Risk-Based Sensing in Support of Adjustable Autonomy

Lawrence A. M. Bush
Massachusetts Institute of Technology
Department of Aeronautics and Astronautics
77 Massachusetts Avenue
Cambridge MA 02139-4307
339-368-1078
bushL2@csail.mit.edu

Andrew J. Wang
Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
77 Massachusetts Avenue
Cambridge MA 02139-4307
617-452-4012
wangaj@mit.edu

Brian C. Williams
Massachusetts Institute of Technology
Department of Aeronautics and Astronautics
77 Massachusetts Avenue
Cambridge MA 02139-4307
617-253-1678
williams@csail.mit.edu

Abstract—**Current unmanned systems are typically tele-operated and manpower intensive relying on human operators and their decision-making capabilities to perform platform and mission tasks. We envision a future where unmanned platforms have greater decision making abilities and autonomous behaviors are the norm. For example, unmanned autonomous systems will be deployed as teams along with their human supervisor. The UAS will interact with the supervisor at a high level, then take on an expanded role in mission planning, resource allocation and route planning.**

Toward this end, we have developed a risk-based adjustable autonomy system with a task directed adaptive sensing technology concept to allow system autonomy operation at a level in which an operator has confidence of success. We test our idea in simulation for a natural disaster recovery task. We present experimental results verifying the utility of our technology.

TABLE OF CONTENTS

1. INTRODUCTION

While sensing and avionics technology has made great strides in the past decade, the autonomous capabilities of UAVs lag behind. Autonomous systems lack humans' creative reasoning abilities to maintain safe and robust behavior in uncertain environments. Thus, most unmanned systems still rely on human tele-operation, which is manpower intensive and requires reliable communication with a ground base. In this work, we propose a new approach to adjustable autonomy for reducing operator workload while retaining trust that the mission will proceed safely. Previous work in adjustable autonomy usually involves humans manually assessing the utility and setting the level of autonomy. This approach requires them to maintain complete situational awareness throughout the mission. However, applications that demand large-scale distributed efforts, such as disaster relief, environmental surveys, and convoy protection render this approach impractical. Our goal is to design a system that assesses its own abilities and requests human assistance in the amount needed. The challenge is to build the trust of the human operator that this system truly knows what assistance it needs and can operate safely otherwise.

Our approach to adjustable autonomy gains operator trust by making and explaining decisions based on risk. Our work is developed in the context of a disaster relief scenario, where a rescue vehicle must traverse an uncertain landscape to administer aid. The mission planner must tradeoff between risk reduction and decision deadlines. We consider two aspects to risk: the probability of mission failure and decision uncertainty in the mission planning process. Due to the inherent uncertainty in our belief, the probability of mission failure is not known exactly, so the mission planner evaluates risk as a distribution over this probability. In particular, the metric used to evaluate risk is the confidence that the probability lies above a threshold. The metric is calculated over the entire rescue vehicle mission path and it is calculated through a combination of Dijkstra's algorithm and value iteration. These risk distributions give rise to uncertainty when choosing between mission plans for lowest probability of failure.

To mitigate this, aerial scouts are sent ahead of the rescue vehicle to reduce belief uncertainty where it is needed most. The scouts are tasked with surveying areas that yield the highest expected uncertainty reduction within the time before the mission vehicle makes its next decision. Path planning for the scouts is achieved via approximate dynamic programming using a stochastic value function represented by a Gaussian Process. For each scout, we use a unique representation of the state-space which is centered and rotated around the vehicle. Based on the updated belief provided by the scouts and the timing constraints of the scenario, the mission planner determines whether mission plans exist that have low enough risk associated with them.

978-1-4577-0557-1/12/$26.00 ©2012 IEEE

Figure 1 - Motivating Scenario: Natural Disaster Relief Scenario

Based on a series of risk thresholds, different amounts of human assistance are requested. Thus, we can engage the operator specifically when needed and at the proper amount, as well as provide reasoning and context for why and where the operator's help is needed. Simulation experimental results are presented which verify that the scout planner effectively reduces belief and decision uncertainty given any area to survey. The results demonstrate the potential of this new approach to achieve consistent safety and robustness throughout a mission by requesting operator assistance at the appropriate times.

2. INNOVATIONS

Adjustable autonomy has long been appealing for its vision to utilize the best abilities of humans and autonomous agents. However, the difficulties in creating a truly synergistic dynamical relationship between humans and robots make adjustable autonomy a loosely defined concept. In this section, we introduce the risk-based innovations of our approach to adjustable autonomy, describe the system architecture, and explain the major algorithms driving our framework.

An adjustable autonomy system makes two types of decisions: The first type determines what vehicle actions to take in the future and the second governs when and how to engage the human operator. *Our first innovation is that we ground both these decision types on the risk to the mission goals.* Given a logistical plan for the mission, we probabilistically quantify the risk to each component of the plan. The component risks imply a risk configuration over each mission goal, which informs how we engage the human. Furthermore, the notion of risk is built into the planning process. Our adjustable autonomy architecture takes advantage of this to provide situational awareness, keeping the human involved at the appropriate level of detail for each mission component.

Our second innovation is how we improve our knowledge of the risks to the mission through the use of scout aerial vehicles. Using only the initial low-resolution knowledge of map risk severely limits the quality of decisions we can make, so we dynamically deploy scouts to collect more detailed information. Our algorithms lead the scouts toward the most valuable data that will help us identify the least risky paths for future components of the mission. The scouts are tasked to survey areas least well known about relative to the logistics plan. Flying over an area that our logistics vehicles will never cross on ground is useless unless it lies on the quickest path from one area of interest to the next. In summary, our contribution to adjustable autonomy is to encode risk at each decision-making process.

3. ARCHITECTURE

The algorithmic modules within our artificial intelligence architecture enable the incorporation of risk information and involvement of the human operator. The modules include the logistics executive, the scout executive, and the adjustable autonomy module. These components interact with the logistics vehicle, the scout vehicle, and the human operator, respectively, depicted in Figure 2. The logistics executive contains several sub-modules. Two of them are the "high-level" logistics planner and the "low-level" roadmap planner, each containing a risk assessment functionality that operates on the risk belief map. Together, these items determine the course of action for the

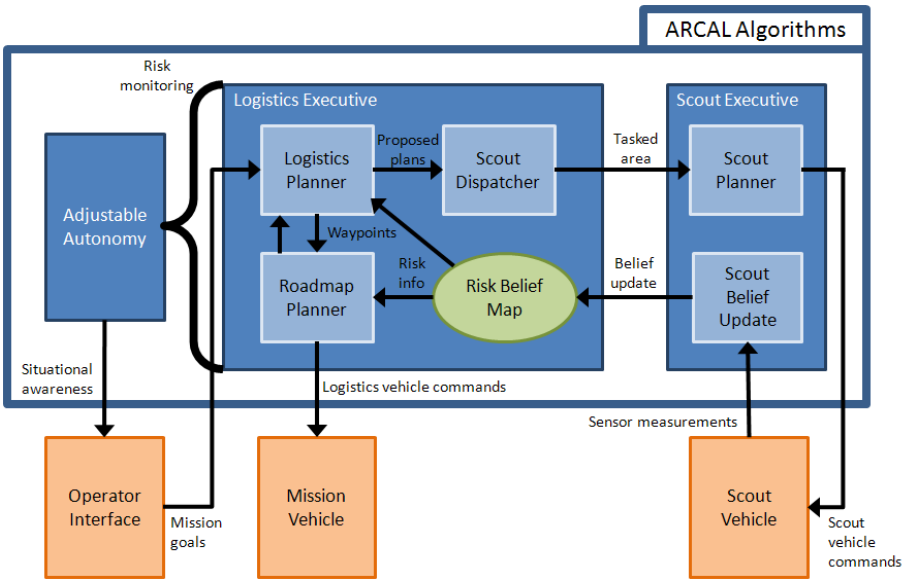


Figure 2 - System Architecture

logistics vehicle. The logistics planner accepts mission goals from the operator and generates sequences of waypoints, producing a "high-level roadmap" which will achieve the mission goals. Then, finding the actual path taken between waypoints is performed by the roadmap planner.

It is the job of the logistics planner to choose the actual sequence of waypoints in such a way that it balances and reduces the risk among each component of the mission. However, to make well-informed decisions, it will need the scouts to gather additional data in areas the logistics vehicle may cross in the future. The scout dispatcher determines where to send the scouts, given the plans currently considered by the logistics planner. Each plan has some uncertainty in how truly risky it is to execute. This plan risk uncertainty is mapped into map uncertainty, or, in other words, the scout dispatcher determines the map locations that are responsible for most of the plan uncertainty. It tasks the scouts to survey these areas, for the information they collect will help disambiguate candidate plans. Each scout's executive runs a scout planner that accepts these areas as input as well as a time limit for reporting results on each area. The executive must also receive the current risk belief for the relevant area. The planner runs an adaptive sampling algorithm that is trained to fly the path that achieves the highest expected information gain within the time allotted. As sensor measurements arrive, the belief update module incorporates them into the risk belief, and at the end of a sensing task, the scout reports the updated risk belief back to the logistics executive.

In a non-adjustable autonomy architecture, the human operator would directly interface with the logistics executive, but here, the adjustable autonomy module mediates their interaction. This module continuously monitors the risk associated with each mission component according to the entire state of the logistics executive. It tracks the possibility that each component's risk might exceed user-specified thresholds. As these risks evolve due to additional planning and updated risk beliefs, adjustable autonomy gradually requests human attention or intervention for certain mission components. Thus, while the operator still specifies mission goals to the logistics planner, she now has an interface to override the different components of the logistics executive at varying levels of control. Together, all these modules provide a rational, risk-based framework to help direct the operator's attention to the most pressing issues.

4. ALGORITHMS

Below high-level descriptions of the major algorithmic components corresponding to our two innovations are presented. Respectively, the focus is on the risk assessment calculation and the scout planner.

4.1 Risk Assessment

A key capability of our system is to assess the risk to the mission goals. Risk is defined as the likelihood that the logistical plans we produce will achieve each goal. In other words, if a plan is to succeed, then every part of the plan must succeed. The risk assessment problem is then stated as follows: *Given a path plan that nominally achieves the mission goals and a belief map of the environment, we wish to compute a distribution over the path success probability.* While we would like to know the true path success

probability, this is impossible since we do not have the true map of the environment. Rather, we possess a belief map which models not just where we think certain features and obstacles are, but also how well we know them. This reflects the intuition that although we may know a certain type of obstacle exists at a general vicinity, without extremely fine sensing, we have only a general idea of its precise location and threat level. Thus, we must compute and our algorithms must operate on a probability distribution over the success probability, i.e. a *risk distribution*.

Given this definition of risk, we describe how we represent risk in a belief map. Then, we can build paths over this map and operate on the risks this path encounters to devise a risk distribution for the path. Finally, we will explain how the scout measures and updates the risk belief map.

Our belief map is represented by a grid of square cells. Each cell contains a distribution over the probability of success if we traverse that cell in any direction, independently of all other cells. We use this interpretation because it allows us to use the Markov assumption later when composing cells into paths. The distribution in each cell takes the form of a Beta distribution, which represents a distribution over the probability parameter p of a binomial distribution. It is parameterized by (α, β), where α and β are the effective number of observed successes and failures, respectively. The mean and variance of a Beta distribution are given by

$$\begin{aligned} \mu &= \frac{\alpha}{\alpha+\beta} \\ \sigma^2 &= \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}. \end{aligned}$$

We may compute the inverse relationships as well:

$$\begin{aligned} \alpha &= \frac{\mu^2(1-\mu)}{\sigma^2} - \mu \\ \beta &= \frac{\mu(1-\mu)^2}{\sigma^2} - \mu + 1. \end{aligned}$$

In our belief map, we parameterize each cell with a mean and variance to represent a Beta distribution. Not only does the Beta admit an intuitive interpretation, its parameterization is also appealing for real-time calculation.

The form of our belief map makes it convenient to compose cells into paths, although we will need to approximate the distribution for the resulting path. We make the Markov assumption that the probability of successfully traversing a certain cell is independent of the probabilities for other cells. Then, given a path of length n cells with random variable probabilities $p_1, \cdots, p_n$ of successfully traversing each one, the success probability for the entire path is

$$p = \prod_{i=1}^{n} p_i. \tag{4.1}$$

Unfortunately, the true distribution for the entire path is not a Beta distribution and is hard to compute, so we approximate it as a Gaussian. Gaussian distributions are also parameterized by a mean and variance. Thus, by applying the independence property, we obtain the following expressions for mean and variance of the path's risk distribution.

$$\begin{aligned} \mu &= E[p] = E\left[\prod_{i=1}^{n} p_i\right] \\ &= \prod_{i=1}^{n} E[p_i] = \prod_{i=1}^{n} \mu_i \\ \sigma^2 &= E[p^2] - E[p]^2 \\ &= E\left[\prod_{i=1}^{n} p_i^2\right] - \mu^2 \\ &= \prod_{i=1}^{n} E[p_i^2] - \mu^2 \qquad (4.2) \\ &= \prod_{i=1}^{n} (\sigma_i^2 + \mu_i^2) - \mu^2. \qquad (4.3) \end{aligned}$$

Equation 4.2 is straightforwardly interpreted as the estimated risk. Equation 4.3 specifies that due to the multiplicative nature of Equation 4.1, a cell will amplify the variance effects of other cells if both its mean probability of success and its variance are large, and vice versa.

The Gaussian introduces the issue that it extends to $\pm\infty$. Thus, we introduce another approximation by truncating the distribution at 0 and 1 and scaling the resulting curve so that the area beneath it integrates to 1.

There still remains the question of how we measure and model obstacles from the environment into our belief map. We assume our sensor has algorithms for detecting and characterizing features of the environment. For example, suppose the scout's camera detects a pothole and computes a "measurement" of p for the success probability. If the camera's resolution is characterized by a variance σ_z^2, then the pothole's risk distribution is characterized by $\mu = p$ and $\sigma^2 = \sigma_z^2$. Now, we must encode this information into the grid cells c_i that the pothole occupies. Assuming each grid cell has the same distribution, we take the characteristic length d of the pothole (such as diameter or side length), and invert the relationships in Equations 4.2 and 4.3 to get

$$\begin{aligned} \mu_i &= \mu^{(1/d)} \\ \sigma_i^2 &= (\sigma^2 + \mu^2)^{(1/d)} - \mu_i^2. \end{aligned}$$

Thus, we are effectively spreading the obstacle's risk distribution over the set of cells it occupies.

However, the Beta distribution parameterized by $(\mu_i,\ \sigma_i^2)$ is a measurement and not what we insert into the belief map. At each time step t, the belief map will already have a prior distribution $(\mu_t,\ \sigma_t^2)$. We use a Kalman filter to integrate the measurement into the belief, which reduces to the Equations 4.4 and 4.5.

$$\mu_{t+1} = \frac{\sigma_t^2\mu_i + \sigma_i^2\mu_t}{\sigma_t^2 + \sigma_i^2} \quad (4.4)$$

$$\sigma_{t+1}^2 = \frac{\sigma_t\sigma_i}{\sigma_t + \sigma_i}. \quad (4.5)$$

To summarize, we formulate risk assessment in terms of finding the risk distribution over a path, given a risk belief map. The map is gridded into cells, each of which contains a Beta distribution. Paths are sequences of adjacent cells, and we represent their risk distributions as truncated and scaled Gaussians. To update the belief map with scout measurements of an obstacle, we spread the obstacle's distribution over the grid cells it encompasses, and we apply the Kalman filter to update our belief.

4.2 Scout Planner

This section describes the algorithm for the Scout Planner component of the ARCAL architecture (Autonomous Robot Control via Autonomy Levels). As described in Section 1 and Section 2, the scout's purpose is to obtain more detailed scans of certain areas that could yield safe routes for the logistics vehicle. The scenario is depicted in Figure 1. While the logistics executive tasks the scout to examine certain areas, it would be inefficient for the scout to traverse every square mile of its assigned area. For example, a human operator would pilot the scout immediately to the areas that we are most uncertain about and thus stand to gain the most from detailed surveillance. Furthermore, the scout only has limited time to complete its scans and report back to the logistics executive. Our scout planner algorithm addresses these observations and directs the scouts to collect data that optimally reduces risk uncertainty for the logistics vehicle.

Figure 3 zooms into the scout planner within the ARCAL architecture and illustrates various components of the scout planner algorithm. The following subsections motivate and walk through these components. Section 4.2.1 begins by defining the scout planning problem in the framework of the Markov Decision Process (MDP). Using this formalism, we can use well-developed value iteration techniques described in Section 4.2.2 to solve for the optimal policy that dictates what path the scout should take. The policy is typically encoded as a value function. However, solving this MDP exactly for a typical scout scenario would require intractable computation, and the value function could only be represented in unreasonable amounts of storage space. Thus, Section 4.2.2 augments the value iteration process with approximations to yield non-optimal but reasonable solutions. These calculations are performed offline, and the approximate solution is stored in an approximate value

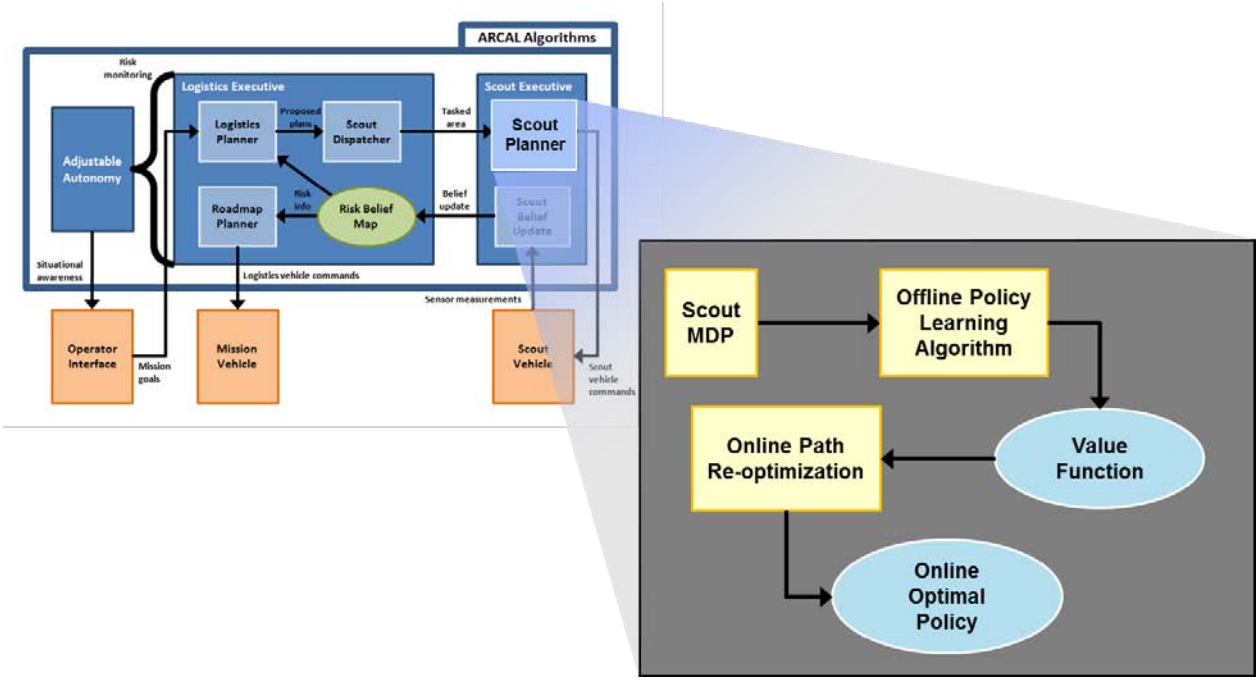


Figure 3 - Scout Planner Architecture

function. When the time comes for the scout to execute actions online, it further re-optimizes the value function to its particular situation within the available computation time. This online process is described in Section 4.2.4.

4.2.1 Scout Planning Problem as a Markov Decision Process—The scout planning problem in the context of ARCAL is formulated as follows. The scout dispatcher tells the scout which subset of the full map needs to be examined to have the uncertainty in the risk belief reduced. This subset is represented as a set of grid cells. Each grid cell has a prior risk distribution associated with it. The scout's goal is to fly a path over the area in the allotted time exactly such that it maximizes the total reduction in variance over these grid cells. (The total variance reduction is the sum of all variance reductions in each grid cell.) To solve this problem, we cast it in the general framework of the MDP, which is as follows.

An MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where:

- S is the set of all possible states.
- A is the set of all possible actions in each state.
- P is the set of state transition probabilities.
- R is the reward function, with γ as a discount factor for future rewards.

MDPs operate on discrete time steps. When executing action a, in state s, the probability of transitioning to state s' in the next time step is denoted as $P^a_{ss'}$ and the expected reward associated with that transition is denoted $R^a_{ss'}$. The state transition probabilities enable MDPs to be used in stochastic environments. In a deterministic setting, $P^a_{ss'}$ is 1 if and only if taking action a in state s takes us to state s'; otherwise it is 0. The reward structure is set up so that over sequences of actions, the rewards accumulate but with a discount factor γ so that future payoffs are less valuable than more immediate payoffs.

A solution to an MDP is a *policy*, which assigns an action to each state of the MDP. The value of a state under a policy, $V_\pi(s)$, is the expected sum of discounted rewards obtained when policy π is followed, starting in state s. The objective is to find an optimal policy π^*, which maximizes the value of every state ($V_{\pi^*}(s) = V^*(s) \forall s \in S$). Note that for a policy to be optimal, it must choose action a in state s such that the expected value of the subsequent state s' is maximized. In mathematical terms, $\forall s \in S$, the optimal policy and value function are related as follows.

$$\pi^*(s) = \arg\max_{a \in A} \sum_{s' \in S} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^*(s')] \tag{4.6}$$

$$V^*(s) = \max_{a \in A} \sum_{s' \in S} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^*(s')] \tag{4.7}$$

Equations 4.6 and 4.7 codify *Bellman's Principle of Optimality*. In other words, the optimal plan starting from state s is to choose the action that lands us in the next best state, and then continue with the optimal plan starting from s'. Thus, to find the optimal policy, it suffices to solve for the optimal value function, and then read-off the policy from it.

For our scout planning problem, we define the following components of an MDP.

- The state S includes the vehicle location and pose as well as the belief map (i.e. the risk distributions over the relevant grid cells).
- The action set A defines how the scout vehicle can move. In our problem, the available actions are *left*, *right*, and *straight* at any grid cell.
- Our problem is deterministic. Thus, the transition probabilities $\mathcal{P}^a_{ss'}$ are 1 if and only if the new location, pose, and belief in state s' match those according to the dynamics and Kalman filtering resulting from taking action a in state s. In effect, P is the specification of the problem dynamics.
- The reward $\mathcal{R}^a_{ss'}$ is defined to be the total reduction in uncertainty in the relevant grid cells going from state s to s'. It is calculated by taking the sum over all the reductions in variance resulting from the Kalman filter updating the state from the observations.

It is interesting to note that the state space includes the belief map in addition to the location and pose. This information is a necessary part of the state because the reward in going between states is solely defined by the reduction of variance. It is more valuable to move between cells and see a great decrease in variance because of high initial uncertainty in the area swept over by the sensor than to move between the same cells and see a small decrease because the uncertainty was already low to begin with. Also, including the belief map makes our state space continuous. This will prompt our approximation architecture described in Section 4.2.3.

4.2.2 Value Iteration for Exact MDP Solutions—Given our MDP model $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, the classical method to determine the optimal value function is to use value iteration. Starting with an initial V_0 that is zero for all states, we iterate on Equation 2.7 for all states, which gives us the following recursion:

$$V_{k+1}(s) = \max_{a \in A} \sum_{s' \in S} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V_k(s')] \tag{4.8}$$

The contraction mapping

$$\max_{s \in S} |V_{k+1}(s) - V^*(s)| \leq \gamma \max_{s \in S} |V_k(s) - V^*(s)| \quad (4.9)$$

implies that V_k converges to V^* as $k \to \infty$. At any point, we may choose to stop the iteration, and with our resulting V, we can compute the policy:

$$\pi(s) = \arg\max_{a \in A} \sum_{s' \in S} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')] \quad (4.10)$$

In our scout scenario, the one downside of this representation is that querying the policy online is cumbersome. The reward is calculated based on the sensor dynamics, which can be complicated. Therefore, we adopt the alternative but equivalent convention of computing the state-action value function, or sometimes called the Q-function, rather than the value function directly. Like the value function, the Q-function represents the value obtained by following a particular policy, but with respect to a given state and action from that state, rather than just the state. Therefore, the Q-function is related to a regular value function as follows:

$$Q(s,a) = \sum_{s' \in S} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')] \quad (4.11)$$

Substituting into Equation 2.10, the policy is simply

$$\pi(s) = \arg\max_{a \in A} Q(s,a) \quad (4.12)$$

which implies the inverse relationship between the value function and the Q-function:

$$V(s) = \max_{a \in A} Q(s,a) \quad (4.13)$$

Now, referring back to Equation 4.8, we can rewrite the value iteration step in terms of Q-functions:

$$Q_{k+1}(s,a) = \sum_{s' \in S} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma \max_{a' \in A} Q_k(s',a')] \quad (4.14)$$

Using Equation 4.14, we can perform value iteration with the same amount of effort as before, but now we store the result as a single Q-function. Now it is more convenient to query the policy, as Equation 4.12 shows that we just need to select the maximum over a small set of discrete actions.

Algorithm 1 details the entire value iteration algorithm formulated in terms of Q-functions. Each state value is initialized to zero. Using the state values, a state-action value is updated for each state-action pair using the Bellman Equation. The policy and state values are recalculated from the state-action values. This process repeats until a convergence threshold is met. Finally, the algorithm returns the Q-function.

This section introduced the classic value iteration algorithm. The algorithm essentially "learns" the best action to take in any state, using the mathematical property that iterating on the Bellman equation converges to the optimal value function. The advantage of this method is that it finds the optimal solution within epsilon tolerance. However, it is only practical for a small, discrete state space. Our scenario deals with a large, continuous state space. Nevertheless, value iteration is central to nearly all MDP solution

Algorithm 1 Exact value iteration algorithm for solving MDPs using Q-values.

```
ValueIterationQ(⟨S, A, P^a_ss', R^a_ss', γ⟩) ⇐ {
for all s ∈ S do
  V_0(s) ⇐ 0
end for
t ⇐ 0
repeat
  t ⇐ t + 1
  for all s ∈ S do
    for all a ∈ A do
      Q_t(s,a) ⇐ Σ_{s'∈S} P^a_ss' [R^a_ss' + γ V_{t-1}(s')]
    end for
    π_t(s) ⇐ arg max_a Q_t(s,a)
    V_t(s) ⇐ Q_t(s, π_t(s))
  end for
until max_s |V_t(s) − V_{t-1}(s)| < ε
return π_t
}
```

methods. We will use Algorithm 1 as a building block to the more sophisticated techniques discussed in Sections 4.2.3 and 4.2.4.

4.2.3 Approximation Architecture for MDP Value Iteration—The above formulation assumes a discrete state set S. If the state space is discrete, V and Q can be represented as a table of values, one record for each discrete state. The table of values is initialized arbitrarily and improved iteratively. The problem with this approach is that many real world state spaces are continuous and an acceptable discrete representation is intractably large. In many cases we cannot even store the huge table representation in memory. Additionally, to learn the value function, we must perform a value iteration backup for every single state, which would take far too long.

We address these complexities by representing Q using approximation architecture ($\widehat{Q}$). This architecture stores the state-action value function in a compressed form, such as a linear function over problem variables rather than an explicit combination of each possible variable assignment. When inserted in the value iteration process, we get an approximate value iteration procedure which alleviates the storage problem and shares information across state variables, thus decreasing learning time. The approximation architecture, therefore, solves both problems inherent to exact value iteration.

Algorithm 2 outlines the approximate value iteration algorithm where $\widehat{Q}$ is the approximate value function represented by an estimation architecture. Algorithm 2 takes as input the MDP tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where $|\mathcal{S}|$ is very large and possibly infinite. Algorithm 2 begins by randomly initializing state-action value table $\bar{Q}(s,a)$ over state subset $\bar{S}$. We then perform a Bellman backup over state subset $\bar{S}$, using $\widehat{Q}$ for future state-action value estimates. The newly computed state-action values are stored in table $\bar{Q}$. The policy, approximation architecture and state value table are then updated. This process repeats until a convergence threshold is met. Algorithm 2 returns approximation architecture $\widehat{Q}_t$.

Although this approximation yields better storage space and learning time, it may suffer from an inability to adequately represent the state-action value function. We acknowledge this by augmenting the state-action value function so that for every state-action pair, we return a distribution over the value rather than just a single number. Assuming normal distributions, we can write this in terms of a mean and variance:

$$\mathcal{N}(\mu_{s,a},\, \sigma^2_{s,a}) \Leftarrow \bar{Q}(s,a) \tag{4.15}$$

Thus, we incorporate representational uncertainty into the approximate state-action value iteration process so that we

Algorithm 2 Approximate state-action value iteration algorithm where $\widehat{Q}$ stands for an approximation architecture representation of the state-action value function, $\bar{Q}$ stands for a lookup table of state action values over a subset $\bar{S}$ of the full statespace S. The key changes from the exact algorithm are highlighted in green.

ApproxStateActionValueIteration$(\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle) \Leftarrow \{$
for all $s \in \bar{S} \subset S$ **do**
  for all $a \in A$ **do**
    Initialize $(\bar{Q}(s,a))$
  end for
end for
InitializeApproximationArchitecture$(\bar{Q}_t, \widehat{Q}_t)$
$t \Leftarrow 0$
repeat
  $t \Leftarrow t+1$
  for all $s \in \bar{S} \subset S$ **do**
    for all $a \in A$ **do**
      $\bar{Q}(s,a) \Leftarrow \sum_{s' \in S} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma \max_a \widehat{Q}(s',a)]$
    end for
  end for
  $\widehat{Q} \Leftarrow$ **UpdateApproximationArchitecture**$(\bar{Q}, \widehat{Q})$
  for all $s \in \bar{S}$ **do**
    $\pi(s) \Leftarrow \arg\max_a \widehat{Q}(s,a)$
    $\bar{V}_t(s) \Leftarrow \bar{Q}(s, \pi(s))$
  end for
until $\max_{s \in \mathcal{S}} |\bar{V}_t(s) - \bar{V}_{t-1}(s)| < \epsilon\ ,\ \forall s \in \bar{S}$
return $\widehat{Q}_t$
$\}$

learn a distribution estimate over the value. Our offline value function returns these distributions, not just an expectation function. This enables the online algorithm, when deciding between possible actions to execute, to specifically investigate the values of state-action pairs which are not well known *and* are likely to be viable alternatives. Stated another way, we can use an online search algorithm that is guided by the state-action value function. We discuss this next in Section 4.2.4.

Before moving on, we first summarize the offline scout algorithm, which is illustrated in Figure 4. The scouts use approximate dynamic programming to create a policy for acting in the world. A policy is a mapping from states to actions, which tells the scouts what to do in any situation. Their state-space includes not only location, pose and risk, but it also encompasses uncertainty in the risk map belief.

Computing a value function can be computationally expensive, so we compute this offline approximately through value iteration before the mission starts. We do so by simulating the scout flying to explore where the greatest reward lies, and we save snapshots of this simulation as data points for our $\bar{Q}$ table. We then generate an approximation architecture on each iteration by regressing over these data points, taking into account the representational uncertainty.

4.2.4 Online Planning and Re-Optimization Algorithms—In this section we highlight the weaknesses of the previous algorithm and explain how it can be augmented with a search algorithm. The previous algorithm computes a policy offline. Another approach is to compute an action online, just for the current situation. We actually propose both. We suggest using the offline policy as a starting point and then improving it online, tailored to the current situation.

In this section we summarize why we need online planning and re-optimization. In short, re-evaluating actions over a pre-determined horizon helps to reduce the approximation error, which helps us select the best action to ultimately execute. This re-evaluation turns out to be more accurate than the off-line state-action value, because the off-line function is necessarily compressed and therefore not expressive enough to capture every detail. The discussion includes how we use the state-action value function uncertainty to select actions for re-evaluation. In particular, we will describe a family of tree-search algorithms which use the state-action value function to determine which branches of the tree to search.

Section 4.2.3 outlines a method for computing the state-action value function. Computing the state-action value function is tantamount to computing a policy because it tells you how good each currently available action is given your current state or situation. In other words, it tells you how good each action is and you simply have to select and execute the best one. However, the resulting state-action value estimates are not as accurate as they could be. The off-line generated function is necessarily compressed and therefore not expressive enough to capture every detail. Therefore, instead of using it raw, we improve upon the estimates by performing some additional simulations online. Recall that the state-action value function covers the entire state-space. In other words, it will provide an estimate of the action values emanating from any state or situation we may encounter. This is a challenging request. In contrast, our situation is a far less daunting task. However, since we do it in real-time, we do not have as much time to complete that task. Our approach attempts to make the best trade-off between online and offline computation by computing a state-action value function offline and using it as a starting point for our online algorithm.

.

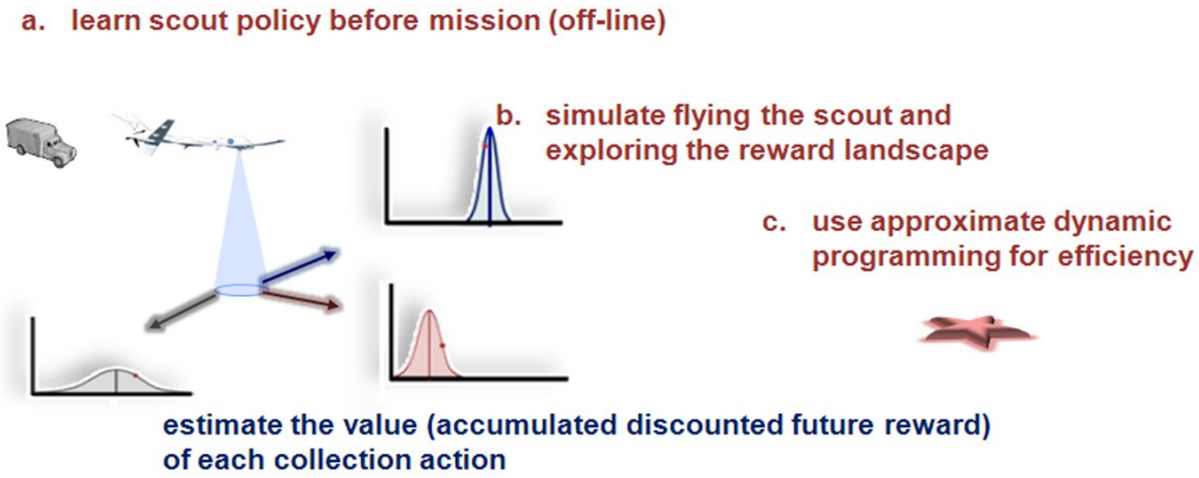


Figure 4 - Visualization of Q-function in the Approximation Architecture with Uncertainty

The process of online reevaluation involves generating a tree of candidate scout vehicle paths to test (Figure 5). We generate the tree of paths with the help of the offline generated state-action value function. Starting from the current state, the algorithm selects b actions to simulate, where b is called the branching factor. After simulating the actions, the algorithm will reach b new states. From each of those new states, the algorithm again selects b actions to simulate. After two steps, the algorithm will reach b^2 new states. This process proceeds to a predetermined planning horizon h. Collectively, the algorithm simulates b^h paths. The algorithm's task is to re-estimate how much each scout collection path will improve our knowledge about the risks to the mission vehicle. We think of this increase in knowledge as a reward in accordance with the MDP framework. Thus, the algorithm adds up the cumulative reward garnered from each path. If the planning horizon is long enough to reach the end of the mission, then the simulated accumulated reward is used as the value of the initial action. If the planning horizon is not long enough to reach the end of the mission, then the remaining value is estimated using the offline state-action value function.

The above algorithm uses branching factor b and horizon h. We select b and h such that we have enough time to re-evaluate b^h paths. To do this, we first determine how many computations can be performed in the allotted time between decisions. We then select a planning horizon and calculate a branching factor which will result in that number of calculations.

A central question in this process is how to choose the actions to reevaluate. As stated above, we generate the tree of paths with the help of the offline generated state-action value function and we select b and h such that we have enough time to re-evaluate b^h paths. One assumption in this process is that we do not have enough time to reevaluate every action over the planning horizon. Therefore, we elect to evaluate actions that have a promising outcome, with consideration for how sure we are about our estimates of those outcomes.

At each step in the algorithm described above, we select b actions to simulate. This selection uses the offline state-action value function. The function provides an estimate of the value (future cumulative reward) of each action, if taken from the current state. The estimate actually includes a distribution described by a mean and variance. The distribution captures how well we know a given value. A high variance distribution means that we do not know that value very well. Likewise, a low variance distribution means that we know that value rather precisely.

We select a sample from these distributions, one for each action, and then choose the action with the highest sample value. If one of the action distributions consistently produces a high sample value, this tells us that we have little reason to evaluate other options.

However, if there is a state-action value distribution with an especially high variance, it will sometimes produce a sample

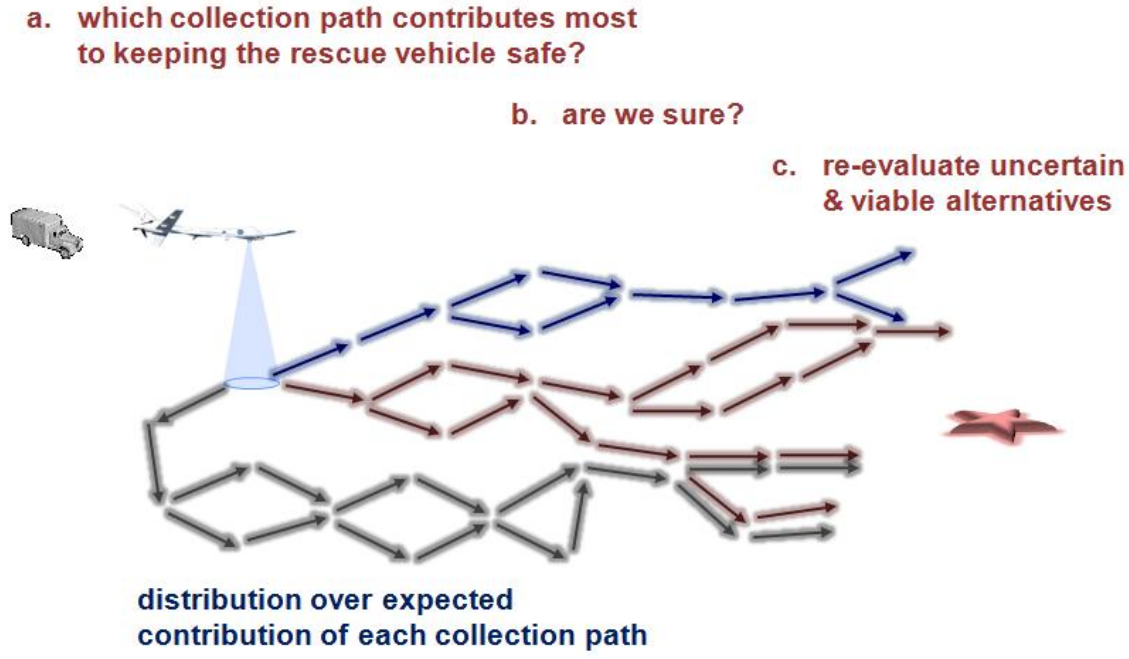


Figure 5 - Process of online reevaluation involves generating a tree of candidate scout vehicle paths to test.

with the highest value even though its' mean is lower. This phenomenon exactly mirrors the probability that said action is the best, given what we know. In other words, we explore the actions in proportion to how good they are and how certain we are about that. In summary, we determine which actions to re-evaluate by representing the uncertainty about their true value. This distribution is used to select actions (Figure 5) for re-evaluation which appear to be good but the truth is uncertain. We may also evaluate poorer looking plans which do not look very good, yet the true value has such high variance that it warrants a second look.

To summarize, our method allows us to use offline knowledge and processing to guide our scouts online. The way we construct the offline policy informs the additional online processing. In this way, we can exploit both on and offline control processing in a complimentary way. We can compare this to how people deal with a planning task. Suppose you are navigating home from work. You have a policy based on experience, which tells you which road to take from a given intersection. However, given the time of day you may wish to consider alternatives. For example, given the traffic, you may be unsure if your standard policy is optimal. Therefore, you can simulate in your head the consequences of some alternative routes. The result of the simulation tells you which way to go (what action to take) right now. Your mental simulations will surely be biased toward perceived viable alternatives. In other words, you will consider some alternatives and others you will not. In this way, your offline knowledge informs your online situational reevaluation.

The above process actual describes a family of algorithms because changing our search horizon and branching factor fundamentally changes the algorithm. For example, if we use a very small branching factor with a long horizon, the algorithm closely resembles the rollout algorithm. Rollout is a longstanding method of evaluating moves in the game of backgammon [5]. On the other hand using a short horizon with a large branching factor closely resembles model predictive control. Different configurations will work better for different applications as well as at different points in the mission. For example, toward the end of a mission, it could be helpful to use a wider search (larger branching factor) to make sure that we rightfully consider the end goal. Further extensions to our algorithm may include using different branching factors at different levels of the search tree. For example, it is easier for a function to capture long term objectives then short term details. Therefore, this small extension would allow the algorithm to rely more on offline knowledge in the middle of the mission and rely more on simulation for starting and ending the mission. All in all, our method describes a broad family of algorithms via a set of configuration parameters. Describing the algorithm in this way enables us to tune it, making it broadly applicable to many applications.

5. ALGORITHM DEMONSTRATION RESULTS

At this stage, we are currently designing and implementing the Logistics Executive and Adjustable Autonomy components. We have implemented the scout path planning algorithm and run preliminary tests on it, with results discussed in this section. Section 4.2 above describes the theory underlying the scout path-planning problem, the offline value iteration procedure, and the online search for decision-making. In this section, we visualize the latter two to demonstrate the effectiveness of this theory.

To show how our scout finds efficient sensing paths, we center our discussion on four example scenarios. Each scenario tasks the scout to reduce variance within a 10-by-10 area, but the areas of high variance differ across scenarios. Scenario A has high variance in the southwest quarter patch and zero variance (i.e. perfect knowledge) everywhere else. This corresponds to a very large feature in one corner of the map which we have no information about, and everywhere else we have perfect information already. In scenario B, we split the uncertain feature into two smaller features, one in the southwest corner and the other in the northeast. Scenario C involves a feature in the middle that we know a great deal about (i.e. low variance) but we grow more uncertain the farther we go toward the edges. Finally, scenario D randomly smears high variance across the entire map, simulating a realistic setting where we may have poor prior information of a certain region.

Each scenario was run with a mission length of 25 time steps. Within each scenario's context, we first illustrate the evolution and convergence of the approximate value function during value iteration. We use 100 samples to represent the state subset. Note that there are 100 unique grid cells the scout can be in, four possible orientations in each, and an infinite number of map beliefs possible. Thus, our value function representation is extremely sparse relative to the actual state space. Then, we display the paths constructed during online execution to show how the scout chooses to survey areas with higher uncertainty within the time allotted. The tree search algorithm is limited to 20 node traversals of computation, but searches down to a depth of seven nodes.

5.1 Scenario A

Figure 6 below depicts the state-action value table $\bar{Q}$ evolving over ten sets of value iteration. The x-axis represents different sample states in our lookup table, and the y-axis shows the value associated with that state. When querying the value of a state, we are not querying $\bar{Q}$, but rather $\tilde{Q}$ representing the estimation architecture which interpolates over the sample states in the $\bar{Q}$ table. However, to aid conceptual convenience and transparency, we will

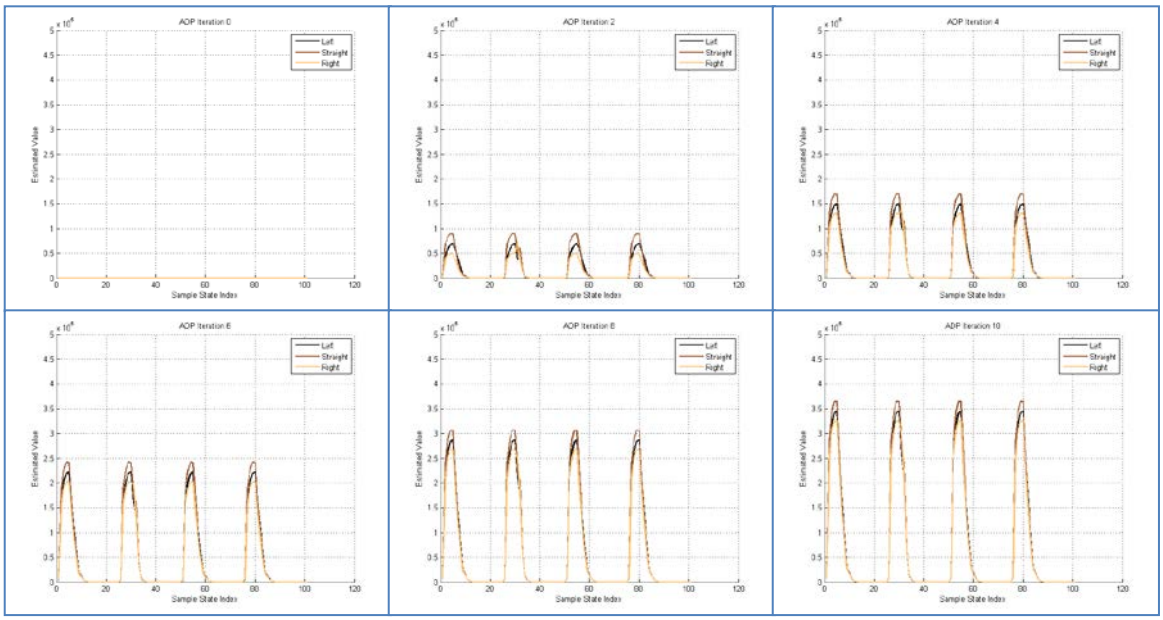


Figure 6 - Value Function Representation for Scenario A

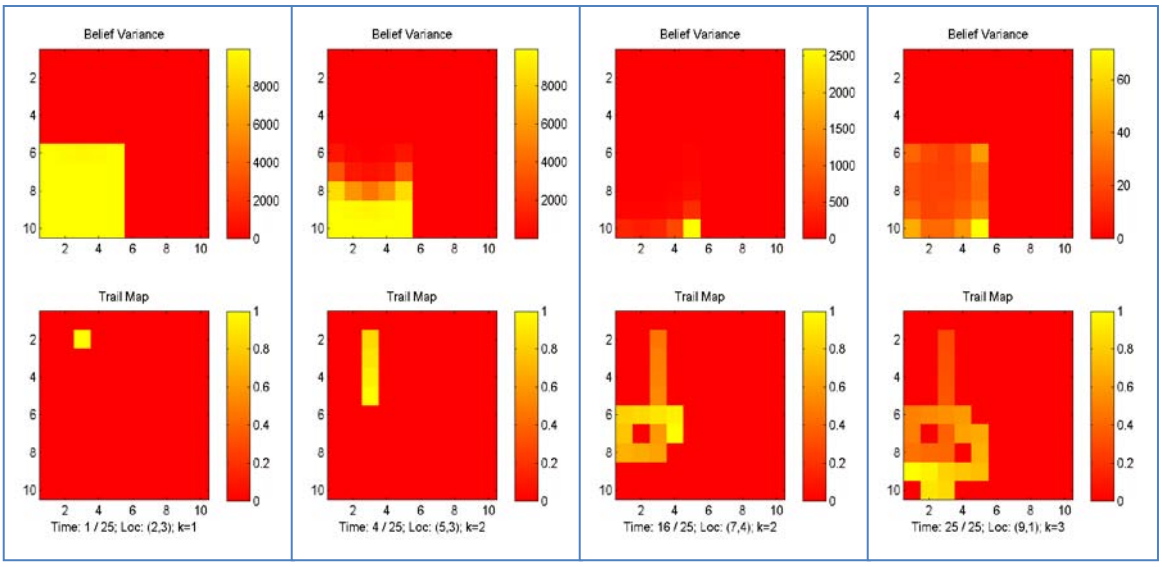


Figure 7 - Belief Variance and Scout Path for Scenario A

refer to the $\bar{Q}$ plots as the value function plots. The sample states were constructed by initializing a simulated scenario four times and letting the scout fly a pre-determined lawnmower pattern which sweeps across the area for the length of the mission, 25 time steps. To avoid gathering the exact same data each time, stochasticity was introduced into the path, more with each subsequent pass.

The resulting representation of $\bar{Q}$ on the even iterations are shown in sequence on value plots in Figure 6. At first, the values for each sample state are initialized with low random noise, which is not visible at the scale shown. In subsequent iterations, the values accrue at each step, since each state "looks ahead" to the next best state and adds that state's value to its own reward (i.e. variance reduction) for taking the action leading into that state. The values gradually converge (i.e. the increase at each step gets smaller) since the accruement is increasingly discounted over later iterations. This is consistent with equation (1.8) in the Scout Planner section. However, the most interesting parts of these plots are the four peaks, corresponding to when the scout passes over the patch of high variance and hence gathers the

most reward. Afterward, the reward tapers off since the scout's sensor coverage is overlapping areas just previously seen. This illustrates how the value function effectively encodes and exploits the structure of belief variance in the scenario.

We have shown the construction of the path at time-steps 1, 4, 16, and 25 of the algorithm's run in Figure 7. The top plot in each frame shows the belief variance with a color scale on the side, and the bottom plot the scout's path so far. As can been seen, the scout takes the reasonable action to plunge south into the area of highest uncertainty and then loops through it until the end of the mission. Note how the scout "carves away" at the belief variance in the top plots. We rescale the colors so that areas with the highest remaining variance always appear yellow, and hence one can see how they guide and attract the scout. However, one should note how much the scale has changed by the end of the mission, thus showing the extent of variance reduction.

5.2 Scenario B

Figure 8 shows the value function for two uncertain features in this scenario. For each of the four runs, the value function peaks in two places, corresponding to the two features. However, the first peak in each of the first three runs is slightly taller than the second peak. This is due to the width of the sensor footprint. When it is primarily sensing the first feature, it also captures a little bit of the second, given their proximity. Thus, when it actually flies over the second, it will not generate as much variance reduction as the first. Since there are only these two features, the effects of stochasticity in generating the sample states are not very pronounced. However, as the stochasticity is turned up with each pass, we notice the peaks for the last pass look substantially different than the first three. This indicates we have allowed ourselves to learn about the features from a slightly different perspective, which gives us a richer representation.

Now we turn to the scout's path shown in Figure 9. The scout begins by heading to the feature in the southwest corner. Not only is it slightly closer to the scout's initial position, but the value function also suggests it is of higher value than the other features. This is an artifact of our approximation architecture, but it is a small price to pay and easily mitigated on average with stochasticity introduced during the online algorithm. Yet, the scout does not go all the way to the bottom feature because as it approaches, its sensor footprint already provides good coverage of the feature. Thus, two steps away, it turns toward the northeast feature.

Similarly, it does not need to head all the way east. About halfway through the mission, it turns back again to better understand the first feature, rather than spend all its remaining time around the second feature and not reducing overall variance as much. This kind of behavior illustrates the value of the online search algorithm, for knowing when to turn back cannot be encoded in the value function itself. The look-ahead feature of search allows the scout to plan based on how far ahead its mission end is. Here we used a search depth of seven. Had we searched even deeper, we

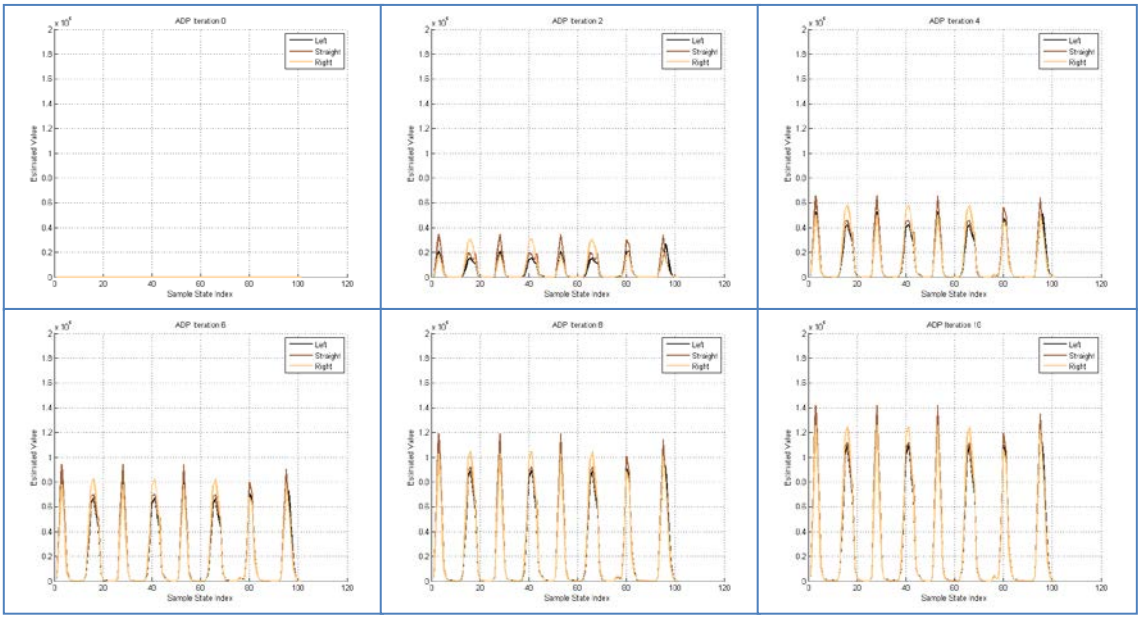


Figure 8 – Value Function Representation for Scenario B

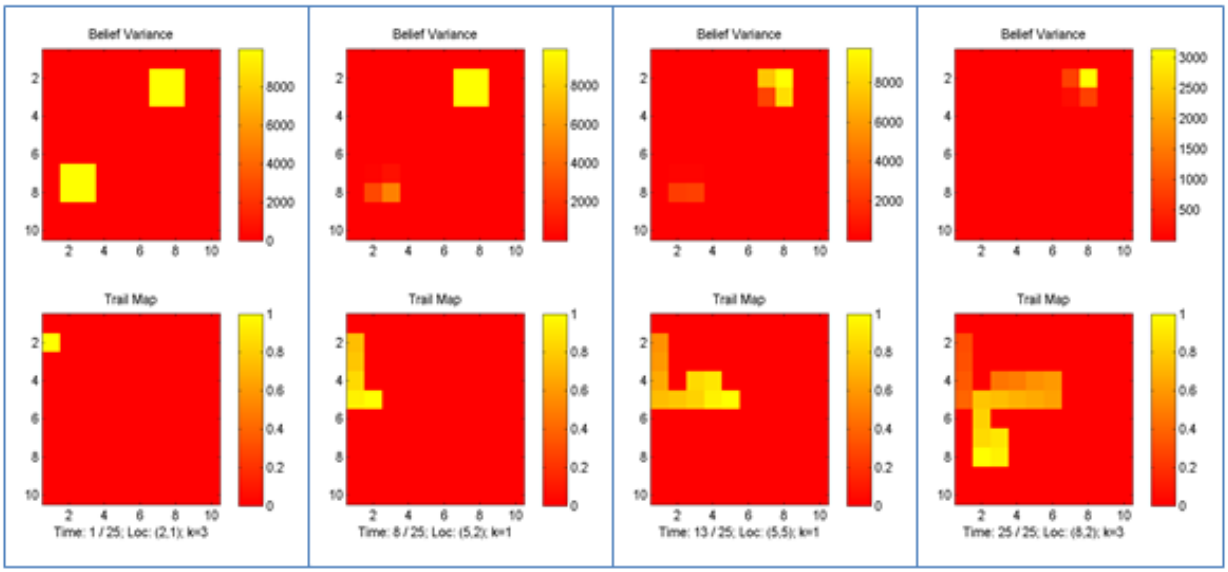


Figure 9 – Belief Variance and Scout Path for Scenario B

may have arrived at the more efficient solution of spending half the time at each feature and not have to turn back. However, searching deeper costs time. If we are to maintain the same online computational time, then searching too deep would prune the branching factor and thus degrade the quality of search.

5.3 Scenario C

In this scenario, we have the highest variance on the boundaries, and it decreases as we approach the center. Since we are flying a lawnmower pattern over the area, we notice periodic peaks as we approach the edges in the value function plots shown in Figure 10. As we make sweeps over the central area where variance is lower, we notice a decreasing trend in the values, but it picks back up as we start to swing closer to the opposite edge. The high initial peak at each run is due to the scout sensing completely new terrain; it corresponds to states where the scout is in the top-left corner and the belief map variance is high throughout the region covered by the sensor footprint. Since there is much more total variance spread across the region in this scenario than in A and B, the effects of stochasticity are much more clearly seen. While the general trend for the values across each run is similar, the details are quite different. This captures a much richer and more informative representation of the value function.

The reasonable thing for the scout in this scenario is to hug the edges where the variance is highest. This is exactly what Figure 11 shows. However, it does not just hug the edges statically, but it dynamically tries to go inside a bit, too, where valuable information also exists. Given the time it has, the scout tries to balance how much time it spends closer to the outside and inside. In 25 time-steps, it is infeasible for the scout to make a full circuit around the perimeter, which is 36 grid cells. With an online look-head depth of seven, the scout realizes in the south leg of its path that it is more valuable to maintain a distance of one cell away from the south edge rather than travel right on the edge. Also, in this path, we can notice stochasticity in the online process. On the third step of its path, the scout turns north toward the edge rather than continue on its present path east. This is clearly suboptimal, and it could only have arisen by "mistake." The "mistake" arises from the representational uncertainty in the value function approximation architecture $\widehat{Q}$. We encode uncertainty into the value function because we know it is not perfect. However, this means each query to the value function is a probabilistic draw from a distribution characterized by this uncertainty. Thus, we have to submit to the possibility of a clearly suboptimal decision every once in a while. As the path shows, though, these mistakes are uncommon and quickly recovered from.

5.4 Scenario D

When we have random high variance everywhere in the graph, the corresponding value function shown in Figure 12 is messy and difficult to interpret by hand. It bears some resemblance to Scenario C's value function, since there is value everywhere in the region. However, it is substantially less structured. Nevertheless, the value function still encodes the random variations in this map as well as the larger structure, and this will be apparent in the path the scout chooses.

Similar to the plots for Scenario C, Figure 13 shows that the

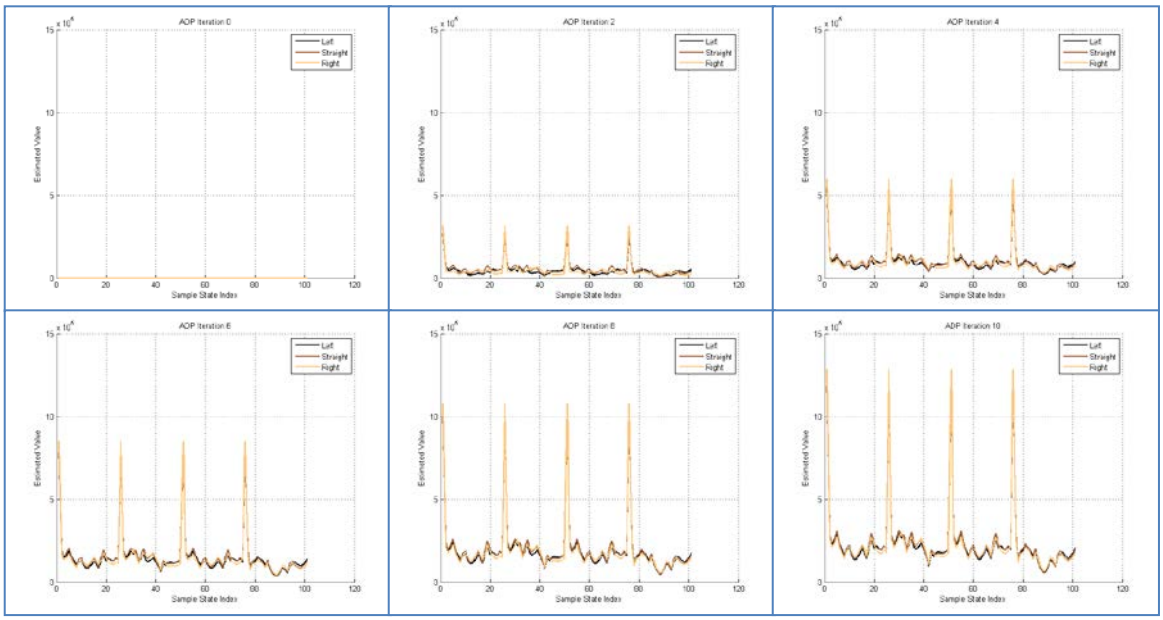


Figure 10 – Value Function Representation for Scenario C

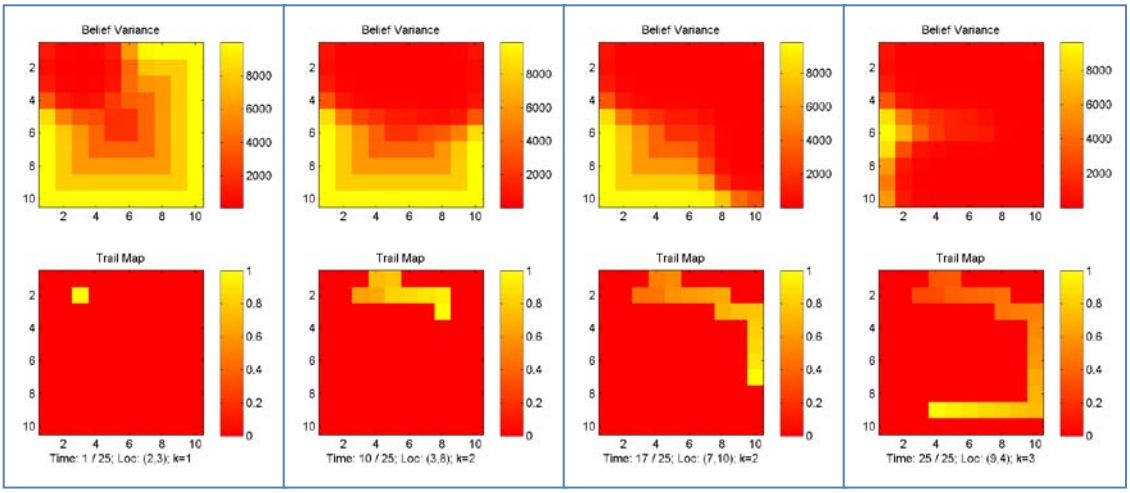


Figure 11 - Belief Variance and Scout Path for Scenario C

scout chooses to fly a circuit around the region to provide greatest coverage. As before, the scout does not have enough time to fly cover every piece of high uncertainty, and thus the scale at the end is much larger than the scale for Scenarios A and B. However, the path leaves less uncertainty uncovered than in Scenario C because the scout can afford to hug the outer edges less and thus has less distance to travel to make a circuit. Note that the scout does not even have to complete the circuit or else it would overlap with its initial sensor footprint. It is worth noting that in running multiple trials for each of these scenarios, Scenario D showed the greatest variation in the planned path. That is, while the paths for the other scenarios followed predictable and expected patterns, sans the occasional "mistake," the paths for Scenario D may choose to wander in toward the center or wander out at any given point. This suggests there is larger representational uncertainty in the value function, which is to be expected in a less structured environment. It means multiple good solutions may exist, and since our representation models this aspect, we do not limit ourselves to a single deterministic path in any scenario.

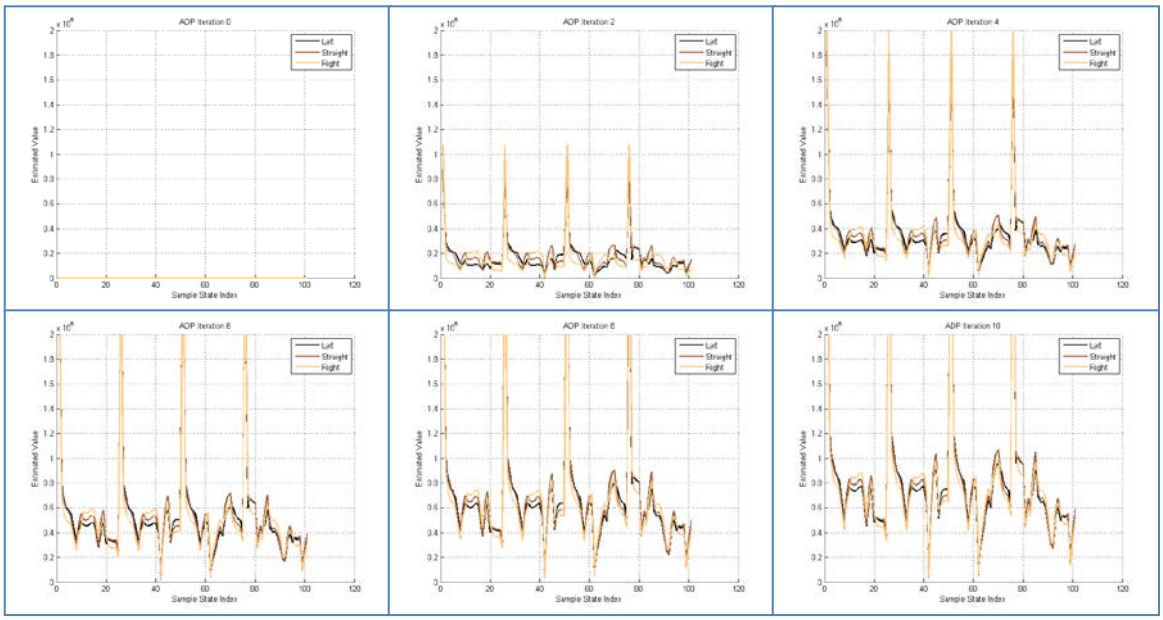


Figure 12 - Value Function Representation for Scenario D

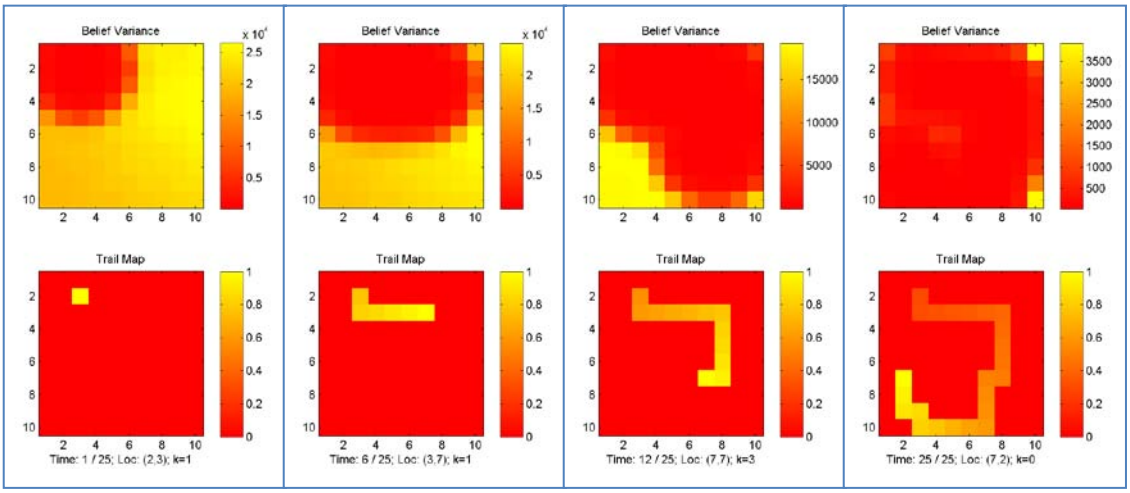


Figure 13 - Belief Variance and Scout Path for Scenario D

5.5 Algorithm Summary

These examples show that our scout planning algorithm is capable of finding a path through an area which purposefully surveys the most uncertain features, thus generating the most valuable data for the logistics planner. This is accomplished through a combination of the offline value iteration and online search procedures discussed in Section 4.2. The flexibility of our algorithm arises from the Markov Decision Process framework, which easily adapts to any given scenario. An important detail of our approach is that since we cannot exactly represent the value function, we acknowledge it by introducing stochasticity into our decision-making. Thus, our non-deterministic solutions, while rarely optimal, are more robust to this uncertainty. When integrated with the entire ARCAL system, the scouts shall be valuable additions to the logistics vehicle.

6. SUMMARY

In the future, unmanned platforms will gain high-order decision-making intelligence, form teams, and perform collaborative tasks. However, for successful field

deployment, operators will need confidence that autonomous decision-making leads to proper behaviors, especially in uncertain environments.

To address this issue, the ARCAL project proposes two complimentary areas of research which when combined can increase operator confidence in future autonomous system behaviors. The first incorporates concepts of risk-based adjustable autonomy with risk verification within system functions and task-directed adaptive search techniques. An operator can adjust the autonomy level, employ autonomy functions in which he has confidence in success, or can revert to fully manual control at any time. The second involves new methods to effectively test and evaluate collaborative autonomous team behaviors prior to field deployment.

In this paper we presented simulation results. However, we plan to implement the entire scenario on a test bed consisting of a radio controlled ground vehicle and two Parrot ARDrone quadrotors (Figure 14). Building the framework for the test bed commenced last summer (2011) and is ongoing.

To develop and validate the concepts and technologies used in this project a natural disaster recovery scenario has been employed as an example. A team of UASs is dispatched to determine the safest and fastest path for a disaster recovery convoy to deliver relief supplies. Although a natural disaster recovery scenario is used as an example, the autonomous algorithms, concepts and technologies being developed can certainly be used for other scenarios with the UAS Simulation Environment as well.

The theoretical basis for adjustable autonomy used during control and supervision of a team of UASs performing a collaborative task with innovations of applying risk to mission goals is introduced. Task-directed search algorithms for UAS scout path-planning, used to improve knowledge of the risks to the mission, are developed and implemented. An algorithm test battery was developed and used to run tests on the scout path-planning algorithms. Test results from a number of runs with the algorithm applied to terrain examples with various belief uncertainty concentrations are shown and described. An ARCAL systems architecture is developed incorporating these autonomy algorithms as applied to the natural disaster recovery scenario example. Initial results are encouraging showing that the algorithm performs quite effectively.

Although considerable autonomous systems research has been performed and progress made over the past decade, current unmanned systems are still for the most part tele-operated and manpower intensive relying on human operators and their decision-making capabilities to perform platform and mission tasks. However as unmanned systems become more complex and costs continue to rise for specialized training and deployment of operators, incorporation of autonomous capabilities is designated to assist operators and lighten their task load. Autonomous functions will need to be robust, reliable, exhibit the correct behaviors for the situation and operators will need confidence that the mission can be performed effectively.

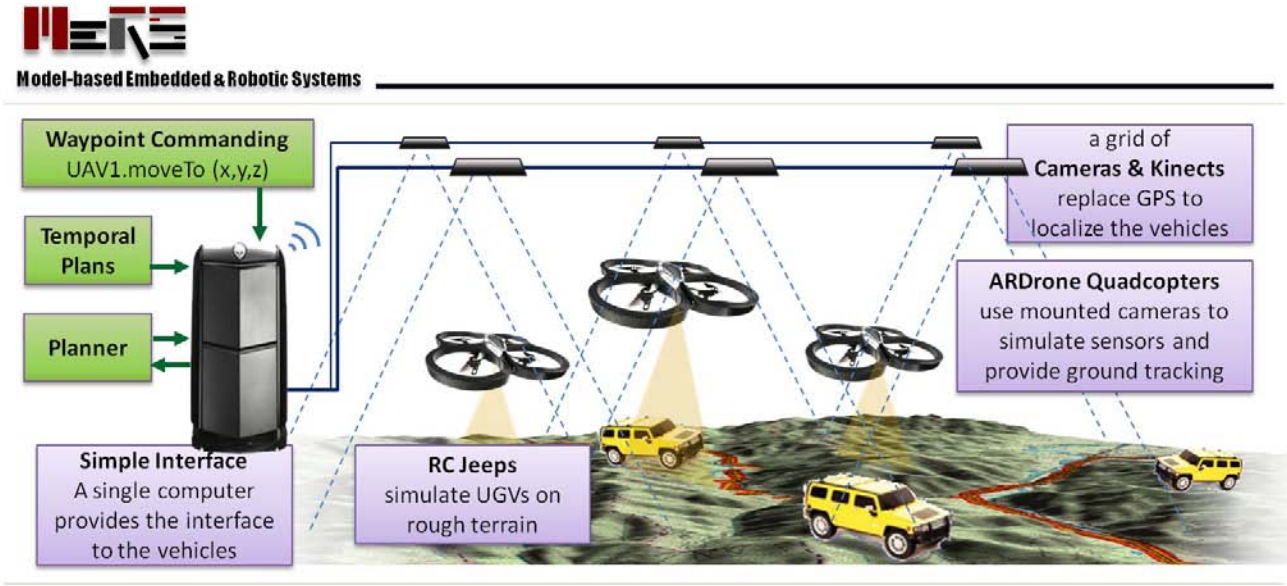


Figure 14 – Heterogeneous Robotics Test Bed

REFERENCES

[1] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, 3rd ed. Vol. 1, MIT Press, Cambridge MA, 2005.

[2] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, 3rd ed. Vol. 2, MIT Press, Cambridge MA, 2007.

[3] D. P. Bertsekas and D. A. Castanon, *Rollout Algorithms for Stochastic Scheduling Problems,* Journal of Heuristics; 5 (1); 89-108; 1999.

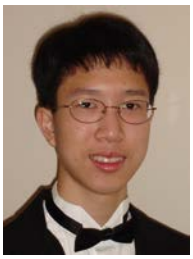
[4] R. S. Sutton and A.G. Barto, *Reinforcement Learning: An Introduction*, 1998.

[5] G. Tesauro, "Temporal Difference Learning and TD-Gammon," *Communications of the ACM*, 1995.

[6] J. N. Tsitsiklis and B. Van Roy, "Feature-based methods for large scale dynamic programming," *Machine Learning,* 22(1):59–94, 1996.

BIOGRAPHIES

Lawrence A. M. Bush *holds a Bachelor of Science in Industrial Engineering and Operations Research from the University at Buffalo. Larry has a Master of Science in Computer Science from Rensselaer Polytechnic Institute. He is a Ph.D. Candidate at the Massachusetts Institute of Technology in the Department of Aeronautics and Astronautics and the Computer Science and Artificial Intelligence Laboratory (CSAIL). Larry's thesis topic is Decision Uncertainty Minimization for Sensing Missions. His areas of expertise are pattern recognition, optimization, active learning and active sensing. His prior employment includes work on expert systems at Cornell University, New York State Agricultural Research Station and work on machine learning at MIT Lincoln Laboratory.*



Andrew J. Wang *earned Bachelor of Science degrees in Aerospace and Electrical Engineering & Computer Science from the Massachusetts Institute of Technology in 2011. He is currently a Master's of Engineering student at MIT's Computer Science and Artificial Intelligence Lab (CSAIL) with plans to pursue a Ph.D. Andrew's research interests involve autonomous decision-making for transportation and exploration vehicles as well as for communications and energy infrastructures.*

Brian C. Williams *leads the Model-based Embedded and Robotic Systems group, within the Computer Science and Artificial Intelligence Laboratory (CSAIL) at the Massachusetts Institute of Technology. His research concentrates on model-based autonomy -- the creation of long-lived systems that explore autonomously, while commanding, diagnosing and repairing themselves using fast, commonsense reasoning.*

Professor Williams received his S.B., S.M and Ph.D. in Computer Science and Electrical Engineering at MIT, and worked at the Xerox Palo Alto Research Center and NASA Ames Research Center, prior to joining the faculty at MIT. He is a pioneer in the fields of qualitative reasoning, model-based diagnosis and autonomous systems. He received a NASA Space Act Award for Remote Agent, the first fully autonomous, self-repairing space explorer, demonstrated onboard the NASA Deep Space One probe in May, 1999. He was a member of the Tom Young Blue Ribbon Team in 2000, assessing future Mars missions in light of the Mars Climate Orbiter and Polar Lander incidents, and is currently a member of the Advisory Council of the NASA Jet Propulsion Laboratory at Caltech. He has won four best paper prizes for his research in diagnosis, qualitative algebras, propositional inference and soft constraints. He is a fellow of AAAI, has served as guest editor of the Artificial Intelligence Journal and has been on the editorial boards of the Journal of Artificial Intelligence Research, and MIT Press.