by which R is chosen and/or learned from operator demonstrations. Once R is chosen, the MDP is fully specified and a host of reinforcement learning techniques can be used to find an optimal policy. Section IV describes the methods by which the policy is found given that the state space size for this MDP can be extremely large.

Solving for the policy is computationally expensive but can be done offline. Once a policy is found, a schedule is generated as follows:

1. The current real-world state of the carrier deck is converted into the abstracted MDP state. This involves setting the location, discretized fuel level, and priority status of each active aircraft as well as the operational status of the catapults and landing strip.

2. Using this as the initial state s_0, the pre-computed policy is executed to simulate the MDP forward to the point at which all aircraft have been launched, performed the mission, recovered, and parked. Ordering suggestions from the human operator are taken into account during the simulation process so long as the suggestions do not result in infeasibility.

3. The state sequence which results from the simulation is then used as the schedule of actions for each aircraft.

Forward simulation of the MDP is computationally inexpensive, and thus the above process can be done online in real-time (typically in less than 5 seconds for problems with ten aircraft or less).

III. Reward Function Representation and Inverse Reinforcement Learning

This section describes the feature-based representation of the reward function and the process of learning the reward function from expert demonstrations.

A. Reward Function Representation

Since the state space size of the MDP model can grow very large (for 5 vehicles it is already 5×10^{11}), a state-by-state (tabular) representation of the reward function is not realistic. A common solution is to instead represent the reward function as a linear combination of features:

$$R(s) = w^T \phi(s) \tag{1}$$

where w is a vector of weights and $\phi(s)$ is a vector of state features. For the experimental results presented herein, a total of 24 features were used to represent the reward function. They are summarized as follows:

- One for each fuel level that represents how many aircraft have that fuel level at any location
- One for each fuel level that represents how many aircraft have that fuel level while airborne

- The number of vehicles with no fuel in an airborne state (crashed)
- The number of vehicles with no fuel in a deck state (stalled on deck)
- The number of priority vehicles in an airborne state
- One for each location that represents how many aircraft are at that location

Features were selected that attempt to capture most of the relevant information that a human or automated planner might use in generating an intelligent policy. It should be noted, however, that manually selecting features is not a very systematic process. There is no way of knowing when enough features have been selected, or which types of features will increase policy performance other than trial and error methods. This remains an open problem for many inverse reinforcement learning techniques.

The vector of feature weights w can be selected by hand using a priori knowledge of the relative importance of each of the features. A more systematic way of finding the weights is to learn them from expert demonstrations, as described below.

B. Inverse Reinforcement Learning

An inverse reinforcement learning algorithm is used to learn the reward function from expert demonstrations. There are many inverse reinforcement learning methods in the literature [1,5–8,11], however the one chosen here is from Abbeel (2008) [1]. This was chosen due to the wide array of examples presented in the work that the algorithm has been successfully applied to (driving, quadruped motion, helicopter aerobatics). There are two main assumptions made. The first is that the expert has some internal reward function that he/she is optimizing in applying a policy. The second is that the reward function can be represented by a linear combination of features as described above.

The following is a brief overview of the algorithm, but the reader is referred to [1] for more details. Expert demonstrations are defined as a set of m state trajectories $\{s_0^{(i)}, s_1^{(i)}, \ldots\}_{i=1}^{m}$. From the expert's *feature expectations* $\hat{\mu}_E$ are calculated as:

$$\hat{\mu}_E = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{T} \gamma^t \phi(s_t^{(i)}) \tag{2}$$

This is similar to the expected reward from the expert, except that the expression is missing the feature weights w. The feature expectations $\hat{\mu}$ for an arbitrary policy can be found by simulating the policy m times and using the same expression.

An iterative algorithm is then used which, intuitively, finds the feature weights by attempting to match the feature expectations of the resulting reward function with the feature expectations from the expert demonstrations. Upon termination, a policy is found that is guaranteed to perform as well as the expert minus an ϵ. It should be noted that the true value function is not necessarily recovered, but the resulting policy produces similar feature counts and thus similar overall reward/performance. It follows that the behavior of the resulting learned policy should also be qualitatively similar to that of the expert.

C. Recording Expert Demonstrations

In order to record demonstrations from an "expert" operator, a computer game representing the MDP planning problem was implemented. Figure 2 shows the screenshot of the graphical interface. The user selects a vehicle by clicking on it. Once selected, the game highlights all the possible actions that the selected vehicle can take (e.g. if the vehicle selected is at Park/Refuel, each of the operative catapults will be highlighted). The user then selects an action by clicking on the highlighted location. All of the appropriate constraints and transition dynamics are reflected in the game (in fact, the underlying MDP is being used to control the game's state). This includes periodic failures of the catapults and landing strip. The fuel level of each vehicle is displayed as a progress bar.

The game runs at 10Hz, and the full state trajectory is recorded and logged to a file. State logs are then used by the inverse reinforcement learning algorithm described above to attempt learn the reward function of the operator and generate a similar policy.

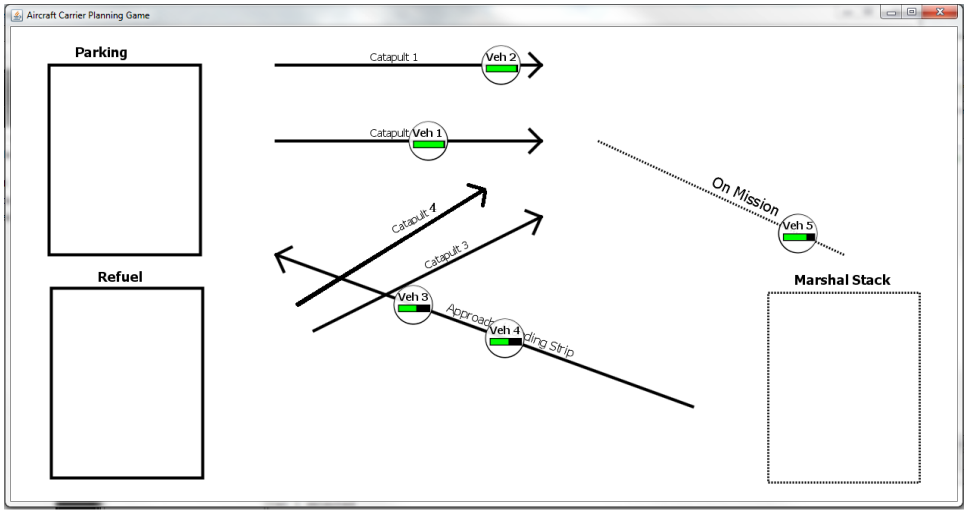

Figure 2. User interface screenshot of game for recording expert demonstrations.

IV. Finding a Policy

Once the reward function is set (either using hand-selected weights or weights learned from IRL), a policy must still be found from the now fully-specified MDP. This can be done using any one of a number of reinforcement learning algorithms in the literature. The following is a description of the methods used for the experimental results that are presented.

For small state spaces (less than five aircraft), conventional methods which use a full tabular value function representation (such as value iteration, for instance) can be used to find a policy. For larger state spaces (more than five aircraft), however, algorithms which rely on value function approximation must be used due to limits on computational power and memory. Many such approximate algorithms exist [2,4], and of those a common assumption is that the value function can be adequately approximated by a linear combination of features:

$$V(s) = v^T \phi(s) \tag{3}$$

where v is a vector of weights and $\phi(s)$ is a vector of state features, exactly as in the reward function representation. Many algorithms (such as fitted value iteration) find the weights v using Bellman backups at a set of sample states. However, the noisy cross-entropy method [3] is used herein for several reasons. Foremost, promising results presented in [12] show that the noisy CE method outperforms most other reinforcement learning algorithms when applied to the classic Tetris problem from [2]. Also, the algorithm is simple to implement, easily parallelized, and is simulation-based. A brief description of the algorithm is given below:

1. Choose an arbitrary distribution for the weight vectors v, $f_0 \sim \mathcal{N}(\mu_0, \sigma_0^2)$. Set $i = 0$.
2. Sample f_i n times to get a set of n weight vectors. For each sampled weight vector, simulate the MDP using that weight vector in the value function and calculate the standard discounted reward for each simulation.
3. Select the best $[\rho \cdot n]$ sampled weight vectors (those that produced the highest rewards).
4. Make μ_{i+1} the mean of the best weight vectors, and σ_{i+1}^2 the variance of the best weight vectors. Add noise Z_{i+1} to the variance.
5. Set $i = i + 1$, iterate from Step 2 until convergence.

The optimization parameters to be chosen are the number of simulations n per iteration, the fraction of samples ρ to keep, and the noise profile Z_i. Intuitively the algorithm is very similar to genetic-style stochastic optimization algorithms. The amount of noise prescribed by the noise profile decreases with more iterations and prevents early convergence to a suboptimal policy. The bulk of the computational work is in performing the simulations in Step 2, but this can be easily parallelized.

It should be noted that the IRL algorithm from Section III (and most other IRL algorithms) involves repeatedly using some reinforcement learning algorithm to solve for the policy. The assumption is made that the policy found is the *optimal* policy given some reward function, but with approximate methods such as the noisy cross-entropy method the policy found is not necessarily optimal. It follows that some of the IRL guarantees relating to the closeness of the resulting policy's feature counts to the feature counts of the expert demonstrations may not hold. This is not investigated in this paper and is left as an area of future work. The IRL algorithm is used *as if* the optimal policy is being found, when in fact it may be sub-optimal.

V. Experimental Results

Experimental results are presented for scenarios with four and eight aircraft. In the four-aircraft case, standard value iteration is used to find the optimal policy of the MDP. In the eight-aircraft case, noisy cross-entropy is used as described above. For the cross-entropy method, the number of simulations per iteration is set to $n = 1000$ and the fraction of samples to be kept is set to $\rho = 0.1$. The noise profile Z_i is taken from that prescribed in [12]. In all experiments a discount factor $\gamma = 0.98$ is used. In the IRL cases, 30 iterations were used and policies chosen as prescribed in [1].

For both the four- and eight-aircraft cases, two scenarios are tested. The first scenario considered uses the nominal failure model, in which at each step the catapults and landing strip fail with probability 0.005. The second scenario uses an increased failure model, in which at each step the landing strip fails with triply increased probability 0.015.

In each scenario tested, the initial MDP state is as follows. Half of the aircraft start in the mission state, and half of the aircraft start in the parked state. Every aircraft starts with half fuel. Aircraft that start in the mission state must be landed and parked. Aircraft that start in the parked state must be launched, perform the mission, land, and return to parking. The refueling of aircraft is optional (whether aircraft are refueled before takeoff is up to the policy being tested). If aircraft run out of fuel while in an airborne state they are said to have crashed. The scenario is complete when each aircraft is either parked (the initially-parked aircraft must have been launched and recovered first) or is out of fuel.

Five policies are compared in the results:

1. **Hand-Picked Reward:** Feature weights of the reward function are hand-picked based on intuition as to what features are "good" and "bad". The resulting reward function is used to find the optimal policy of the MDP.

2. **Safe Expert:** A human "expert" operator (the author) takes actions so as to weight safety over minimizing mission time. Practically, this means refueling aircraft fully before launch, and landing vehicles with lowest fuel first, considering the priority aircraft. The software from Section III is used to record the MDP state trajectory from 10 trials.

3. **Safe IRL:** The inverse reinforcement learning algorithm from Section III finds reward function weights using the Safe Expert demonstrations. The learned reward function is used to find the optimal policy of the MDP.

4. **Risky Expert:** A human "expert" operator (the author) takes actions so as to weight minimizing mission time over safety. Practically, this means launching aircraft before they are refueled and landing aircraft as soon as they arrive in the Marshal stack, paying little attention to priority aircraft. The software from Section III is used to record the MDP state trajectory from 10 trials.

5. **Risky IRL:** The inverse reinforcement learning algorithm from Section III finds reward function weights using the Risky Expert demonstrations. The learned reward function is used to find the optimal policy of the MDP.

The statistics presented below were collected by averaging the results from 100 trials for each of the hand-picked, Safe IRL, and Risky IRL policies. The average over the 10 recorded expert demonstrations were used for Safe Expert and Risky Expert. Data is presented with $2-\sigma$ error bars. Mission performance metrics compared are overall mission time (number of MDP states from start to finish as defined above, lower is better), summed fuel of all aircraft averaged over then entire mission duration (where 0 is empty and 1 is full), the number of aircraft that crashed during the mission, and the percentage of the time that the

priority aircraft was landed first when there was more than one aircraft in the Marshal stack. One vehicle was randomly chosen to be the priority aircraft for each test.

A. Mission Performance: four-aircraft Scenarios

Figure 3 shows experimental results for the four-aircraft scenarios tested. The left column corresponds to the nominal failure model and the right column corresponds to the increased landing strip failure rate.

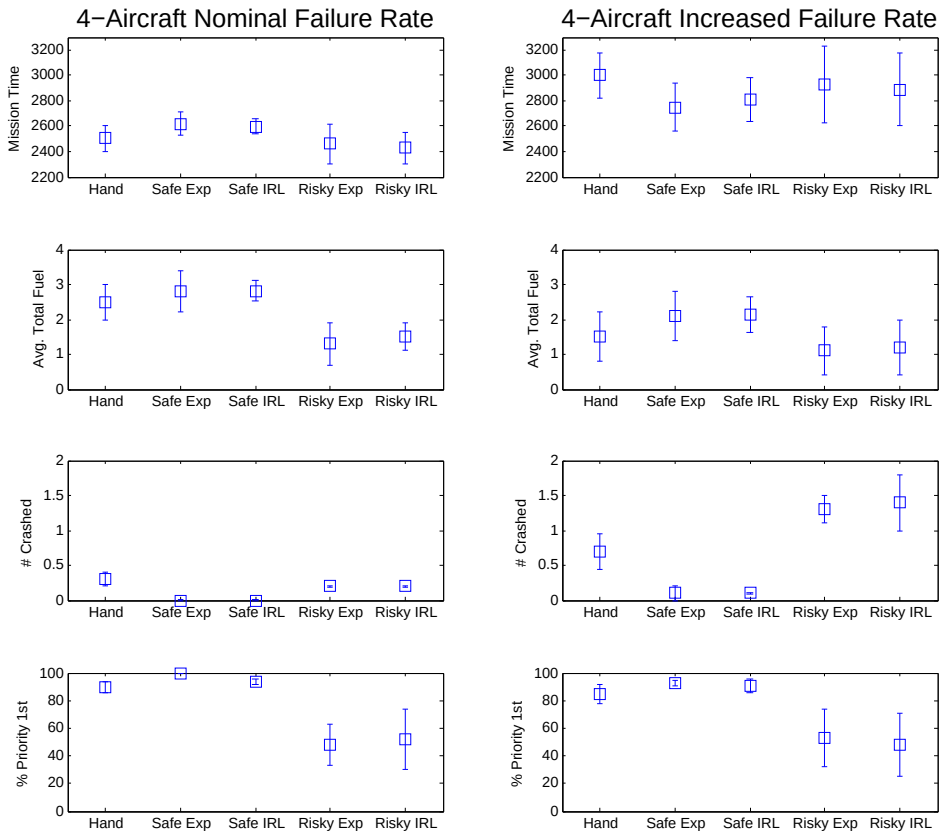

Figure 3. Results for four-vehicle scenario. Nominal failure rate in the left column and increased landing strip failure rate on the right. The x-axis corresponds to, from left to right, the hand-picked reward function, the safe expert demonstrations, the safe expert IRL-learned policy, the risky expert demonstrations, and the risky expert IRL-learned policy.

For the nominal case in the left column, the data for mission time is as expected in that the risky policies perform the mission slightly faster than the safe policies. In terms of total fuel level, the risky policies expectedly show lower fuel level since refueling was often skipped to reduce mission time. This is reflected in the slightly higher fraction of crashed aircraft for the risky policies, though all crash rates were relatively low. Finally, it is clear that the hand-picked and safe policies landed the priority aircraft first almost every time there was an option to do so, reflecting the fact that the safe expert was instructed to do so. The risky policies did not pay attention to the priority aircraft status as expected, and landed it first about half the time with high variance.

For the high failure rate case in the right column, mission times overall had a higher mean and wider variance due to the increased failure rate of the landing strip. As a result, fuel levels were expectedly lower, with the risky policy fuel levels still being the lowest for the same reasons as before. However, the big difference in the high failure case is that the risky policies have a much higher crash rate while the safe policies still remain fairly low. This is due to aircraft with low fuel levels being stranded airborne in the Marshal stack when the landing strip fails and is being repaired. This highlights the lack of robustness of the risky policy which tends to not fully refuel aircraft before launch. Similar results are seen in the priority

aircraft order, as the safe policies still landed aircraft first and the risky ones did not.

Another result seen in the data for both the nominal and high-failure cases is the qualitative matching between the expert human demonstrations and the learned policies. For all four performance metrics the safe expert data matches with the safe IRL-learned policy, and the same can be said for the risky policies. Intuitively this means that the IRL algorithm has succeeded in learning a reward function that results in a policy that is qualitatively similar to the expert demonstrations.

B. Mission Performance: eight-aircraft Scenarios

Figure 4 shows experimental results for the eight-aircraft scenarios tested. The left column corresponds to the nominal failure model and the right column corresponds to the increased landing strip failure rate.

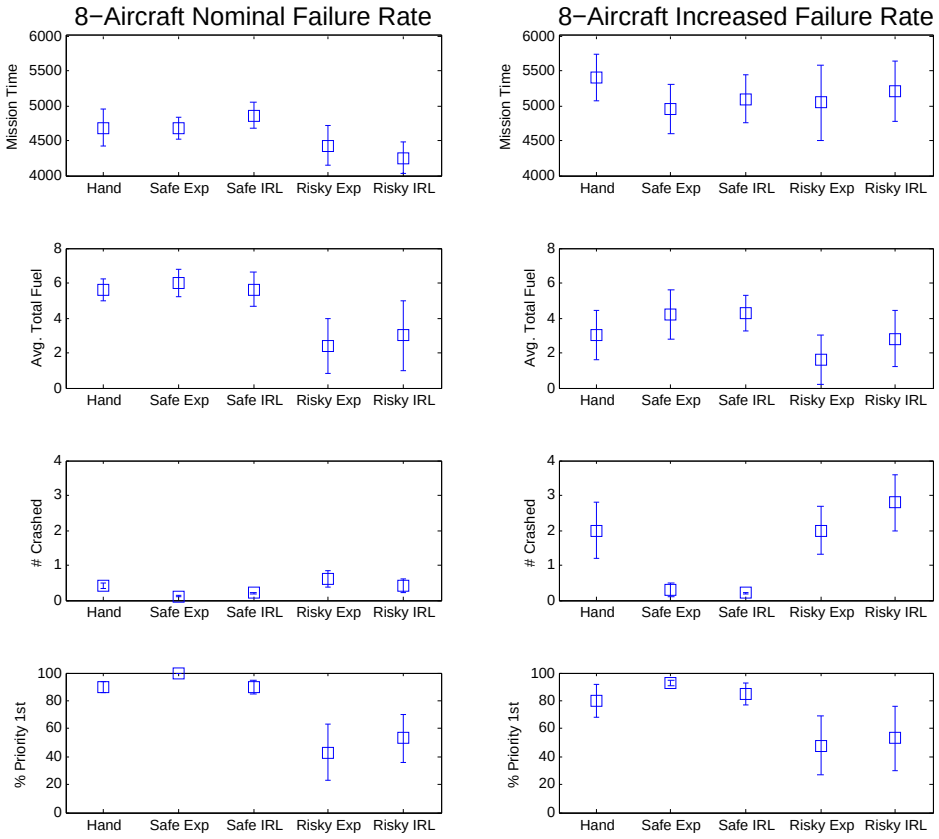

Figure 4. Results for eight-vehicle scenario. Nominal failure rate in the left column and increased landing strip failure rate on the right. The x-axis corresponds to, from left to right, the hand-picked reward function, the safe expert demonstrations, the safe expert IRL-learned policy, the risky expert demonstrations, and the risky expert IRL-learned policy.

In terms of performance trends, the eight-aircraft case is similar to the four-aircraft case. Mission times are lower in the nominal case for the risky policies, but even out in when the landing strip fails more frequently. Fuel levels are lower for the risky policies in both the nominal and failure cases, as expected. The safe expert had a slightly higher crash rate in the eight-aircraft case, presumably due to the increased load of managing four more vehicles. The risky policies again showed a much higher crash rate in the failure case due to not refueling before launch. The same trends are again observed for priority aircraft, as the safe policies land priority aircraft first and the risky policies do not half the time.

It is worth noting, however, that the qualitative matching between demonstrated and learned policies is not as strong in the eight-aircraft case. For both the mission time and fuel level metrics, the learned policy still follows the trend of the expert demonstrations but does not match as closely as in the four-aircraft case. As discussed earlier, this could be due to the use of an approximate reinforcement learning technique

(noisy cross-entropy) in the eight-aircraft case which does not necessarily find the optimal policy within the IRL algorithm. It is suspected that this has an impact on the guarantees of the IRL algorithm, but further investigation is left as future work.

C. Learning Comparison

To more clearly determine how well the IRL task of learning from expert demonstrations is done, the feature expectations of the demonstrated and learned policies is compared. Table 1 compares the feature expectations for the three main location features of the demonstrated policy and the learned policy. Data is presented only for the nominal failure rate case. The learned policy feature counts are estimated using Monte Carlo sampling as in [1].

4 Aircraft					8 Aircraft				
		Park	Refuel	MS			Park	Refuel	MS
Safe	$\mu_{\text{Exp.}}$	0.252	0.316	0.092	Safe	$\mu_{\text{Exp.}}$	0.285	0.330	0.129
	μ_{IRL}	0.265	0.301	0.136		μ_{IRL}	0.240	0.281	0.087
Risky	$\mu_{\text{Exp.}}$	0.243	0.192	0.127	Risky	$\mu_{\text{Exp.}}$	0.192	0.168	0.122
	μ_{IRL}	0.231	0.161	0.110		μ_{IRL}	0.215	0.133	0.084

Table 1. Feature expectation matching between the expert demonstrations and IRL-learned policies. Four-aircraft case on the left and eight-aircraft case on the right.

For the four-aircraft case, the feature expectations for both the safe and risky policies match closely for all three of the location features shown. This is generally the case for the other features not shown as well. For the eight-aircraft case, however, the feature expectations do not match as closely for the three features shown or the other features not shown. This agrees with the mission performance metric matching discussed above, and again it is suspected that the use of the approximate reinforcement learning algorithm in the eight-aircraft case led to the weaker matching between expert demonstrations and learned policy.

VI. Conclusions and Future Work

In this paper an MDP policy-based planning algorithm is presented for the problem of automated aircraft carrier deck course of action planning. The planner accepts priority aircraft and schedule suggestion feedback from the human operator. An inverse reinforcement algorithm is implemented so that demonstrations from an expert human operator could be used for policy learning. Results confirm that the learned policy is qualitatively and quantitatively similar to the expert demonstrations. Matching is closer in the four-vehicle case when conventional reinforcement learning methods are able to find the optimal policy for a given MDP as opposed to the eight-vehicle case when an approximate algorithm is used that generates potentially sub-optimal policies.

Ongoing work involves integrating and testing the planner using a detailed aircraft carrier deck simulator developed at the Humans and Automation Laboratory at MIT[10]. The simulator accounts for detailed crew movement, vehicle dynamics, deck traffic, and ground vehicles. It also has a specialized interface for accepting, displaying, and modifying schedules generated by the automated planner. The reader is referred to [10] for details.

The results presented highlight a potential problem with many IRL algorithms currently in the literature, and that is the problem of performing IRL in a large state space. Using state features make reward function representation tractable in large state spaces, but every IRL algorithm still relies on some reinforcement learning algorithm for finding an optimal policy. Large state spaces necessitate using an approximate algorithm that may find a sub-optimal policy, and the effect that this has on IRL algorithms should be investigated.

Finally, manual selection of state features is an open problem in IRL. Several recent algorithms such as [8] boost simple features into the representation as needed to address this. Another potential approach is to use non-degenerate kernel functions for reward function representation and an associated kernel-based method for learning the reward function from demonstrations.

Acknowledgments

The authors would like to recognize and thank the faculty and students at the Computer Science and Artificial Intelligence Lab (CSAIL), the Humans and Automations Lab (HAL), and the Laboratory for Information and Decision Systems (LIDS) at MIT who are also involved in the DCAP project. This work is funded under the Office of Naval Research Science of Autonomy program, Contract #N000140910625.

References

[1] Pieter Abbeel. *Apprenticeship learning and reinforcement learning with application to robotic control*. PhD thesis, Stanford, 2008.

[2] D P Bertsekas and J N Tsitsiklis. *Neuro-Dynamic Programming*, volume 5 of *the Optimization and Neural Computation Series*. Athena Scientific, 1996.

[3] Pieter-tjerk D E Boer. A Tutorial on the Cross-Entropy Method. *Annals of Operations Research*, pages 19–67, 2005.

[4] L Busoniu, R Babuska, B De Schutter, and D Ernst. *Reinforcement Learning and Dynamic Programming using Function Approximators*. CRC Press, 2010.

[5] Gergely Neu and Csaba Szepesvari. Apprenticeship learning using inverse reinforcement learning and gradient methods. In *Proc. UAI*, 2007.

[6] A Y Ng and S Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 663–670. Morgan Kaufmann Publishers Inc., 2000.

[7] Deepak Ramachandran and Eyal Amir. Bayesian inverse reinforcement learning. *IJCAI*, pages 2586–2591, 2007.

[8] Nathan Ratliff, David Bradley, and JA Bagnell. Boosting structured prediction for imitation learning. In *Robotics Institute*, 2007.

[9] Gene I Rochlin, Todd R La Porte, and Karlene H Roberts. The Self-Designing High-Reliability Organization : Aircraft Carrier Flight Operations at Sea. *Naval War College Review*, 40(4, Autumn):76–90, 1987.

[10] Jason Ryan, M.L. Cummings, and Axel Schulte. Designing an Interactive Local and Global Decision Support System for Aircraft Carrier Deck Scheduling. In *Proceedings AIAA Infotech*, 2011.

[11] Umar Syed and R E Schapire. A Game-Theoretic Approach to Apprenticeship Learning. *Advances in Neural Information Processing Systems 20*, 20:1–8, 2008.

[12] István Szita and András Lörincz. Learning Tetris using the noisy cross-entropy method. *Neural computation*, 18(12):2936–41, December 2006.