MULTI-AGENT UAV PLANNING USING BELIEF SPACE HIERARCHICAL PLANNING IN THE NOW

Caris Moses

A thesis submitted for the degree of Master of Science

December 2015

Contents

1 Introduction

Planning long duration missions for unmanned aerial vehicles (UAVs) in dynamic environments has proven to be a very challenging problem. Tactical UAVs must be able to reason about how to best accomplish mission objectives in the face of evolving mission conditions. Examples of UAV missions consist of executing multiple tasks such as: locating, identifying, and prosecuting targets; avoiding dynamic (i.e. pop-up) threats; geometric path planning with kinematic and dynamic constraints; and/or acting as a communication relay [7]. 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 objects within that environment. The world state is also only partially observable due to sensor noise, and requires us to plan in the belief space, which is a probability distribution over all possible states. Some *a priori* contextual knowledge, like target and threat locations, is available via satellite imagery based maps. But it is possible this will be "old" data by execution time. This makes classic approaches to *a priori* task, or symbolic, planning a poor choice of tool. In addition, task planners traditionally do not have methods for handling geometric planning problems as they focus on high level tasks. However, modern belief space geometric planning tools become intractable for large state spaces, such as ours.

Simply developing a set of ordered tasks, as those produced by purely symbolic approaches like hierarchical task networks [12], 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[21, 4], with long planning horizons. However, recent tools in the domain of robotic manipulation have approached this problem by combining symbolic and geometric planning paradigms [18, 6, 33]. One in particular, Hierarchical Planning-in-the-Now in belief space [19] (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 [29]. 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.

In addition to all of the complexities associated with UAV mission planning discussed above, it is also common for multiple UAVs to work as a team to accomplish a mission objective. This is due to the fact that some vehicles may have certain sensor capabilities that others lack. Or it could also simply be to spread out and achieve sufficient coverage of an environment. In general, the complexity of centralized multi-agent planning (MAP) is exponential in the number of agents. Therefore, a decentralized planning approach is taken to enable UAV teaming. The developed framework uses BHPN as a method of implementing this loosely-coupled MAP effort. BHPN uses preconditions to bind objects in the physical world to executable actions. This method is used in MAP to select the correct UAV to

execute an action based on its sensor capabilities. As with single-agent planning, MAP continues to develop future plans assuming that the help it receives will be successful, and replans when this assumption is not held.

2 Background

2.1 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 on a low level. This section addresses task planning methods in Section 2.1.1, and motion planning methods in Section 2.1.2. Methods other than BHPN which can handle the integration of the two are discussed in Section 2.1.3. A more detailed discussion about belief space planning, specifically partially observable Markov decision processes (POMDPs) is given in section 2.1.4. Finally, other methods of MAP are given in section 2.1.5.

2.1.1 Task Planners

Task planning consists of searching for an action sequence which will transform the current world state into a goal state. STRIPS [13] (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 planning [2] which loosens restrictions on the order in which actions must be executed. Hierarchal methods, such as HTN [12] (hierarchical task networks), provide significant speedups when compared to classical planning approaches [5]. 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, a low-level motion planner is required.

2.1.2 Motion Planners

The field of motion planning is rich with proposed methods, particularly where they apply to uncertain domains. In the UAV domain the motion planning problem is often to localize and track targets [21]. One method is to solve a partially observable Markov decision process (POMDP) [27, 9]. However, solving a POMDP is generally computationally intractable [17]. A common approximation is to use receding horizon control as opposed to solving for an infinite time horizon [22]. 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 [21].

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, 14]. 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 [8].

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, strong path planning algorithms can be utilized for successive short duration flights, or "planning in the now".

2.1.3 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 tasks which have been experimented with include cooking a meal [3], pushing buttons to dial a number [10], and performing other household tasks [18]. 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 still holds [6]. Another method speeds up the motion planning process by caching previously calculated motions and using them in future planning problems [33]. Guitton and Fargus give an approach for translating geometric constraints into mathematical penalty functions and uses non-linear programming to solve them in the context of HTN planning [15]. Lastly, Hauser proposes a probabilistic tree-of-roadmaps planner which samples the task and motion space to develop feasible plans [16].

2.1.4 POMDPs

In partially observable and non-deterministic domains a common method of choosing optimal actions is by solving 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 [28].

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, or before execution. Therefore, when a system begins executing the policy, it can do so with little effort since all optimal values are already known. 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 depth D. Values are generated for each node using approximations of the 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 [28]. Below are two methods of approximating online planning algorithms.

POMCP (Partially Observable Monte-Carlo Planning) represents 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 $(s', o, r) = g(s, a)$. Given a state and an action, it returns a successor state, an observation, and an immediate reward. Each node in the belief tree stores an average expected value from all simulations, as well as a visitation count. 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 [30].

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 [30].

DESPOT (Determinized Sparse Partially Observable Tree) 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 [31].

2.1.5 Multi-Agent Planning

There are many different methods of performing multi-agent planning (MAP). They range from solving problems in a centralized manner to a distributed manner and from loosely coupled to tightly coupled systems. A common method is to perform constraint satisfaction, and other methods are based on a multi-agent (MA) STRIPS style algorithm. Our work most closely relates to the MA-STRIPS planning method applied to loosely coupled decentralized domains. In our environment each UAV is tasked with its own objective. Help may not always be necessary, and when it is it is generally not a highly invasive request. Therefore multiple agent's subgoals do not interact to a great degree, making our system loosely-coupled.

Nissim, Brafman, and Domshlak [25] developed a distributed MAP algorithm which utilizes constraint satisfaction as well as local planning as a means of developing consistent plans. By solving MAP problems in a distributed manner, they are able to improve planning efficiency beyond that of state-of-the-art centralized planning algorithms. Pellier [26] also used constraint satisfaction to deal with MAP and coordination. He also employs a graph-merge technique to merge individual agent plans and their interactions. Nissim and Brafman [24] developed another method of MAP which uses A* search to solve loosely-coupled problems and leads to significant speedups.

2.2 BHPN Description

The following BHPN description is based on work by Kaelbling and Lozano-Pérez [19]. Integrated task and motion planning requires a means of connecting the high-level task space with the low-level geometric space. BHPN achieves this through the use of fluents, or logical descriptors. Fluents can contain information about the continuous geometric space, or belief space, as well as discrete attributes about the environment. They give information such as the certainty that an object is in a given location, or even the contents of a location.

The initial belief state, as well as the goal state, are described in terms of fluents. Operators (see Figure 2.2.1, or tasks, 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) in 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, many plans are developed, as opposed to flat planning where only one plan is generated. 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 [1], 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 2). To determine the set of preconditions necessary to consider when developing a plan, preconditions are assigned 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. Planning will not continue if there is no means of satisfying a precondition. Therefore, when assigning a precondition a level, it is imperative to put the most critical preconditions first [29] as well as the preconditions which are the most difficult to satisfy. As the tree is built, eventually an operator will have all of its preconditions satisfied, at which point the primitive action is executed. Planning does not continue until the action is executed. It is this functionality which allows us to "plan in the now", or perform "interleaved" planning and execution. When planning, contingency plans are not generated. Plans are assumed to

Operator: Fly(UAV, start, dest, path, ϵ)
Preconditions:
0: the UAV is located at the start location with probability $flyRegress(\epsilon)$
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 ϵ
Primitive Action: Fly from the start to the dest location

Figure 1: Example of an operator in the UAV domain. p_{clear} is user-defined.

have the expected result on the belief state. This is a very loose assumption, particularly in nondeterministic and dynamic domains where the state is rapidly changing. However a major attribute of BHPN is observing the belief state after a primitive is executed and replanning if an action did not have the expected result. As future plans are simply abstract ones, not a significant amount of planning effort is lost as replanning occurs.

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.

2.2.1 Components

Pre-mission, a set of fluents must be initialized based on the UAV's current knowledge. 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 initializing the belief state enables us to incorporate varying degrees of knowledge in many different ways. Concrete examples of fluents will be given in Section 3.2. Operators are also user-defined in terms of domain knowledge. Figure 2.2.1 gives an example of an operator.

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. BHPN utilizes goal regression. In order to achieve the goal, the necessary pre-image must be determined. This is where regression functions, such as $flyRegress(\epsilon)$ above, come into play. The necessary probability currently in the world needs to be calculated 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 a higher initial certainty is needed before executing the Fly action. But, if actions are nearly deterministic, then it is easily ensured that after the Fly task is executed that the UAV will be located in *region 1* with high certainty (greater than 0.95). Regression functions are also important when an operator involves observations, which Section 3.4 will cover. 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 2.2.1 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.

2.2.2 Algorithm

Hierarchical planning is able to improve planning efficiency through the use of abstraction. Giving preconditions abstraction levels actually turns 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 (where all

preconditions are considered). Figure 2 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. 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 Fig. 2). 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.

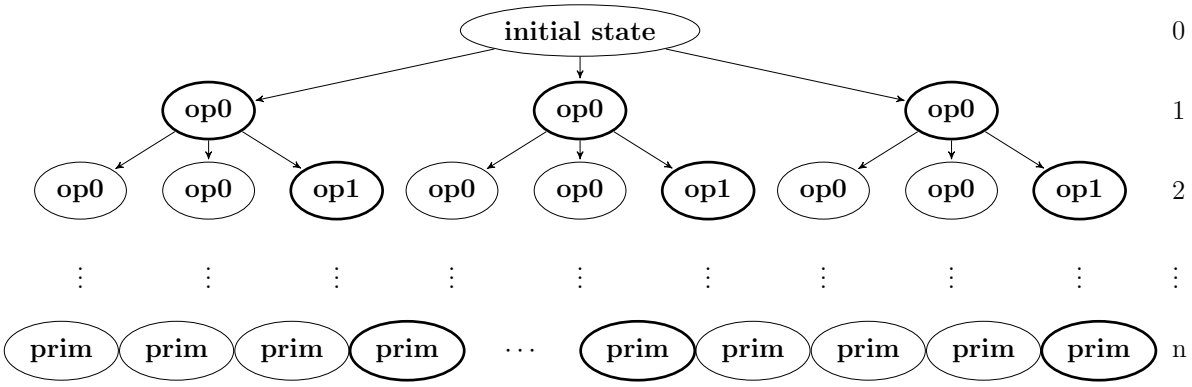

Figure 2: Planning tree with $k = 3$, where op$x$ indicates an operator at abstraction level x and prim represents a primitive action. There are n levels in the tree.

3 Domain Description

This section covers the application of BHPN to the UAV domain. Each component described in Section 2.2 will be given in detail as it applies to our UAV mission environment. This section will cover fluents used to describe the environment as well as the operators used to plan in this environment. The regression functions used in the regression search will be outlined, as well as our realistic observation model.

3.1 Problem Specification

In our mission environment there are one or more UAVs, ground troops, targets, and/or threats. The UAV states are observable and their actions are deterministic. All other objects in the world are only partially observable by the UAV through observations. Our initial belief about their locations is limited. The UAVs operate in a 2D plane above the ground while the other objects in the world remain in a 2D plane below the UAVs. In general, the UAV must navigate around an *a priori* map of the environment's stationary obstacles (e.g. mountains) while also performing on-line planning to avoid threats in the ground plane. The UAV may be tasked with goals such as locating a target, delivering supplies to ground troops, and other mission-like objectives. All UAV actions will be described in detail below.

3.2 Operators

Below is the list of operators for our UAV domain. They consist of the UAV actions pertaining to our particular mission objectives. The Observe operator gives us (noisy) location information. The Fly operator and the uavLoc() fluent do not have a probability associated with them as the UAV's actions are deterministic in our domain. In the operators below, an object can be a target, a threat, or ground troops. There is only one UAV. The fluents are described below, then the generators, and finally the operators. Each operator has a short description along with it. It should be noted that whenever a fluent is not assignmed a value

it is assumed to be True. For example, if an operator has the preciondition objLoc(obj, p), then assume it means objLoc(obj, p) = True, meaning the obj's location must be known with at least $1 - p$ certainty.

Fluents

clearPath(path, p) : the path is free of threats with at least probability p

uavLoc(UAV) = loc : the UAV is located at loc

uavState(UAV) = state : the state of the UAV for state $\in$ {none, relay, prosecute, jamming}

objLoc(obj, p) : The obj's location is known with at least probability $1 - p$

objState(obj) = state : the state of the obj for state $\in$ {none, prosecuted, jammed, tracked}

objType(obj) = type : the type of the obj for type $\in$ {threat, target, gt}

surveyed(region) : the given region has been surveyed (visited) within the last 10 seconds

havePermission(UAV, threat) : the UAV has permission to prosecute the given threat

suppliesDelivered() : the UAV has successfully delivered supplies to the ground troops

Generators

path = genFastestPath(start, dest) : the path is the fastest path from start to dest

path = genSurveyPath(start, region) : the path covers the entire rectangular region starting from the start location

loc = genLikelyLoc(obj) : the obj is most likely located at loc

loc = genRelayLoc(gtLoc) : a location in which the UAV has a LOS to the ground troop loc (gtLoc) as well as the control tower

start = genStartLoc(region, uavLoc) : start is the location from which the UAV should start surveying the region

threats = genThreatsInPath(path) : a list of threats potentially in the given path

Operators

Fly(UAV, start, dest, path)

Preconditions:

0: uavLoc(UAV) = start

1: clearPath(path, p_{fly})

Generators:

path = genFastestPath(start, dest)

Effect: uavLoc(UAV) = dest

The Fly operator moves the UAV from start to dest along the generated path. First, the UAV verifies that the path is clear and then it verifies that the UAV is in the correct starting location. As it flies it is also observing its surroundings and updating the survey times in the cells it passes through.

Observe(UAV, object, loc, ϵ):

Preconditions:

0: objLoc(object, $regress(\epsilon)$)

1: objLoc(object, $p_{observe}$)

2: uavLoc(UAV) = loc

Generators:

loc = genLikelyLoc(object)

Effect: objLoc(object, ϵ)

The Observe operator allows the UAV to increase its knowledge about an object's location to $1-\epsilon$. The first precondition is that the object's location is known with at least $1-regress(\epsilon)$. The Observe action can only be taken if the object's initial probability is greater than some threshold value, $1 - p_{observe}$. If it is not, then the UAV must get the object's location from the ground troops (see below). Lastly, the UAV must be located in the most likely location of the object.

GetInfoFromGT(UAV, gt, object, ϵ)

Preconditions:

0: objLoc(object, $p_{observe}$) = False

1: objLoc(gt, p_{gtLoc})

2: objLoc(object, $regress(\epsilon)$)

3: uavLoc(UAV) = loc

Generators:

loc = genLikelyLoc(gt)

Effect: objLoc(object, ϵ)

The GetInfoFromGT operator allows a UAV to gather information regarding an object's location through communication with the ground troops. The object's initial belief must be below $1 - p_{observe}$ (as opposed to the Observe action where is had to be above $1 - p_{observe}$). This is to ensure that this operator is only used as a last resort when very little is known about the object's location. Second, the UAV must know the ground troops location with at least $1 - p_{gtLoc}$. Third, we must know the object's location with at least $1 - regress(\epsilon)$. Lastly, the UAV must be located at the ground troop's most likely location.

Survey(UAV, region, start, path)

Preconditions:

0: clearPath(path, p_{fly})

1: uavLoc(UAV) = start

Generators:

start = genStartLoc(region, uavLoc)

path = genSurveyPath(start, region)

Effect: surveyed(region)

Survey allows the UAV to gather information about a specific region. A path is generated

such that, when traversed, all locations in the region are within the UAV's visibility radius. First, the UAV verifies that this path is clear and then it checks that it is in the correct starting location.

Track(UAV, object, loc)

Preconditions:

0: objLoc(obj, p_{track})

1: uavLoc(UAV) = loc

Generators:

loc = genLikelyLoc(object)

Effect: objState(object) = tracked

The Track operator enables a UAV to track an object (if it is not a threat). First, it must know the object's location with at least $1-p_{track}$. Second, it ensures that the UAV is located in the most likely location of the object.

ClearPath(UAV, path)

Preconditions:

0: threatState(threat) = prosecuted for threat $\in$ threatsInPath

Generators:

threatsInPath = genThreatsInPath(path)

Effect: clearPath(path)

The ClearPath operator is what changes a path from not being clear to being clear. First, it generates a list of all threats in the path. Then it makes a list of preconditions which state that each threat in the path must be prosecuted before the path can be cleared.

Prosecute(UAV, threat)

Preconditions:

0: havePermission(UAV, threat)

1: objLoc(threat, $p_{prosecute}$)

2: uavLoc(UAV) = loc

Generators:

loc = genLikelyLoc(threat)

Effect: objState(threat) = prosecuted

This operator allows a UAV to prosecute a threat. First the UAV checks that it has permission from the human operator. Then it ensures that it knows the threats location with at least $1 - p_{prosecute}$. Then it moves the UAV to the threat's most likely position.

DeliverSupplies(UAV, gt)

Preconditions:

0: objLoc(gt, $p_{deliver}$)

1: uavLoc(UAV) = loc

Generators:

loc = genLikelyLoc(gt)

Effect: suppliesDelivered()

Sometimes it may be necessary for a UAV to deliver supplies to ground troops. In order to perform this action, a UAV must know the ground troop's location with at least $1 - p_{deliver}$. Then the UAV must move to the ground troop's most likely position in order to deliver the supplies.

Relay(UAV, gt)

Preconditions:

0: objLoc(gt, p_{relay})

1: uavLoc(UAV) = relayLoc

Generators:

gtLoc = genLikelyLoc(gt)

relayLoc = genRelayLoc(gtLoc)

Effect: uavState(UAV) = relay

When communication is lost between the ground troops and the control tower, the UAV may be needed to act as a relay between the two. In order to do this the UAV must know the location of the ground troops with at least $1 - p_{relay}$, and then must be situated at the relay location where it has a LOS between both the UAV and the ground troops.

3.3 Regression Functions

Regression functions are used to calculate necessary precondition probabilities based on required effect probabilities. For example, if a UAV needs to know the location of a target with 0.9 certainty, how certain must is *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 the UAV is directly overhead the target being observed and it does not receive a false positive. Below is the regress function where p_{cp} is the probability of a correct positive, and p_{fn} is the probability of a false negative. It is derived from the UAV's observation model, which will be discussed in Section 3.4.

The following derivation for the regression function comes from work by Kaelbling and Lozano-Pérez [19]. 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 $regress(\epsilon)$, it is assumed 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, BHPN will simply replan, and most likely add more Observe operators to attempt to increase the belief again.

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

Before the Observe action occurs, the object being observed is in location l with some probability $1-\epsilon_r$, such that $1-\epsilon_r = Pr_b(L=l)$, where b represents the belief state before the Observe action takes place. After the action is executed, the belief becomes $1-\epsilon = Pr_{b'}(L=l)$, where b' is the updated belief state after the action takes place.

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

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

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

Then, solving for ϵ_r:

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

3.4 Observation Model

In UAV missions, observations are acquired through UAV camera data using object recognition algorithms. However, our research focuses on planning and not object recognition. In our simulated environment, camera data is modeled as a process 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 also 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 $\mathbf{x}_k$. $\mathrm{R}(\mathbf{x}_k)$ is the set of states within r of the UAV at time step k.

$$\mathrm{R}(\mathbf{x}_k) := \{s \mid ||\mathbf{x}_k - s|| < r\} \text{ where } s \in \mathbb{R}^2 \tag{1}$$

$$\mathbf{z}_k \in \{(x_k, y_k)\} \text{ where } (x_k, y_k) \in \mathrm{R}(\mathbf{x}_k) \tag{2}$$

$P(\mathbf{O}_k)$ 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 [11]. If an object is in $\mathrm{R}(\mathbf{x})$, 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. An incorrect object position is designated as o_{incorr}. If the object is not in $\mathrm{R}(\mathbf{x})$, 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.

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

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 target 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.

4 Complexity Analysis

4.1 BHPN Complexity

As mentioned in Section 2.2.2, 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 recursion 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, the overall complexity of the planning tree is the sum of all 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 $k = l^{1/n} << l$.

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. There are two instances where these assumptions do not hold. First, 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 the UAV still knows where it is. In this case actions are not permanent. Second, subgoals may interact if the action of one undoes a precondition of the other.

This analysis only accounts for the task planning component of the overall BHPN al-

gorithm. The motion planning complexity is dependent on the motion planning algorithm in place. For this simulation A* was implemented to perform UAV motion planning. To evaluate a *combined* task and motion planner is difficult due to the heavy reliance on current belief state and mission objectives. Some missions may prove to be very difficult in a motion planning sense due to the number of threats in the world. Other missions might be very easy to execute in a motion planning sense, but the goal contains many fluents, therefore complicating the task planning.

4.2 POMDP Comparison

Solving a POMDP optimally can be done for some small 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. 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 HPN 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.

5 Single-Agent Planning Evaluation

5.1 Mission Scenarios

Below are examples of mission scenarios which were simulated in Gazebo using ROS (robotic operating system). The Figures referenced in the scenario descriptions are the hierarchical plan decompositions BHPN generates as the mission is planned and executed. They are constructed in a depth-first manner. The green nodes represent primitive actions, or actions being executed in the world. Planning does not continue until the green nodes have been executed. The pink nodes represent partial plans. The number after an operator in a plan refers to the abstraction level at whuch that operator is being considered. For example, Observe:0 means that only the 0th level precondition of the Observe operator is being considered at that particular level of planning. The blue nodes represent subgoals, or lists of fluents which BHPN is attempting to satisfy.

objLoc(target1, 0.9) = True

The UAV needs to know the location of target1 with at least 0.9 certainty. The planning tree created from this scenario is shown in Figure 3. The UAV's initial belief state of target1 is below $p_{observe}$ (0.1 in this case), and so the UAV gets information from the ground troops. Then, it flies to the likely location of target1 to confirm that it is there, once its belief is above $p_{observe}$.

surveyed(region1) = True

In this scenario the UAV is attempting to survey a given region. The planning tree is shown in Figure 4. The preconditions for surveying require that the generated survey path be clear of threats. In the preconditions, it is found that there is potentially a threat is the survey region. The UAV flies to directly prosecute the threat and clear the path. As the UAV flies

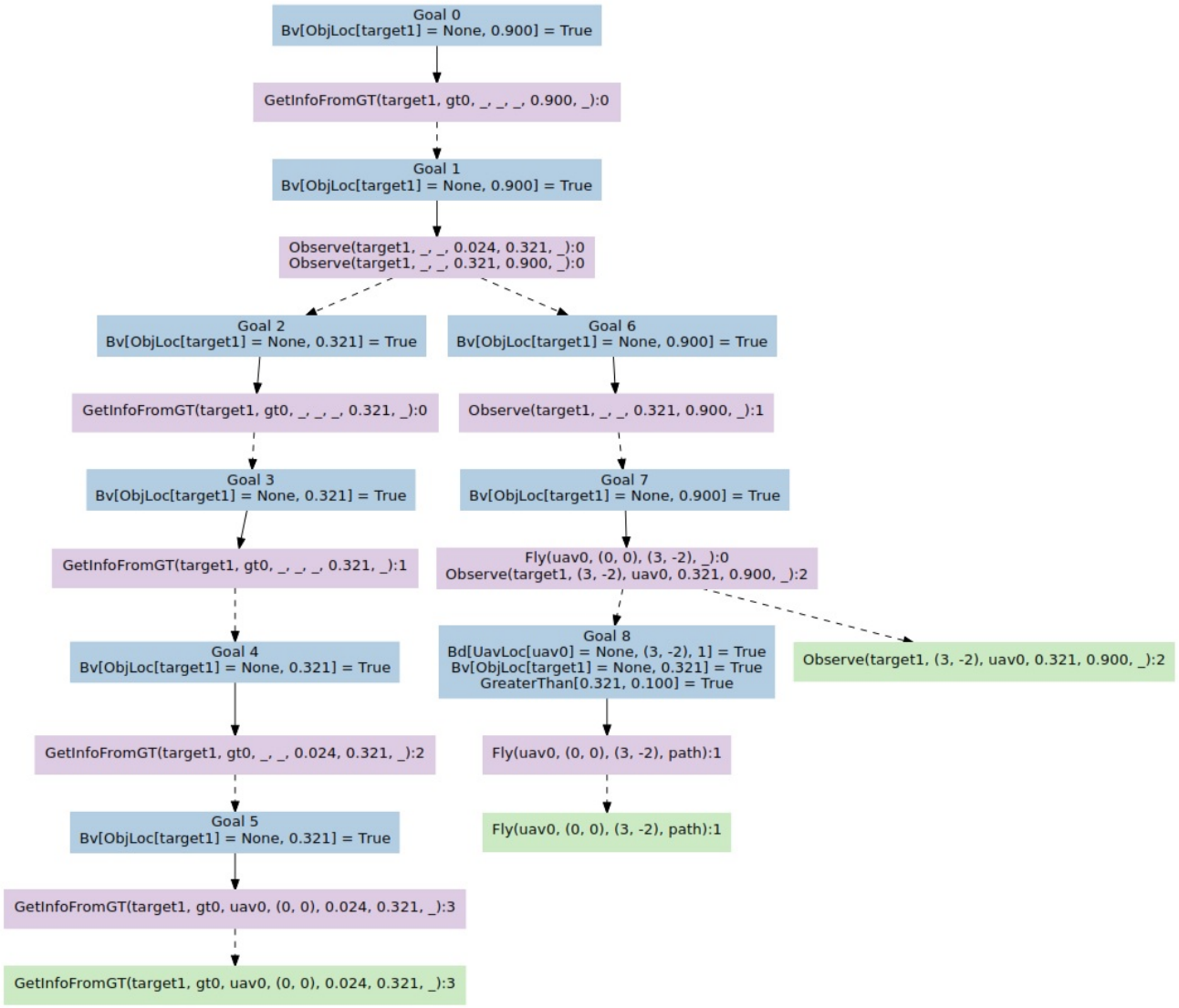

Figure 3: Goal: **objLoc(target1, 0.9) = True**

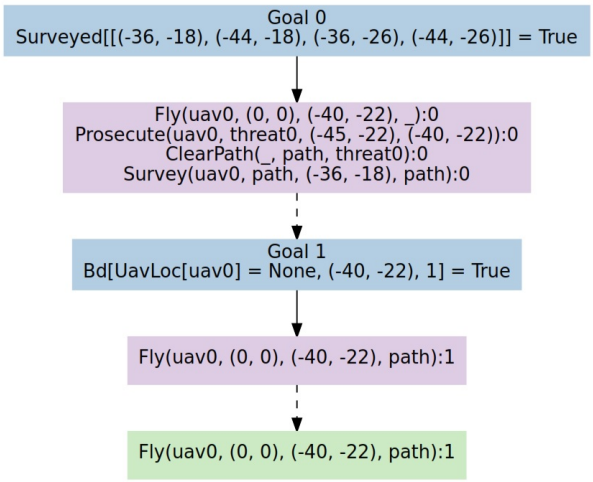

Figure 4: Goal: **surveyed(region1) = True**

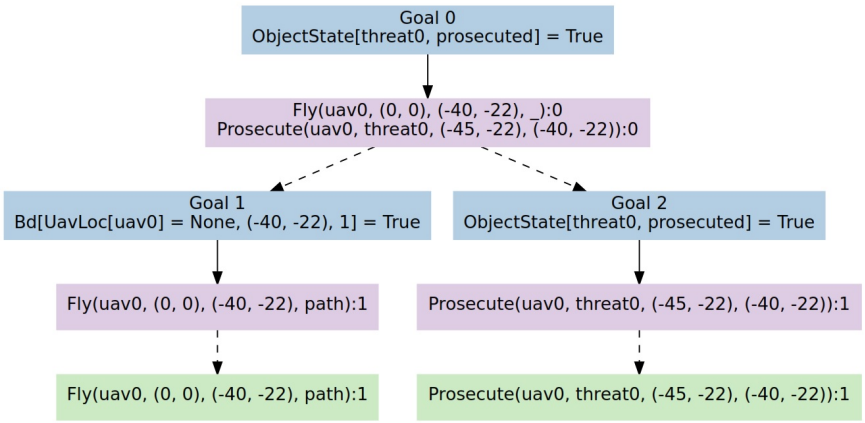

Figure 5: Goal: **objState(threat1) = prosecuted**

it is constantly surveying as its sensors are always on. Therefore on its way to prosecute the threat it surveys the given region and the goal is satisfied.

objState(threat1) = prosecuted

The planning tree for this goal is shown in Figure 5. First, the UAV has to get permission from the human operator to prosecute the threat. Then, it must know where the threat is with at least probability $p_{prosecute}$. Finally the UAV prosecutes the threat.

suppliesDelivered() = True and uavState(UAV) = relay

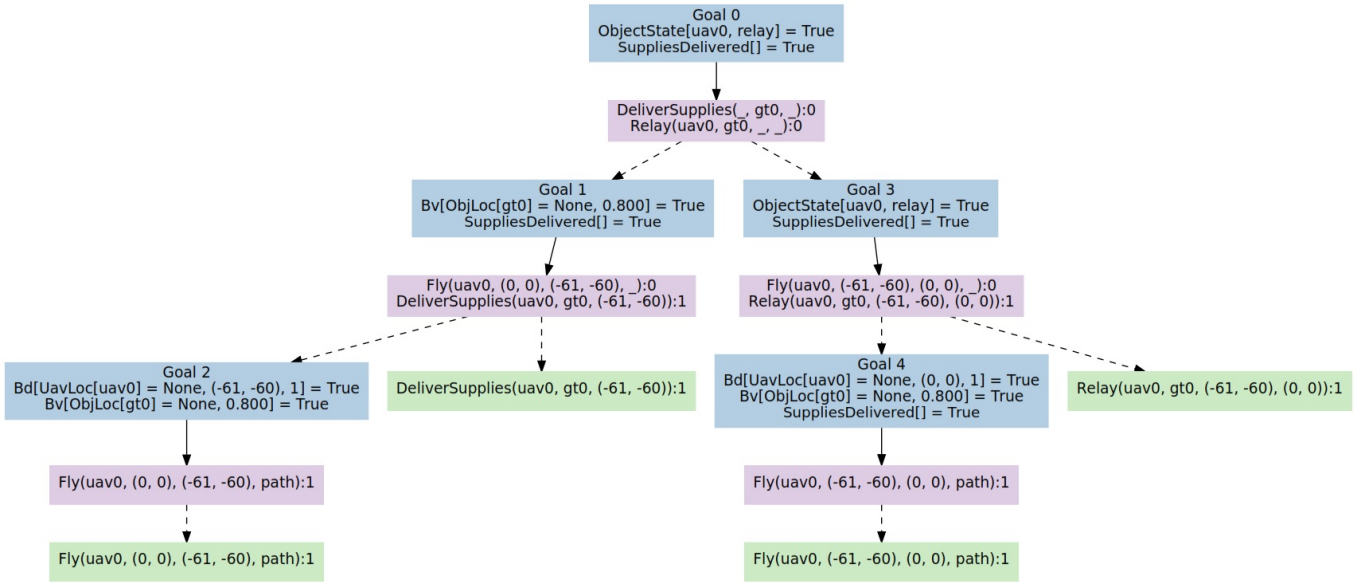

Figure 6: Goal: **suppliesDelivered() = True and uavState(UAV) = relay**

The planning tree for this goal is shown in Figure 6. The UAV is tasked with delivering supplies to the ground troops, then acting as a relay between the ground troops and control tower. Assuming the UAV has supplies with it, the UAV must verify the location of the ground troops. Once this is done, it can drop the supplies down to the ground troops. Then, using its current location (above the ground troops) and the control tower location, it determines the best (least threatening) location where it can achieve a LOS between the ground troops and control tower, flies there and acts as a relay.

5.2 Simulation Results

To display the effectiveness of the single-agent mission planner developed, a realistic ROS-based simulation was developed. This section outlines one of the missions executed with the simulation. Figure 7 gives the hierarchical plan decomposition, and Figures 8- 11 give screenshots of the mission environment as the primitive actions are executed.

The mission objective is to localize a specific target to a certainty of 0.95. The UAV has

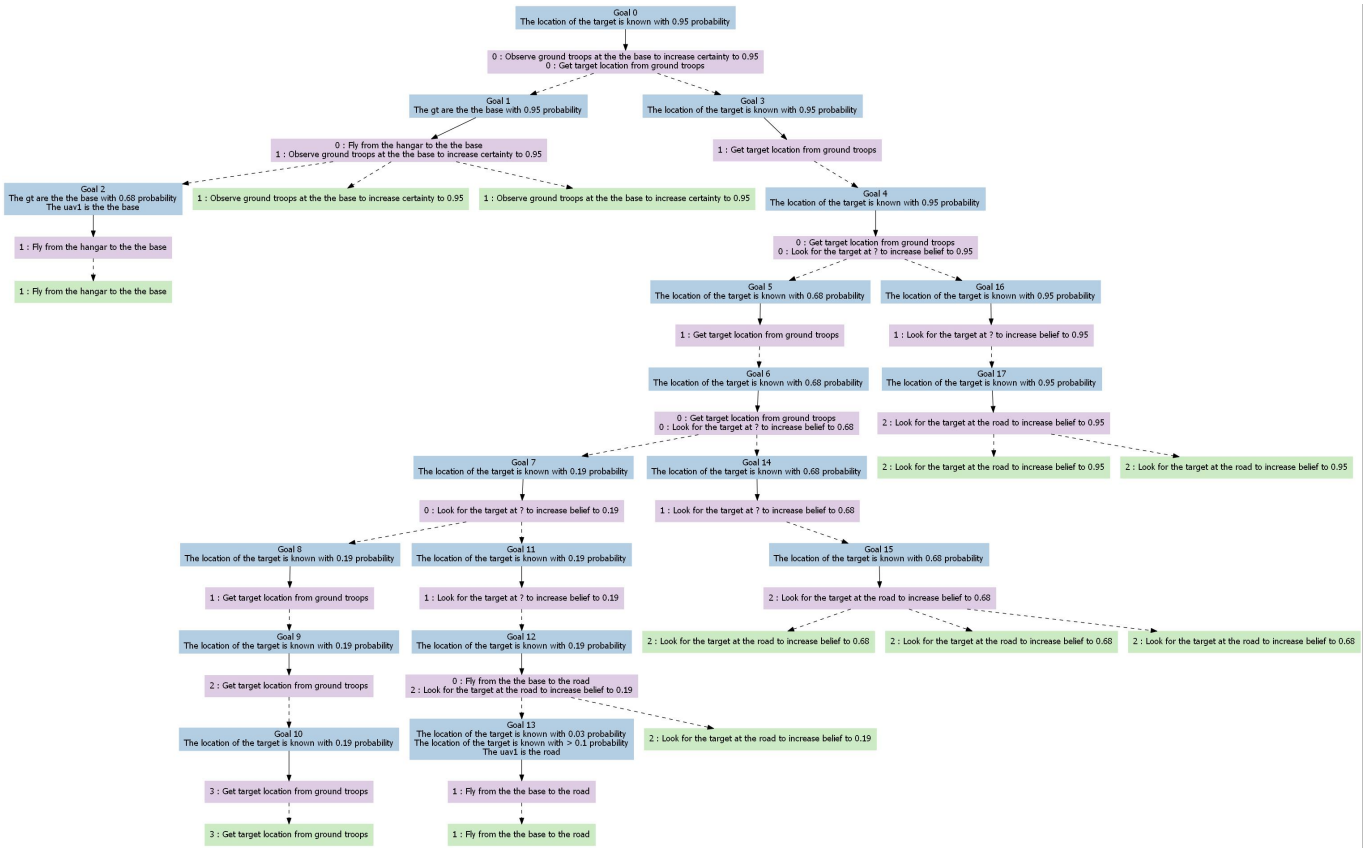

Figure 7: This is the planning tree constructed while accomplishing the overall goal of knowing the target location with 0.95 certainty (Goal 0). The blue nodes represent goals and subgoals (or pre-images), the pink nodes represent plans, and the green nodes represent primitive actions being executed. The numbers before each operator in the plan nodes are the respective operator abstraction levels.

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.

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,

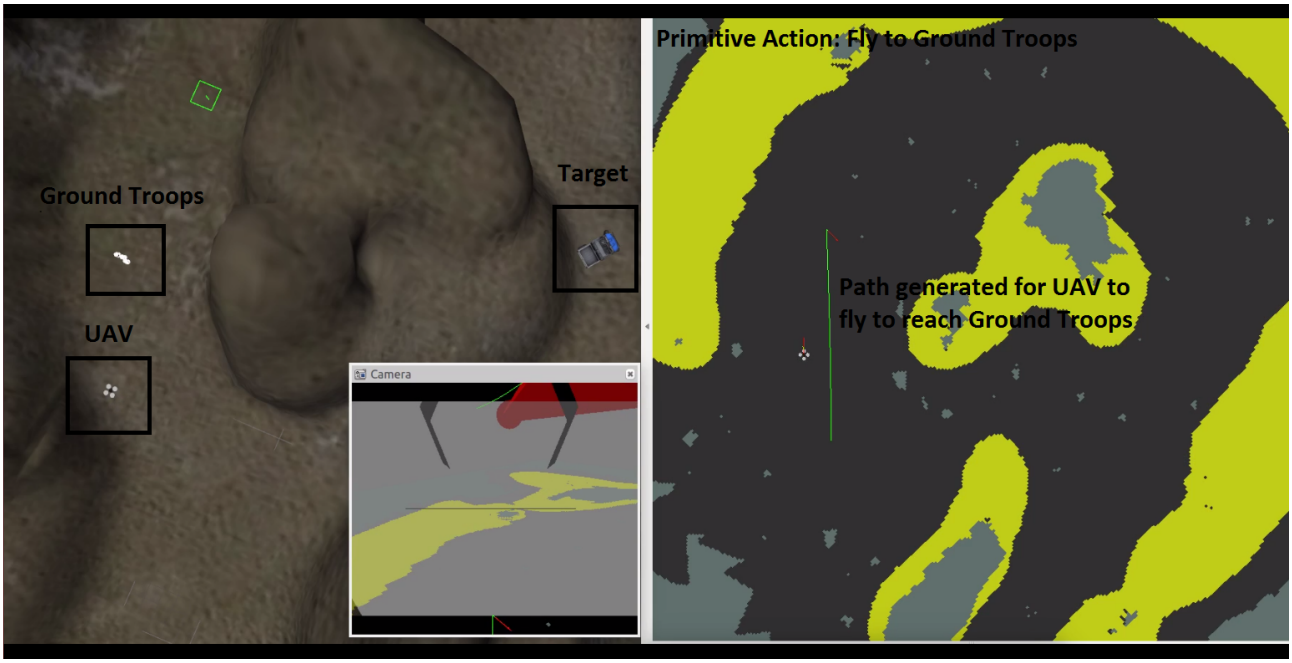

Figure 8: The first primitive action to be returned by the BHPN algorithm is to fly to the ground troops. In our simulation they are represented by several Turtlebots (the small white robots seen in the figure). The green path is the global path returned by the path planner in the Fly primitive action.

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 3.4 and the underlying world state.

Figures 8-11 give screenshots of the mission. Each screenshot corresponds to a primitive action being executed (the green nodes in Figure 7). 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

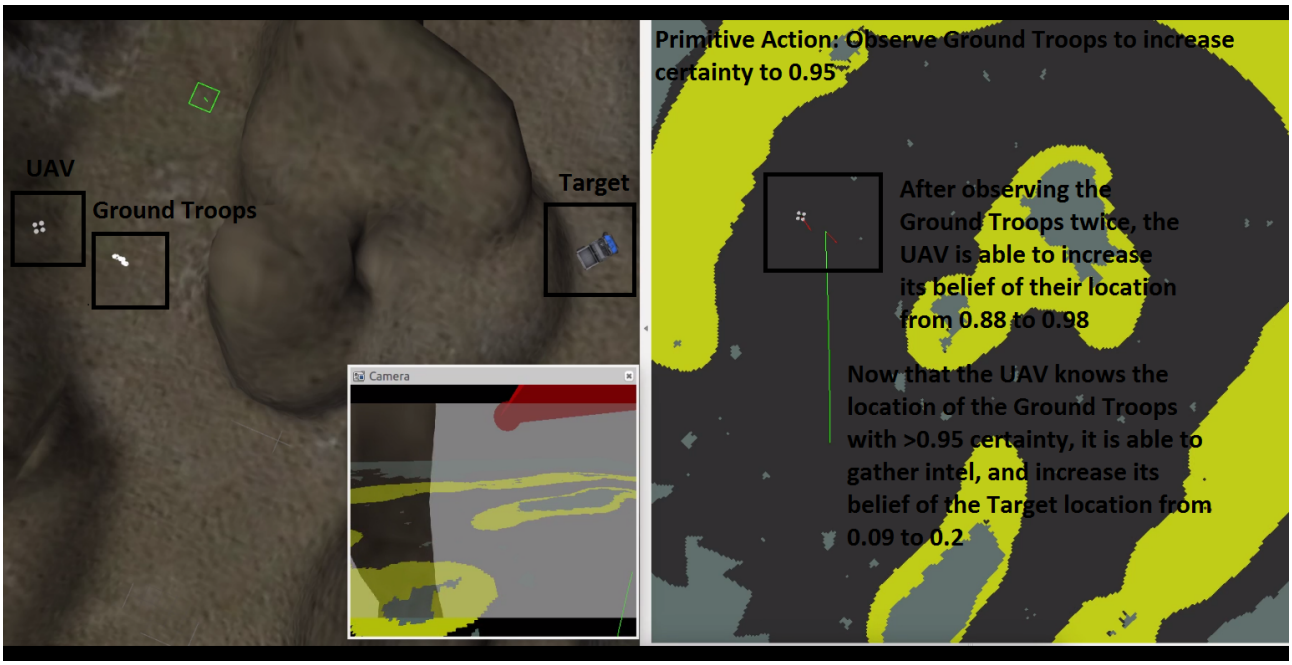

Figure 9: The next primitive action is to observe the ground troops in order to verify that they are actually where the UAV thought that they would be. This primitive must be performed twice because initially the quadrotor does not see the ground troops. Replanning occurs, and it observes again, this time confirming that the ground troops are there. This corresponds to the second and third green nodes in Figure 7. Then, the UAV gets intel from the ground troops regarding the location of the target. This corresponds to the fourth green node in Figure 7.

Figure 10: 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.

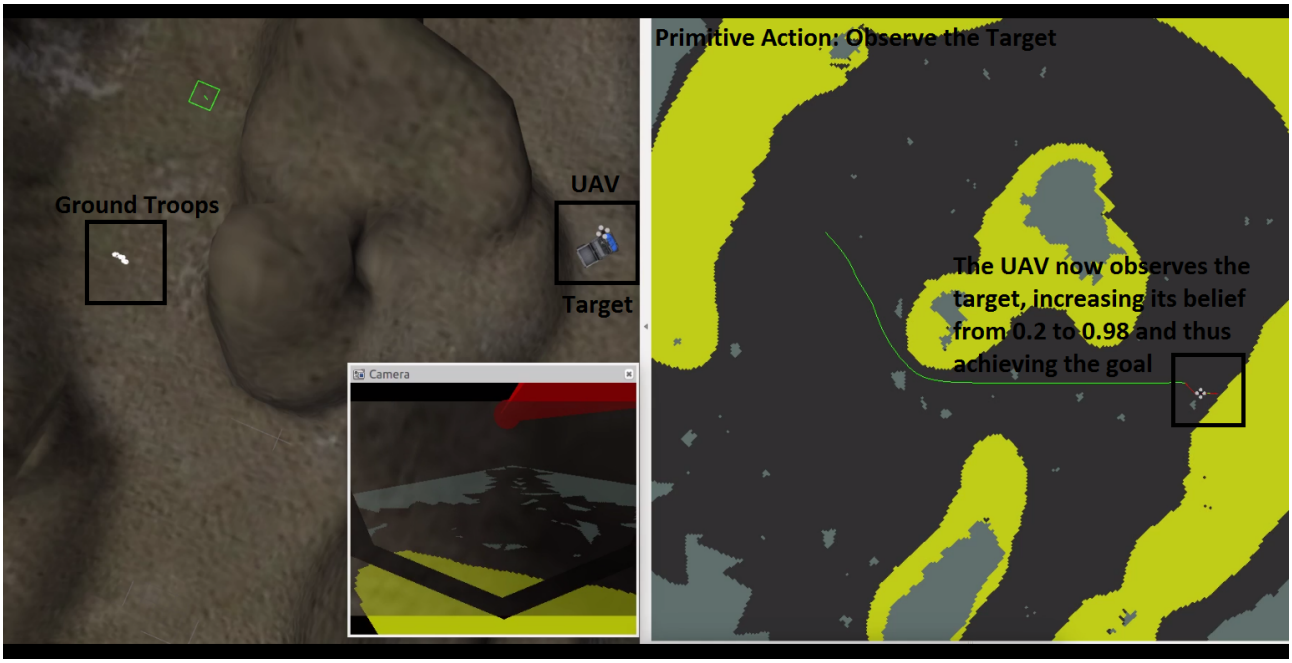

Figure 11: Finally, the UAV observes the target several times to verify that it is in the believed location. The reason so many observations are made (as you can see in Figure 7) is due to the noise in the sensor.

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.

6 Multi-Agent Planning Evaluation

In the multi-agent planning mission environment, multiple UAVs are used to achieve different objectives. The UAVs are heterogeneous, meaning that they all have different abilities. For example, one UAV may be the only one with the ability to prosecute, while another is the only one with the ability to act as a relay. In this environment, all UAVs are knowledgeable about their own and each other's abilities. All operators remain the same as in the single-agent planning domain, except for one key difference. The first precondition of all operators will be:

isAvailableUAV('UAV')

When this precondition is being evaluated, a genUAV() generator is called. In single-agent task planning this generator would always bind the acting UAV to the 'UAV' argument. However, in multi-agent planning, this generator will evaluate other potential UAVs. It has access to all UAV's abilities, as well as the UAV attempting to execute this operator. It binds a UAV that is able to complete this task. It always checks the UAV making the call first, to see if it is capable of performing this action. If it is not, then it asks other UAVs with this capability if they are available to help. If none of them are, then it binds 'None' to the 'UAV' arg, and the isAvailableUAV('UAV') fluent will evaluate to be False, and planning will fail.

In this environment, each UAV has its own goal that it is trying to satisfy. Since planning is decentralized, in order for a UAV to receive help, there needs to be a method of ranking the importance of one's own current primitive or overall goal against the importance of another agent's objective (the one who is asking for help). In this implementation, this is achieved by ranking the importance of each operator. When a UAV is being asked to help another, it compares the operator it is currently executing to the one it is being requested to perform, and chooses which is more important.

6.1 Simulation Results

To display the effectiveness of MAP BHPN this section given a MAP mission execution. In this scenario UAV0 and UAV1 are attempting to achieve independent goals. UAV0 is attempting to deliver supplies to ground troops and UAV1 is attempting to survey a given region. However, UAV0 does not have the sensor capabilities to localize the ground troops before it can deliver the supplies. UAV1 has the capabilities to perform the observe operator. As UAV1 is about to begin its survey operator, UAV0 requests help, so UAV1 return to localize the ground troops. Once it completed this and tells UAV0 it was successful, UAV0 flies to the ground troops and delivers the supplies, and UAV1 returns to the survey region to complete its survey operator. Figures 12-16 give a visual of the mission environment as this was planned and executed. The left images represent the mission environment. The ground troops are represented by the turtlebots (small white dots on the left), the UAVs are the two quadrotors, and the Polaris vehicles are non-threatening targets. The right images display is the RViz environment used in planning. The yellow and black represents the costmap used when path planning. The green lines are the paths the UAVs generate and follow, and the red arrows represent the goal locations.

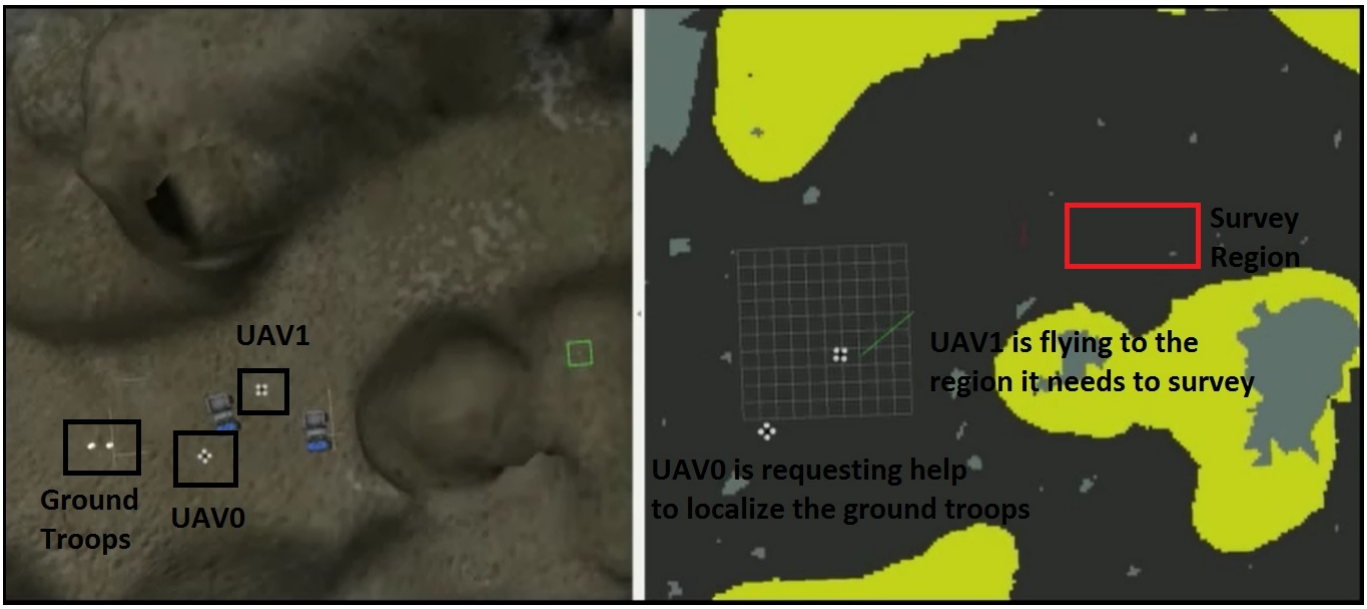

Figure 12: UAV1 has been tasked with surveying the red region. In this image, UAV1 is in transit to the survey region. UAV0 has just begun planning.

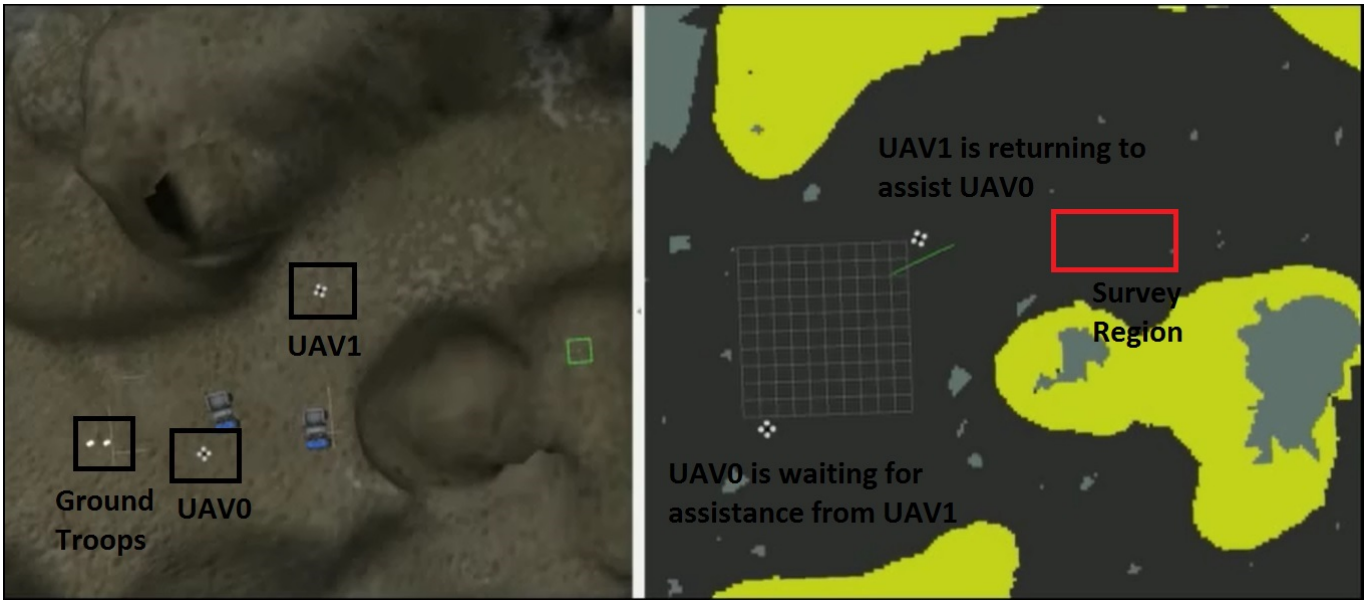

Figure 13: UAV0 needs to localize the ground troops in order to deliver their supplies, however it does not have the sensor capabilities to perform this action. Therefore it communicated with UAV1, and UAV1 returns to help UAV0.

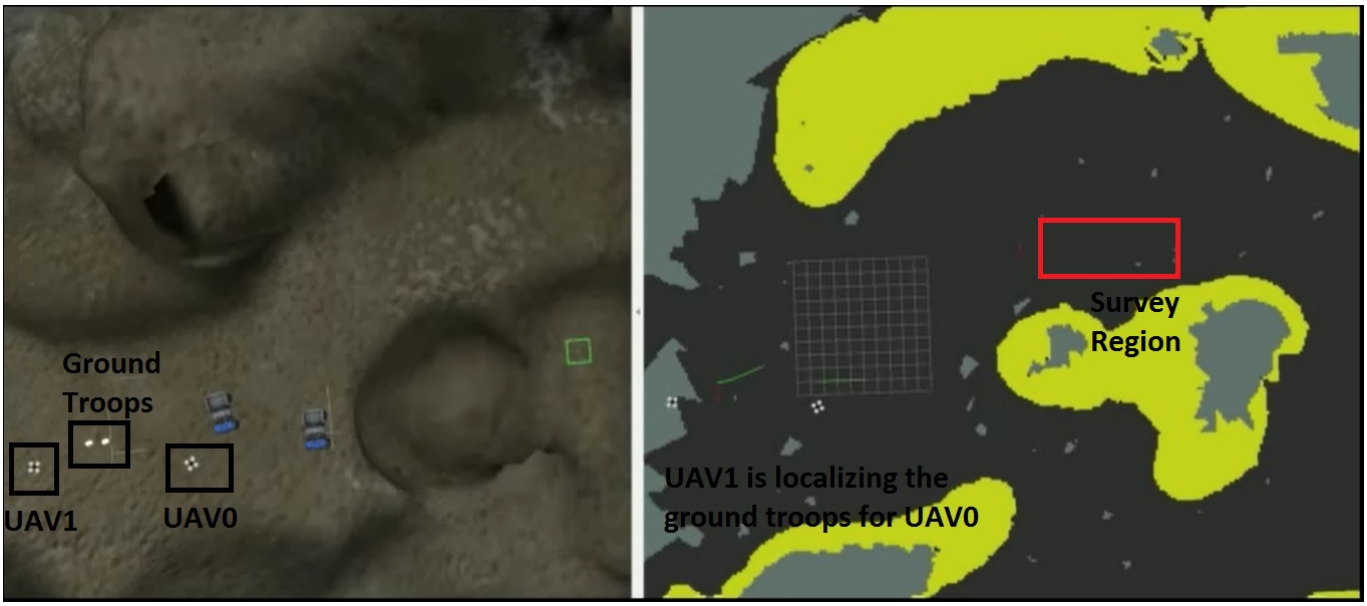

Figure 14: UAV1 localizes the ground troops and communicates to UAV0 that the action was successful.

Figure 15: UAV0 is in transit to the ground troops now that it knows their location. Once it arrives, it delivers the necessary supplies. UAV1 has returned to its original objective, and is in transit ot the survey region.

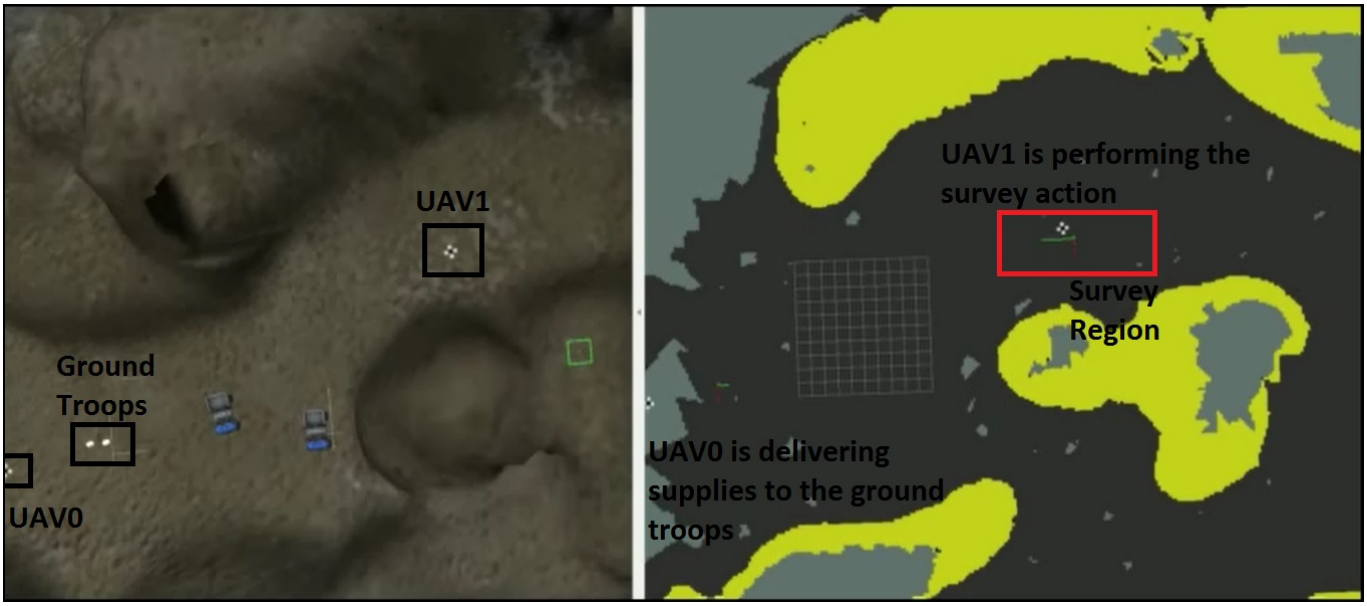

Figure 16: Finally, UAV1 surveys the given region, and all decentralized goals have been achieved.

7 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 [23]. 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. The BHPN algorithm was successfully reapplied 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 was shown, it is able to replan when actions do not have intended results, all the while performing interleaved planning and execution. The framework outlined is also able to plan for many different mission objectives. An analysis on the BHPN complexity was also given, as well as a comparison to solving a POMDP which it the traditional method of solving belief space problems. This system was then extended to handle decentralized multi-agent planning problems in loosely-coupled domains. The framework enables multiple UAVs with different abilities to communicate and cooperate to achieve independent goals.

In the future more work could be done in the coordination between UAV plans. As it is now, UAVs do not replan to help other UAVs efficiently. If they are tasked with helping another, they immediately go and execute the assisting action. Instead, it could potentially be beneficial is the assisting UAV could work the subgoal it needs to help with into its current planning. A replanning methodology would be required to implement such a feature. Work in the field of replanning in MAP problems has been done by Talamadupula, Smith, Cushing, and Kambhampati [32] as well as Komenda, Novak, and Pechoucek [20].

References

[1] Fahiem Bacchus and Qiang Yang. Downward refinement and the efficiency of hierarchical problem solving. *Artificial Intelligence*, 71(1):43–100, 1994.

[2] Anthony Barrett and Daniel S Weld. Partial-order planning: Evaluating possible efficiency gains. *Artificial Intelligence*, 67(1):71–112, 1994.

[3] Michael Beetz, Ulrich Klank, Ingo Kresse, Alexis Maldonado, L Mosenlechner, Dejan Pangercic, T Ruhr, and Moritz Tenorth. Robotic roommates making pancakes. In *Humanoid Robots (Humanoids), 2011 11th IEEE-RAS International Conference on*, pages 529–536. IEEE, 2011.

[4] Frederic Bourgault, Tomonari Furukawa, and Hugh F Durrant-Whyte. Coordinated decentralized search for a lost target in a bayesian world. In *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 1, pages 48–53. IEEE, 2003.

[5] Tom Bylander. The computational complexity of propositional strips planning. *Artificial Intelligence*, 69(1):165–204, 1994.

[6] Stephane Cambon, Rachid Alami, and Fabien Gravot. A hybrid approach to intricate motion, manipulation and task planning. *The International Journal of Robotics Research*, 28(1):104–126, 2009.

[7] Stephen A Cambone. *Unmanned aircraft systems roadmap 2005-2030*. Defense Technical Information Center, 2005.

[8] Stefano Carpin, Derek Burch, and Timothy H Chung. Searching for multiple targets using probabilistic quadtrees. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 4536–4543. IEEE, 2011.

[9] Caroline Ponzoni Carvalho Chanel, Florent Teichteil-Königsbuch, and Charles Lesire. Multi-target detection and recognition by uavs using online pomdps. In *27th AAAI Conference on Artificial Intelligence*, 2013.

[10] Jaesik Choi and Eyal Amir. Combining planning and motion planning. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 238–244. IEEE, 2009.

[11] Michael Dille. Search and pursuit with unmanned aerial vehicles in road networks. Technical report, DTIC Document, 2013.

[12] Kutluhan Erol, James Hendler, and Dana S Nau. Htn planning: Complexity and expressivity. In *12th AAAI Conference on Artificial Intelligence*, volume 94, pages 1123–1128, 1994.

[13] Richard E Fikes and Nils J Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3):189–208, 1972.

[14] Christopher Geyer. Active target search from uavs in urban environments. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 2366–2371. IEEE, 2008.

[15] Julien Guitton and Jean-Loup Farges. Taking into account geometric constraints for task-oriented motion planning. *Proc. Bridging the gap Between Task And Motion Planning, BTAMP*, 9:26–33, 2009.

[16] Kris Hauser. Task planning with continuous actions and nondeterministic motion planning queries. In *Proc. of AAAI Workshop on Bridging the Gap between Task and Motion Planning*, 2010.

[17] Milos Hauskrecht. Value-function approximations for partially observable markov decision processes. *Journal of Artificial Intelligence Research*, pages 33–94, 2000.

[18] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical task and motion planning in the now. In *IEEE Conference on Robotics and Automation (ICRA)*, 2011. Finalist, Best Manipulation Paper Award.

[19] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Integrated task and motion planning in belief space. *International Journal of Robotics Research*, 32(9-10), 2013.

[20] Antonín Komenda, Peter Novák, Michal Pechoucek, Raz Nissim, Daniel L Kovacs, and Ronen Brafman. How to repair multi-agent plans: Experimental approach. In *Proceedings of Distributed and Multi-agent Planning (DMAP) Workshop of 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)*. Citeseer, 2013.

[21] Scott A Miller, Zachary A Harris, and Edwin KP Chong. A pomdp framework for coordinated guidance of autonomous uavs for multitarget tracking. *EURASIP Journal on Advances in Signal Processing*, 2009:2, 2009.

[22] Kevin P Murphy. A survey of pomdp solution techniques. *environment*, 2:X3, 2000.

[23] S. Niekum, S. Osentoski, G.D. Konidaris, and A.G. Barto. Learning and generalization of complex tasks from unstructured demonstrations. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, October 2012.

[24] Raz Nissim and Ronen I Brafman. Multi-agent a* for parallel and distributed systems. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 3*, pages 1265–1266. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

[25] Raz Nissim, Ronen I Brafman, and Carmel Domshlak. A general, fully distributed multi-agent planning algorithm. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 1323–1330. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[26] Damien Pellier. Distributed planning through graph merging. In *International Conference on Agents and Artificial Intelligence*, 2010.

[27] Caroline Ponzoni Carvalho Chanel, Florent Teichteil-Königsbuch, and Charles Lesire. Pomdp-based online target detection and recognition for autonomous uavs. In *The 20th European Conference on Artificial Intelligence*, 2012.

[28] Stéphane Ross, Joelle Pineau, Sébastien Paquet, and Brahim Chaib-draa. Online planning algorithms for POMDPs. *Journal of Artificial Intelligence Research*, 2008.

[29] Earl D. Sacerdoti. Planning in a hierarchy of abstraction spaces. Technical Report 78, AI Center, SRI International, 333 Ravenswood Ave, Menlo Park, CA 94025, June 1973.

[30] David Silver and Joel Veness. Monte-Carlo Planning in Large POMDPs. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 2164–2172. Curran Associates, Inc., 2010.

[31] Adhiraj Somani, Nan Ye, David Hsu, and Wee Sun Lee. DESPOT: Online POMDP Planning with Regularization. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 1772–1780. Curran Associates, Inc., 2013.

[32] Kartik Talamadupula, David E Smith, William Cushing, and Subbarao Kambhampati. A theory of intra-agent replanning. Technical report, DTIC Document, 2013.

[33] Jason Wolfe, Bhaskara Marthi, and Stuart J Russell. Combined task and motion planning for mobile manipulation. In *ICAPS*, pages 254–258, 2010.