

Belief Space Hierarchical Planning in the Now for Unmanned Aerial Vehicles

Caris M. Moses*

Northeastern University, Boston, MA, 02115, USA

Rahul Chipalkatty†

Draper, Cambridge, MA, 02139, USA

Robert Platt‡

Northeastern University, Boston, MA, 02115, USA

I. Introduction

PLANNING long duration missions for autonomous unmanned aerial vehicles (UAVs) in dynamic environments has proven to be a challenging problem. Autonomous UAVs must be able to reason about how to best accomplish mission objectives in the face of evolving mission conditions and environmental uncertainty. For example, tactical UAV missions consist of executing multiple interdependent tasks such as: locating, identifying, and prosecuting targets; responding to dynamic (i.e. pop-up) threats; motion planning with kinematic and dynamic constraints; and/or acting as a communication relay.¹ The resulting planning problem is then one over a large and stochastic state space due to the size of the mission environment and the number of (un)known objects within that environment. The world state is also only partially observable due to the inherent limitations and stochastic nature of sensing and actuation. This requires us to plan in the belief space, which is a probability distribution over all possible states. Some *a priori* contextual world knowledge, like terrain maps, target locations, and threats, is available via satellite imagery based maps. But it is likely threat and target information will be “old” data by execution time in a rapidly changing mission environment. This makes classic approaches to *a priori* task, or symbolic, planning a poor choice of tool in that task dependent decisions are made without full knowledge of the mission environment. In addition, task planners traditionally do not have methods for tightly coupling geometric planning problems with the high level task planning, as most approaches treat these as separate planning problems. However, modern belief space geometric planning tools, like Partially Observable Markov Decision Problem (POMDP) based formulations, become intractable for large state spaces, such as the tactical UAV mission discussed in this paper.

Simply developing a set of ordered tasks, as those produced by purely symbolic approaches like hybrid hierarchical task networks,² provides a means of generating high-level plans in domains where many different types of actions are possible, as in our domain. But, those methods do not take into account the geometric constraints that may arise while trying to perform a mission task (e.g. fly a path from point A to point B subject to our knowledge regarding potential threats in the world). Unstructured and partially observable mission environments with unexpected events create the need for a tactical UAV to reason in both the domain geometry as well as the task space. For example, if a UAV is tasked with locating a target, it must ensure that there are no threats preventing the completion of this task and monitor this condition throughout the flight. If a threat does pop-up, it must be determined if its location endangers the mission with some probability. If it does, then the threat must be avoided or additional actions must be planned to neutralize the danger with some certainty. All of these potential responses, or additional actions, require motion planning, thus requiring both probabilistic symbolic and geometric reasoning.

Geometric planning quickly becomes intractable in partially observable domains, or belief spaces,^{3,4} with long planning horizons. However, recent tools in the domain of robotic manipulation have approached

*moses.c@husky.neu.edu

†rchipalkatty@draper.com

‡rplatt@ccs.neu.edu

this problem by combining symbolic and geometric planning paradigms.^{5–7} One in particular, Hierarchical Planning-in-the-Now in belief space⁸ (BHPN) is a hierarchical planning technique that tightly couples geometric motion planning in belief spaces with symbolic task planning, providing a method for turning large-scale intractable belief space problems into smaller tractable ones. Another facet of this technique is that it is “aggressively hierarchical”, whereby detailed planning is delayed until it is required, hence the moniker “in the now.” It achieves this by adopting “levels” of task precondition priority.⁹ Thus abstract plans can be constructed initially (e.g. look for target, then fly back to base), while more detailed portions of the plan can be constructed *in situ* as more of the mission environment is considered. BHPN also enables functionality to plan for observations that will improve belief space certainty. The trade-off of using this method as compared to more traditional purely symbolic planning or belief space planning methods is that the resulting plans are not optimal with no bounds on sub-optimality. The plans are simply feasible in that the goal objectives are satisfied in the face of a large state space and unknown environment, which is beneficial given this class of problems. In finite domains, completeness and correctness is also guaranteed.⁵

This paper aims to explore the application of BHPN to a UAV domain as opposed to the manipulation domain of the original work. We give the operators used during mission planning, the UAV observation model which is instrumental during the regression search for feasible plans, and an example of a mission plan developed using our implementation of BHPN.

In Section II we discuss background information such as related work and the BHPN algorithm. In Section III we outline the reapplication of BHPN to a UAV domain. We will describe the realistic, physics-based simulation of our work in Section IV. Lastly, in Section V we give an evaluation of the BHPN algorithm and how it compares to other planning methods.

II. Background

A. Related Work

Integrated task and motion planning requires a method of developing feasible actions which manipulate the world at a high level, as well as a method of planning efficient motions at a low level. In this related work section we will address task planning methods in Section 1, and motion planning methods in Section 2. We will look at methods other than BHPN which handle the integration of the two in Section 3. A partially observable Markov decision processes, or POMDP, is a method of developing an optimal policy to achieve a goal in a partially observable domain similar to ours. However, solving a POMDP in our mission environment would be intractable. Section 4 gives a detailed explanation of different POMDP approximations that have been developed to overcome this issue that could potentially be used to solve problems in our mission environment.

1. Task Planning

Task planning consists of searching for an action sequence which will transform the current world state into a goal state. STRIPS¹⁰ (Stanford Research Institute Problem Solver) was the first widely used task planner. STRIPS employs a method of classical planning where the action sequence is found using backward-chaining state space search. Heuristics can be used to speed up this process such as partial-order planning¹¹ which loosens restrictions on the order in which actions must be executed. But overall, classical planning is less efficient than hierarchical planning.¹² Hierarchical methods, such as HTN² (hierarchical task networks), provide significant speedups. HTN uses a combination of primitive tasks, compound tasks, and goal tasks to reduce the problem solving effort. A shortcoming of all aforementioned task planners is that they do not provide a means for executing an action. For example, if a task is to place block 1 on top of block 2, it will not provide a trajectory for the robot arm to follow as it places the block, or a grasp with which to hold the block. For that we require a low-level motion planner.

2. Motion Planning with Uncertainty

The field of motion planning is rich with proposed methods, particularly where they apply to uncertain domains. One method is to solve a POMDP.^{13,14} However, solving a POMDP is generally computationally intractable.¹⁵ A common approximation is to use receding horizon control as opposed to solving for an infinite time horizon.¹⁶ Another approximation is nominal belief optimization which uses a finite-dimensional

representation of the belief space and provides a simplification to the objective function which is specific to the tracking problem.³ More POMDP approximations will be given in Section 4.

Another method of motion planning in belief space for UAVs is to minimize the mean time to detection, or the expected time to find a target.^{4,17} The cost function is dependent on action sequences, and the solution is quasi-optimal. As with POMDPs, the solution approaches the optimal solution as more actions are considered when looking ahead. Entropy, or the uncertainty associated with a target's location, can also guide a UAV's search.¹⁸

3. Integrated Task and Motion Planning

Task planning gives a practical method for describing and planning in an environment at a high level. It is well suited for static environments with simple actions where little geometric knowledge is required. These planners require *a priori* knowledge of the domain which becomes obsolete in the face of a rapidly changing environment (on both a logical and geometric level). On the other hand, purely geometric planners are effective at providing a means of motion through a potentially dynamic and stochastic world. However, geometric planning techniques quickly become intractable for long duration missions with rapidly evolving environments. They also have no method for interpreting possible actions other than those pertaining to motion. For these reasons, previous work has sought to combine task and motion planning. In these domains there are many high-level tasks that need to be considered and low-level motions (e.g. grasps, arm trajectories or paths) that need to be planned for.

Most recent work in the field of integrated task and motion planning is applied to mobile manipulation in unstructured environments. Some scenarios which have been experimented with include cooking a meal,¹⁹ pushing buttons to dial a number,²⁰ and performing other household tasks.⁵ The key to integrated planning is to develop an intelligent connection between logical descriptors and the continuous geometry of the real world. One method plans in the configuration space and initially assumes all motions are valid, then replans closer to execution time to ensure that the assumption is still true.⁶ Another method speeds up the motion planning process by caching previously calculated motions and using them in future planning problems.⁷ Guitton and Fergus give an approach for translating geometric constraints into mathematical penalty functions and use non-linear programming to solve them in the context of HTN planning.²¹ Lastly, Hauser proposes a probabilistic tree-of-roadmaps planner which samples the task and motion space to develop feasible plans.²²

4. POMDPs

In domains where the world state is only partially observable and actions are non-deterministic, a common method of choosing optimal actions is to solve a POMDP. However, solving a POMDP for large state spaces is intractable as it suffers from both the "curse of dimensionality" and the "curse of history." Since the state is only partially observable, it is represented as a probability distribution over all possible states, referred to as a belief state. The "curse of dimensionality" is due to the fact that the size of the belief space grows exponentially with the size of the state space. A history is the set of all actions and observations taken up to the current time step. The "curse of history" is due to the fact that the number of these action-observation pairs used when solving for the optimal solution to a POMDP grows exponentially with the planning time horizon. To solve a finite-horizon POMDP is PSPACE-complete, and to solve for an infinite-horizon POMDP it is undecidable.²³

The optimal value iteration algorithm is a dynamic programming method used to solve for the optimal policy of a POMDP. A policy acts as a system's controller, where given a state in observable domains, or a belief in partially observable domains, and an observation, it can determine the next action. Calculating the optimal policy to a POMDP can be done off-line before execution. After it is computed, when a system begins executing the policy, it can do so with little effort as all optimal values have already been calculated. However, as mentioned above, this can be computationally expensive due to the size of the state space and time horizon. Therefore, approximation methods have been applied to POMDPs to enable online planning. Online planning works by interleaving planning and execution. The planning phase consists of generating a belief tree which represents all possible actions, observations, and successor belief states to some tree depth D . Values are generated for each node using approximations of the optimal value function, and near optimal actions are selected based off of these values. Generally, the belief tree is thrown away after each action is executed, and calculated again after an observation has been made. While this method provides a more

feasible way to select actions in partially observable domains, it can still be computationally expensive.²³ Below are two methods of approximating online planning algorithms.

Partially observable Monte-Carlo planning (POMCP) works by representing the belief nodes in the belief tree as samples, or states, of the belief probability distribution. N samples are taken from the initial belief, and m simulations are run before selecting the next near-optimal action. A generative model, g , of the POMDP is required, where $(s', o, r) = g(s, a)$. Meaning, given a state, s , and an action, a , it returns a successor state, s' , an observation, o , and an immediate reward, r . Each node in the belief tree stores the average expected value from all simulations, as well as the number of times that node has been visited. At the beginning of the simulation a starting state is selected from the initial belief state samples, and an action is selected based on the action nodes' expected values as well as their visitation counts (to encourage exploring new areas of the tree). The generative model is called with this sampled state and selected action and the immediate reward is returned. To calculate the total expected reward of this successor state, the delayed reward, in addition to the immediate reward, is needed. The delayed reward is the reward received from the successor state to the goal state. If the successor state exists within the belief tree, then the simulation continues recursively. If a successor node is not within the tree, then a random rollout policy is used to calculate the delayed reward. Once this simulation has been completed m times, an action node from the root belief node with the highest value is selected. Since the belief nodes are represented as particles, a particle filter is used to perform the belief space update.²⁴

This algorithm deals with the curse of dimensionality by representing beliefs in the belief tree as a sample of states, or particles, from the belief state. It handles the curse of history by sampling histories using the generative model. By using this model, the complexity of the algorithm depends on the complexity of the generative model, and not the underlying POMDP.²⁴

Another method, referred to as Determinized Sparse Partially Observable Tree (DESPOT), also represents a belief state as a sample of particles from the belief state distribution. The POMCP algorithm can suffer from overfitting. To compensate for this, a DESPOT is a sparse belief tree, generated by K sampled scenarios. The resulting tree contains all possible actions, but only observations from the sampled scenarios. This greatly reduces the complexity of the search tree, while still maintaining a good approximation of the entire belief tree. Once the tree is generated, dynamic programming is used to calculate the optimal policy within the tree. Heuristics and branch-and-bound are also used during the tree construction to ensure that only promising branches are included, and that overfitting does not occur.²⁵

B. BHPN Description

The following BHPN description is based on work by Kaelbling and Lozano-Pérez.⁸ Fluents, or logical descriptors, are used to define the belief state. They give information such as the location of objects, or even the contents of a location. The initial belief state, as well as the goal state, are described in terms of fluents. Operators manipulate these fluents in order to achieve the goal state. Operators consist of a primitive action, preconditions, and an effect. A primitive action is the component of an operator which is executed in the world, and all preconditions must be satisfied in order for this to take place. The effect is a fluent which should become True after the primitive action is executed. Another critical component of operators are generator functions, which are used to tie geometric constraints (e.g. ensure that there is a clear path) together with task-oriented planning.

BHPN develops plans through goal regression, or pre-image backchaining. Plans consist of an ordered set of (operator, pre-image) tuples. Pre-images are sets of fluents such that for an operator to be executed, its pre-image must be satisfied in the current world state. When planning hierarchically, plans are recursively generated, as opposed to flat planning where only one plan is generated. In hierarchical planning, first, an initial abstract plan is created, then the first operator of this plan is expanded into another, more concrete, plan. This process is called plan refinement,²⁶ and it effectively adds additional operators between the ones already in place in the higher level abstract plan. The refining process can be thought of as a tree generation in a depth-first manner (see Figure 9). Abstract plans are developed due to preconditions having designated abstraction levels. These levels indicate when in the planning process certain preconditions will be considered. For example, the first abstract plan is generated by only considering preconditions at the 0th level. When assigning preconditions levels, it is imperative to put the most critical preconditions first.⁹ As the tree is built, eventually an operator will have all of its preconditions satisfied, at which point the primitive is executed. It is this functionality which allows us to “plan in the now”, or perform interleaved planning and execution. This process will be explained in detail in Section V.

1. BHPN Components

Pre-mission, the belief state must be initialized based on the UAV’s current domain knowledge. The belief state represents the location of objects in the world as a probability distribution over all states. These distributions are updated as actions are performed, and fluents use this information to determine whether or not a fluent is satisfied in the current belief state. In general, fluents can be characterized in this way: “the *threat* location is *region 1* with certainty *0.5*”, or even like this: “*region 2* is occupied by a target with certainty *0.8*.” The flexibility available when defining fluent types enables us to incorporate varying degrees of knowledge in many different ways. Operators are also user-defined in terms of domain knowledge. Below is an example of an operator.

Operator: Fly(UAV, start, dest, path, ϵ)

Preconditions:

0: the UAV is located at the start location with probability $1 - \text{flyRegress}(\epsilon)$

1: the path is clear with probability p_{clear}

Generators: path = *generatePath*(start, dest)

Effect: the UAV is located at the dest location with probability $1 - \epsilon$

Primitive Action: Fly from the start to the dest location

Figure 1. Example of an operator in the UAV domain.

As you can see, the preconditions and effect are fluents. In belief space, fluents are in terms of probabilities since states are only partially observable. An example of a goal would be “the UAV is located at *region 1* with *0.95* certainty.” Planning uses goal regression, so in order to achieve the goal, we must determine the necessary pre-image. This is where regression functions, such as *flyRegress*(ϵ) above, come into play. We need to calculate the necessary probability currently in the world in order to successfully Fly the UAV from its start location to *region 1*. The regression function is derived from the underlying transition model of the UAV. If the UAV’s actions are highly nondeterministic, then we will need a higher initial certainty before executing the Fly action. But, if our actions are nearly deterministic, then we can easily ensure that after the Fly task is executed, we are certain (greater than 0.95) that the UAV will be located in *region 1*. Regression functions are also important when an operator involves observations, which we will see in Section III. In this case the regression functions are derived from the underlying UAV observation model.

Another important component of operators are generator functions. These can be used to incorporate geometric constraints. In Figure 1 a generator is used to find a path between the start and dest locations. Then, this path becomes part of a precondition such that it must be clear with p_{clear} (user-defined) certainty. Once these two preconditions are satisfied, the UAV will attempt to fly from start to dest.

The final component of the BHPN process is the belief space update. When a primitive action is executed, the UAV must update its belief state based on its own motion or observations it has gathered. This step depends on how the belief space is represented. Methods such as Kalman filtering for Gaussian processes and Bayes filtering for non-Gaussian processes may be implemented.

III. UAV Domain

In this section we will discuss our application of BHPN to the UAV domain. We use a realistic UAV observation model which is crucial to the BHPN algorithm. When the algorithm is choosing operators it uses the regression functions, which are dependent on the observation model, to determine how successful actions should be. Therefore, the more realistic the observation model is, the more realistic and achievable the plans developed by BHPN will be. The observation model is also used in a Bayesian filter belief space update. First, in Section A we will discuss the operators in this domain. Then, in Section B we will give the regression function used in the operators. The observation model is described in Section C. And finally, Section D gives the planning tree developed by our BHPN implementation for a given mission objective.

A. Operators

Figure 2 gives the operators for our UAV domain. They consist of the UAV actions required to achieve our particular mission objective. The mission objective is to localize a target to a certainty level of 0.95. In our mission environment, there are partially observable ground troops and targets. Our initial belief about

their locations is limited, thus we need an Observe operator which gives us (noisy) location information regarding the troops and targets. The UAV itself is fully observable with deterministic actions in our example. Therefore, the Fly operator does not have a probability associated with the UAV locations. Lastly, the GetInfo operator receives information from ground troops regarding object locations.

Operator: Fly(UAV, start, dest, path)

Preconditions:

0: the UAV is located at the start location

1: the path is clear with probability 0.9

Generators: path = *genPath*(start, dest)

Effect: the UAV is located at the dest location

Primitive Action: Fly from the start to the dest location

Operator: Observe(object, loc, UAV, ϵ)

Preconditions:

0: the object is at loc with probability $1 - regress(\epsilon)$

1: $1 - regress(\epsilon) > 0.1$

2: the UAV is at the loc

Generators: loc = *genLikelyLoc*(object)

Effect: the object is at loc with probability $1 - \epsilon$

Primitive Action: look for the object at loc

Operator: GetInfo(object, groundTroops, loc, UAV, ϵ)

Preconditions:

0: the location of the groundTroops is known with probability > 0.95

1: $1 - regress(\epsilon) < 0.1$

2: the object's location is known with probability $1 - regress(\epsilon)$

3: the UAV is located at loc

Generators: loc = *genLikelyLoc*(groundTroops)

Effect: the object is at loc with probability $1 - \epsilon$

Primitive Action: look for the object at loc

Figure 2. The operators for the UAV domain. The preconditions must be met in order for the primitive action to be executed. The preconditions are ranked by their abstraction levels. The generators deal with the geometric space (*genPath*), as well as the belief space (*genLikelyLoc*).

As mentioned before, the Fly operator is a deterministic action which transits the UAV from one location to another. The generated path must be clear with a certainty of 0.9 before the UAV can execute the primitive action of flying. The most critical preconditions should come first when determining abstraction levels. This is why the first precondition for the Fly operator is that the UAV be at the start location. If this condition is not met, then the action will never be successfully executed for the generated path. However, the clear path requirement is less critical due to the flexibility involved in path planning. There are methods of increasing the certainty that the path is clear, either by prosecuting threats or by taking observations to verify that the path is in fact not threatening.

In our domain there are two methods of increasing knowledge about an object's location. The UAV can either directly observe the object, or it can receive information from the ground troops regarding an object's location. However, the act of communicating with ground troops should only be taken as a last resort. To perform this task the UAV must fly to the ground troops, establish communication, and receive information regarding an object's location. While this is a good alternative method for gaining information, it is only valid when very little is known about an object's location, and only results in little knowledge gain. To increase certainty past a certain threshold, the UAV must directly observe an object.

The Observe operator allows the UAV to gather information through direct sensing. Since an object's location is represented as a distribution over states, the UAV must use a generator function (*genLikelyLoc*) and the underlying belief space to determine the best location to observe the threat. The first precondition

involves calculating the minimum required probability of the object's location, $1 - regress(\epsilon)$, with regards to the effect certainty, $1 - \epsilon$. If this precondition fails, then the observe operator will not be guaranteed to achieve its goal. The second precondition says that the minimum required probability to execute this task must be greater than 0.1. This is what ensures that we only perform direct sensing if we are confident we will see the object. Otherwise, we will communicate with the ground troops. Lastly, the UAV must be located at the likely location of the object. This is the easiest precondition to satisfy, and therefore the last of the preconditions.

Finally, the GetInfo operator allows us to "indirectly sense" an object. First, we must know the location of the ground troops with at least 0.95. If this precondition is not met, then we will need to look for the ground troops using the Observe operator previously described. The second precondition is that we know about the object's location with less than 0.1 certainty. This ensures that we only resort to this method of "indirect sensing" when very little in know about the given object. Since the effect of the operator is an increase in knowledge about the object's location, we must again use the *regress* function to determine the minimum threshold for executing this action and achieving our goal. Lastly, the UAV must be located where the ground troops most likely are.

B. Regression Function

Regression functions are used to calculate precondition probabilities based on effect probabilities. For example, if a UAV wants to know the location of a target with 0.9 certainty, how certain must it *currently* be about the target's position? These functions depend on the UAV's sensor capabilities. If the sensor has little noise, then the UAV can be confident that it will achieve the goal of 0.9 certainty. However, if the UAV has a very noisy sensor, it will need a higher certainty as a precondition to execute the observe operator. The regression functions are best-case, and are computed under the assumption that we are directly overhead the target we are observing and we do not receive a false positive. Below is the regress function where p_{cp} is the probability of a correct positive, and p_{fp} is the probability of a false positive. It is derived from the UAV's observation model, which we will discuss in Section C. The derivation for this function is outlined in the Appendix.

$$regress(\epsilon) = \frac{p_{cp}\epsilon}{p_{cp}\epsilon + p_{fp}(1 - \epsilon)}$$

C. Observation Model

In UAV missions, observations are acquired through UAV camera data using object recognition algorithms. Our research focuses on planning and not object recognition, therefore in our simulated environment camera data is generated. The generated data is modeled with Gaussian noise and no data association errors. In other words, the observations consist of the object being identified with no error, and the location of that object with Gaussian noise. There is a chance of either incorrectly missing an object which is visible (false negative), or seeing an object when it is not visible (false positive). There is a visibility radius around the UAV so that if an object is detected, it must be within this radius.

The observations, \mathbf{z}_k , are functions of the underlying object states, o_k . The range of the UAV's camera visibility is characterized by a radius, r , and the state of the UAV is \mathbf{x}_k . We designate $R(\mathbf{x}_k)$ to be the set of states within r of the UAV at time step k .

$$R(\mathbf{x}_k) := \{s \mid \|\mathbf{x}_k - s\| < r\} \text{ where } s \in \mathbb{R}^2$$

$$\mathbf{z}_k \in \{(x_k, y_k)\} \text{ where } (x_k, y_k) \in R(\mathbf{x}_k)$$

$P(\mathbf{O}_k)$ is the probability distribution of an object's location over all possible states. For example, if an object's state is known with absolute certainty, $P(\mathbf{O}_k = o_k) = 1$. For the remainder of this section the k subscript is dropped for ease of reading.

The following formalization of the observation model is based on Dille.²⁷ If an object is in $R(\mathbf{x})$, there are three possible observations: the object is detected and the observation is correct, the object is detected but the observation is incorrect, or the object is not detected. We will refer to an incorrect object position as o_{incorr} . If the object is not in $R(\mathbf{x})$, then there is a chance of a false positive occurring. The table below outlines all of these possibilities and the probabilities associated with each of them.

	$o \in R(\mathbf{x})$	$o \notin R(\mathbf{x})$
$\mathbf{z} = \emptyset$	$P(\mathbf{z} = \emptyset o) = 1 - p_{cp}$	$P(\mathbf{z} = \emptyset o) = 1 - p_{fp}$
$\mathbf{z} \in R(\mathbf{x})$	$P(\mathbf{z} = o o) = p_{cp}[\mathcal{N}(\mathbf{o} \mathbf{x}, \sigma^2)]$	$P(\mathbf{z} = o_{incorr} o) = p_{fp}$

The variables p_{cp} and p_{fp} are the probabilities of a correct positive and a false positive, respectively. Within the UAV's visibility radius, observations are Gaussian. σ is the variance which characterizes the performance of the object recognition algorithm. The greater it is, the better the UAV can detect objects that are far away, and vice versa. During plan execution, the observations and the observation model above are used to update object belief states through Bayes filtering.

D. Mission Example

To display the effectiveness of the simulated mission planner we have developed, Figure 3 gives a hierarchical plan decomposition. The mission objective is to localize a specific target to a certainty of 0.95. The UAV has very little initial belief about the location of the target, but a high certainty of where the ally ground troops are located. It gets target location information from the ground troops, then confirms the target location with direct sensing. The blue nodes represent goals and subgoals generated through plan refinement. The pink nodes represent the plans themselves, and the green nodes in the figure represent the primitive actions. The numbers before each operator in the plan (pink) nodes are the respective operator abstraction levels. As the tree is constructed, when a green node is reached, the primitive action is executed, and planning does not continue until the action is complete.

IV. Realistic Simulation

Our implementation of the BHPN algorithm is wrapped in a ROS (robotic operating system) node, and Gazebo is used to render plan execution. There is one quadrotor executing the plans developed by the BHPN algorithm. Several ROS nodes are running simultaneously. The main one is the BHPN node. It is initialized with a belief of the world. The world contains ground troops and a target all in the ground plane. The BHPN node also contains all operators used to develop plans. As planning occurs, when a primitive action is required to be executed, a ROS service is called to execute the action. Once the action is completed, the BHPN node uses the observation to update its belief of the world. It then verifies if the action generated the expected result. If it did not then replanning occurs. The observations are simulated based on the observation model given in Section C and the underlying world state. Figure 4 outlines the framework for the BHPN realistic simulation.

Figures 5-8 give screenshots of the mission. Each screenshot corresponds to a primitive action being executed (the green nodes in Figure 3). The left side of the figures show the simulation environment as rendered by Gazebo. The ground troops are represented by several Turtlebots (the small white robots on the left). The target is the Polaris ground vehicle on the right. The UAV is represented by the quadrotor. The camera view from the quadrotor is shown in the center of the figure. On the right is the RViz environment. Paths are shown in green, poses are represented by a red arrow. The yellow and black map is the global costmap used for path planning. It was generated at a fixed height of 10 m.

The given mission scenario is just one of many possible missions that could be executed by the same BHPN framework. Once all operators are defined and the initial belief state is represented, BHPN is able to generate plans to achieve many different types of goals.

V. Algorithm Evaluation

A. Task Planning

Hierarchical planning is able to improve planning efficiency through the use of abstraction. By giving preconditions abstraction levels, we are actually turning one operator into many operators. For example, the Fly operator is actually a Fly0 operator (only precondition 0 is considered), a Fly1 operator (both preconditions are considered), and a Fly primitive action. Figure 9 shows the planning problem represented as a tree. It is constructed in a depth first manner. The nodes are abstract operators or primitive actions. Plans of length k are recursively generated, each time planning for an operator at its next abstraction level.

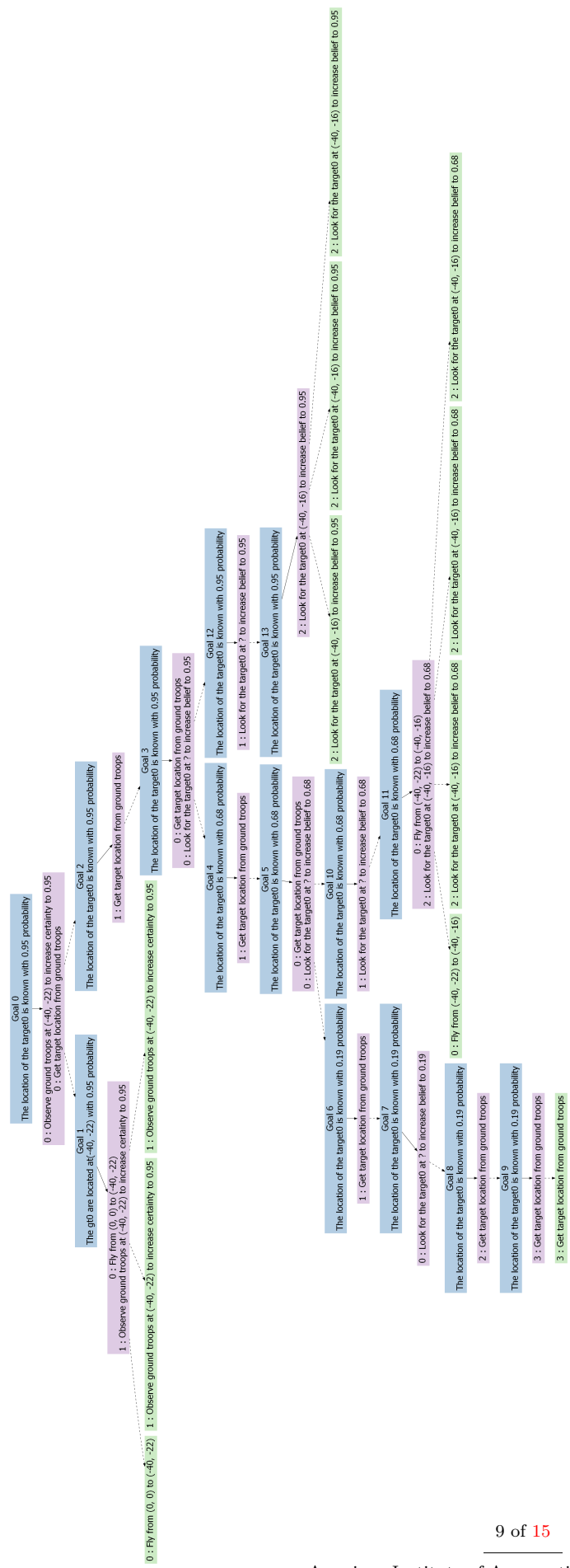


Figure 3. This is the planning tree constructed while accomplishing the overall goal of knowing the target location with 0.95 certainty (Goal 0).

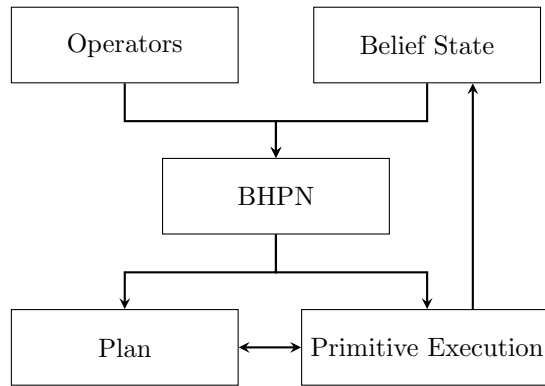


Figure 4. BHPN Flowchart

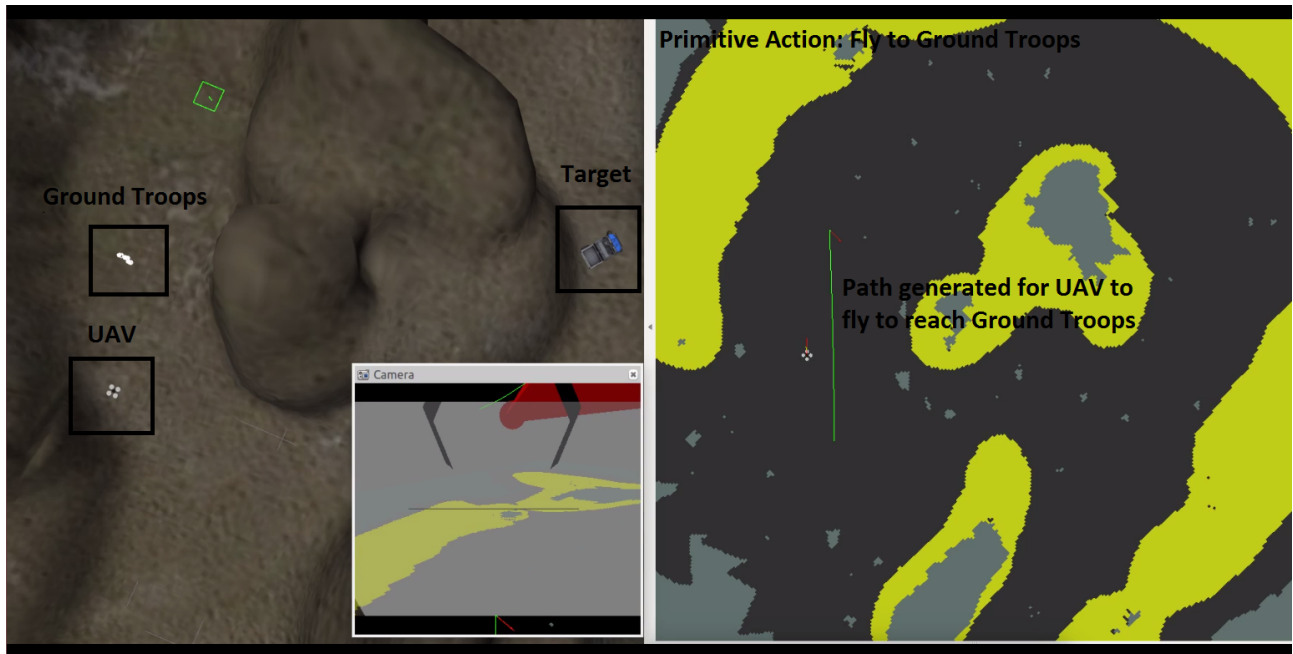


Figure 5. The first primitive action to be returned by the BHPN algorithm is to fly to the ground troops. In our simulation they are represented by several Turtlebots (the small white robots seen in the figure). The green path is the global path returned by the path planner in the Fly primitive action.

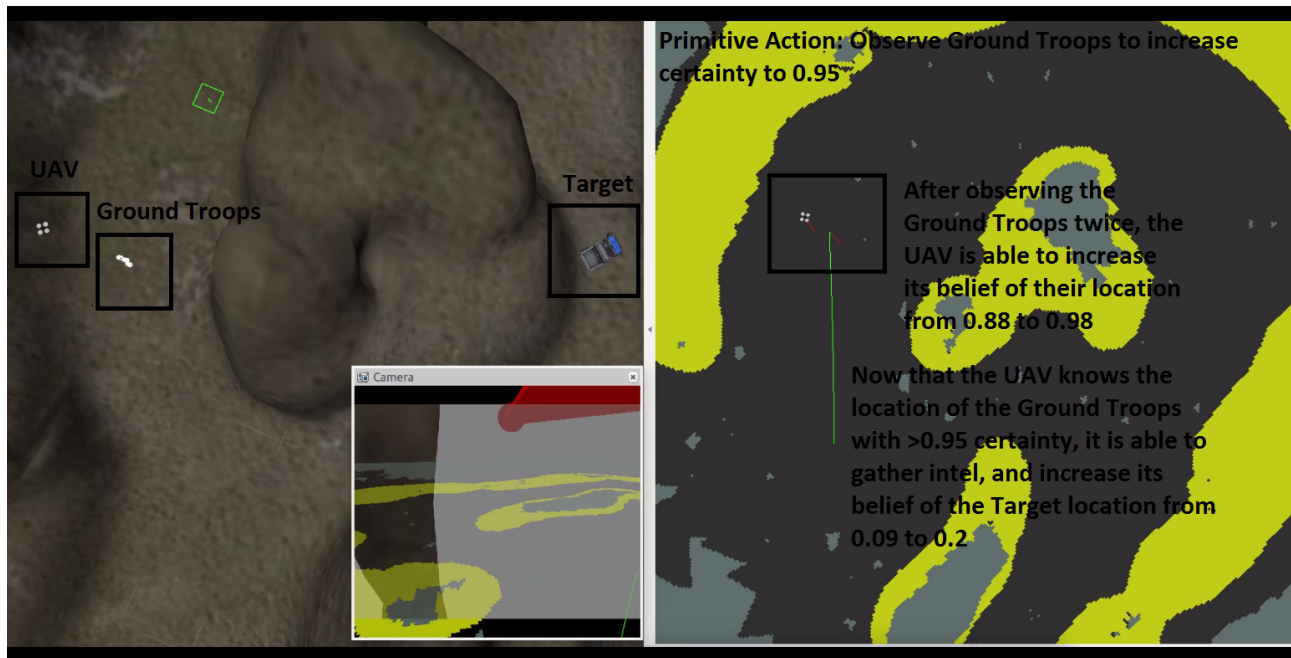


Figure 6. The next primitive action is to observe the ground troops in order to verify that they are actually where the UAV thought that they would be. This primitive must be performed twice because initially the quadrotor does not see the ground troops. Replanning occurs, and it observes again, this time confirming that the ground troops are there. This corresponds to the second and third green nodes in Figure 3. Then, the UAV gets intel from the ground troops regarding the location of the target. This corresponds to the fourth green node in Figure 3.



Figure 7. Now the UAV has updated its belief of where it thinks the target is located, and it must go increase this belief further by directly observing the target. The green path shown on the right is the path it generates and follows to reach the target.



Figure 8. Finally, the UAV observes the target several times to verify that it is in the believed location. The reason so many observations are made (as you can see in Figure 3) is due to the noise in the sensor.

The highlighted nodes represent the same operator at different abstraction levels. When the left-most node in the tree is a primitive action, it is immediately executed. Then, planning proceeds until all of the leaf nodes of the tree are primitive actions which have been executed. Each operator is associated with a pre-image (not shown in Figure 9). Pre-images are sets of fluents which represent what the belief state must look like in order to execute the operator with all of its preconditions met.

Hierarchical planning is efficient because it develops multiple plans of length k , which are shorter in length than the final plan, l , they are contributing to. The length of the final plan is the number of leaf nodes in the planning tree. Flat planning is the method of planning without the use of abstraction. In flat planning there is no plan refinement and all preconditions are considered simultaneously. This problem is much more constrained than the hierarchical problem, and the solution complexity grows exponentially with the overall plan length.

The complexity of hierarchical planning depends on the planning tree depth, n , the number of operators added per subgoal, k , and the total number of possible operators, b . Assume the tree is regular, or all nodes have k children. Each level, i , of the tree represents an ordered plan of length k^i . Level n represents an executable plan of length $l = k^n$. The work done in expanding each node is b^k . Therefore, to calculate the overall complexity of the planning tree we must sum up the work for each level: $\sum_{i=1}^n k^i b^k$, which gives $O(lb^k)$ or $O(b^k)$. If flat planning had been used, then the complexity would be $O(b^l)$ as opposed to $O(b^k)$. This shows the savings generated by hierarchical planning, since $k = l^{1/n} \ll l$.

This analysis assumes that all plans are refineable and no backtracking is necessary, all actions are permanent and there is no replanning, and all plans serialize or no subgoals interact. These assumptions may not hold if, in a dynamic world, not all actions are permanent. For example, if a target is dynamic, then it may be necessary to localize it several times throughout a mission to ensure that we still know where

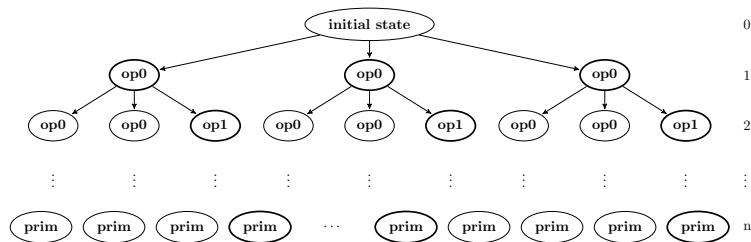


Figure 9. Planning tree with $k = 3$, where op_x indicates an operator at abstraction level x and $prim$ represents a primitive action. There are n levels in the tree.

it is. In this case actions are not permanent. Also, subgoals may interact if the action of one undoes a precondition of the other.

B. Motion Planning Under Uncertainty

The analysis above only accounts for the task planning component of the overall BHPN algorithm. The motion planning complexity is dependent on the motion planning algorithm in place. For our simulation we implemented the navigation ROS stack, which uses A* as the global motion planner. To evaluate a *combined* task and motion planner is difficult due to the domain-dependent operators. To give a baseline of comparison we will give the complexity of solving a POMDP, and show that BHPN scales better in terms of the size of the state space and the planning time horizon.

Solving a POMDP optimally can be done for some finite horizon problems with a small state space, using the optimal value iteration algorithm. This value function can be represented by a set of $|S|$ -dimensional hyperplanes, Γ_t , for any horizon t .²³ They describe a value function over the belief space for a specific action. The best value is then calculated based on these value functions. The complexity of this algorithm is $O(|A||Z||S|^2|\Gamma_{t-1}|^{|Z|})$, where A is the size of the action space, Z is the size of the observation space, and S is the size of the state space.

Standard online planning algorithms require building a belief tree of depth D . Evaluating a tree of all reachable beliefs to depth D requires $O((|A||Z|)^D|S|^2)$. This has a few drawbacks: it is exponential in D , and there typically is not enough time at each planning step to fully build and evaluate the entire belief tree. Methods like POMCP and DESPOT efficiently handle these drawbacks.

The complexity of task planning using BHPN is $O(b^k)$ where b is the number of available operators, and k is the number of actions added each time the BHPN tree is refined. Adding certain actions requires calling a path planner. Since the belief space is discretized for planning efforts, the path planner is not required to work in the space of probability distributions, and it assumes that actions are deterministic. Then, as paths are followed during execution, deviations from this assumption are monitored and replanned for. This method is efficient as it does not require planning in belief space, however domain knowledge is required to plan efficiently using BHPN.

Methods for optimal motion planning are intractable largely due to the long time horizon. BHPN breaks the motion planning problem into multiple motion planning problems. By employing this method, we can utilize strong path planning algorithms for successive short duration flights, or “plan in the now.”

VI. Conclusion

In mission environments with uncertainty, it can be difficult for a tactical UAV to reason about how to best accomplish a mission objective. Hierarchical planning in the now in belief space provides a method of combining the task planning element of mission planning with the geometric reasoning required to plan in a partially observable environment. Domain specific operators must be constructed in order to implement this algorithm. However, work has been done on enabling a system to learn these tasks and how to construct them independently.²⁸ The resulting planning framework is flexible in that it is guaranteed to develop a plan if a feasible plan exists in the current belief state. We successfully reapplied the BHPN algorithm to an example mission environment, and gave a sample mission scenario in which a UAV must detect a target with the help of ally ground troop intel. As we showed, it is able to replan when actions do not have intended results, all the while performing interleaved planning and execution. An analysis on the BHPN complexity was also given, as well as a comparison to solving a POMDP which is the traditional method of solving belief space problems.

Acknowledgements

This research was supported by the Charles Stark Draper Laboratory and Northeastern University. We would also like to thank Leslie Kaelbling for her help and guidance in developing our implementation of BHPN.

References

- ¹Cambone, S. A., *Unmanned aircraft systems roadmap 2005-2030*, Defense Technical Information Center, 2005.
- ²Erol, K., Hendler, J., and Nau, D. S., “HTN planning: Complexity and expressivity,” *12th AAAI Conference on Artificial Intelligence*, Vol. 94, 1994, pp. 1123–1128.
- ³Miller, S. A., Harris, Z. A., and Chong, E. K., “A POMDP framework for coordinated guidance of autonomous UAVs for multitarget tracking,” *EURASIP Journal on Advances in Signal Processing*, Vol. 2009, 2009, pp. 2.
- ⁴Bourgault, F., Furukawa, T., and Durrant-Whyte, H. F., “Coordinated decentralized search for a lost target in a Bayesian world,” *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, Vol. 1, IEEE, 2003, pp. 48–53.
- ⁵Kaelbling, L. P. and Lozano-Pérez, T., “Hierarchical Task and Motion Planning in the Now,” *IEEE Conference on Robotics and Automation (ICRA)*, 2011, Finalist, Best Manipulation Paper Award.
- ⁶Cambon, S., Alami, R., and Gravot, F., “A hybrid approach to intricate motion, manipulation and task planning,” *The International Journal of Robotics Research*, Vol. 28, No. 1, 2009, pp. 104–126.
- ⁷Wolfe, J., Marthi, B., and Russell, S. J., “Combined Task and Motion Planning for Mobile Manipulation.” *ICAPS*, 2010, pp. 254–258.
- ⁸Kaelbling, L. P. and Lozano-Pérez, T., “Integrated Task and Motion Planning in Belief Space,” *International Journal of Robotics Research*, Vol. 32, No. 9-10, 2013.
- ⁹Sacerdoti, E. D., “Planning in a Hierarchy of Abstraction Spaces,” Tech. Rep. 78, AI Center, SRI International, 333 Ravenswood Ave, Menlo Park, CA 94025, June 1973.
- ¹⁰Fikes, R. E. and Nilsson, N. J., “STRIPS: A new approach to the application of theorem proving to problem solving,” *Artificial intelligence*, Vol. 2, No. 3, 1972, pp. 189–208.
- ¹¹Barrett, A. and Weld, D. S., “Partial-order planning: Evaluating possible efficiency gains,” *Artificial Intelligence*, Vol. 67, No. 1, 1994, pp. 71–112.
- ¹²Bylander, T., “The computational complexity of propositional STRIPS planning,” *Artificial Intelligence*, Vol. 69, No. 1, 1994, pp. 165–204.
- ¹³Ponzonei Carvalho Chanel, C., Teichteil-Königsbuch, F., and Lesire, C., “POMDP-based online target detection and recognition for autonomous UAVs,” *The 20th European Conference on Artificial Intelligence*, 2012.
- ¹⁴Chanel, C. P. C., Teichteil-Königsbuch, F., and Lesire, C., “Multi-Target Detection and Recognition by UAVs Using Online POMDPs.” *27th AAAI Conference on Artificial Intelligence*, 2013.
- ¹⁵Hauskrecht, M., “Value-function approximations for partially observable Markov decision processes,” *Journal of Artificial Intelligence Research*, 2000, pp. 33–94.
- ¹⁶Murphy, K. P., “A survey of POMDP solution techniques,” *environment*, Vol. 2, 2000, pp. X3.
- ¹⁷Geyer, C., “Active target search from UAVs in urban environments,” *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, IEEE, 2008, pp. 2366–2371.
- ¹⁸Carpin, S., Burch, D., and Chung, T. H., “Searching for multiple targets using probabilistic quadrees,” *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, IEEE, 2011, pp. 4536–4543.
- ¹⁹Beetz, M., Klank, U., Kresse, I., Maldonado, A., Mosenlechner, L., Pangercic, D., Ruhr, T., and Tenorth, M., “Robotic roommates making pancakes,” *Humanoid Robots (Humanoids), 2011 11th IEEE-RAS International Conference on*, IEEE, 2011, pp. 529–536.
- ²⁰Choi, J. and Amir, E., “Combining planning and motion planning,” *Robotics and Automation, 2009. ICRA’09. IEEE International Conference on*, IEEE, 2009, pp. 238–244.
- ²¹Guitton, J. and Farges, J.-L., “Taking into account geometric constraints for task-oriented motion planning,” *Proc. Bridging the gap Between Task And Motion Planning, BTAMP*, Vol. 9, 2009, pp. 26–33.
- ²²Hauser, K., “Task planning with continuous actions and nondeterministic motion planning queries,” *Proc. of AAAI Workshop on Bridging the Gap between Task and Motion Planning*, 2010.
- ²³Ross, S., Pineau, J., Paquet, S., and Chaib-draa, B., “Online planning algorithms for POMDPs,” *Journal of Artificial Intelligence Research*, 2008.
- ²⁴Silver, D. and Veness, J., “Monte-Carlo Planning in Large POMDPs,” *Advances in Neural Information Processing Systems 23*, edited by J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, Curran Associates, Inc., 2010, pp. 2164–2172.
- ²⁵Somani, A., Ye, N., Hsu, D., and Lee, W. S., “DESPOT: Online POMDP Planning with Regularization,” *Advances in Neural Information Processing Systems 26*, edited by C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Curran Associates, Inc., 2013, pp. 1772–1780.
- ²⁶Bacchus, F. and Yang, Q., “Downward refinement and the efficiency of hierarchical problem solving,” *Artificial Intelligence*, Vol. 71, No. 1, 1994, pp. 43–100.
- ²⁷Dille, M., “Search and pursuit with unmanned aerial vehicles in road networks,” Tech. rep., DTIC Document, 2013.
- ²⁸Niekum, S., Osentoski, S., Konidaris, G., and Barto, A., “Learning and Generalization of Complex Tasks from Unstructured Demonstrations,” *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, October 2012.

Appendix

The following derivation for the regression function comes from work by Kaelbling and Lozano-Pérez.⁸ When a UAV performs the Observe primitive action, it is attempting to increase its belief of an object’s

location to some certainty, $1 - \epsilon$. To determine the correct precondition, it must use a regression function based on the observation model. More specifically, the probability of a false positive observation, p_{fp} , and the probability of a correct positive observation, p_{cp} . When calculating $regress(\epsilon)$, we assume that the UAV is directly above the object being observed, and that it will see the object. This is a best case scenario, and assumes that the UAV's belief of the object's location will increase after the primitive action is performed. If this assumption does not hold, and the belief actually decreases, then BHPN will simply replan, and most likely add more Observe operators to attempt to increase the belief again.

$$regress(\epsilon) = \frac{p_{cp}\epsilon}{p_{cp}\epsilon + p_{fp}(1 - \epsilon)}$$

Before the Observe action occurs, the object being observed is in location l with some probability $1 - \epsilon_r$, such that $1 - \epsilon_r = Pr_b(L = l)$. After the action is executed, we want to increase the belief to $1 - \epsilon = Pr_{b'}(L = l)$.

$$1 - \epsilon = Pr_{b'}(L = l | A = Observe(l), O = true)$$

$$1 - \epsilon = \frac{Pr(O = true | L = l, A = Observe(l)) Pr_b(L = l)}{Pr(O = true | A = Observe(l))}$$

$$1 - \epsilon = \frac{p_{cp}(1 - \epsilon_r)}{p_{cp}(1 - \epsilon_r) + p_{fp}\epsilon_r}$$

Then, solving for ϵ_r we get:

$$\epsilon_r = regress(\epsilon) = \frac{p_{cp}\epsilon}{p_{cp}\epsilon + p_{fp}(1 - \epsilon)}$$