5-2024

Sliding Markov Decision Processes for Dynamic Task Planning on Uncrewed Aerial Vehicles

Trent Wiens
University of Nebraska-Lincoln, trent.wiens@huskers.unl.edu

SLIDING MARKOV DECISION PROCESSES FOR DYNAMIC TASK PLANNING ON UNCREWED AERIAL VEHICLES

by

Trent Wiens

A THESIS

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfilment of Requirements

For the Degree of Master of Science

Major: Mechanical Engineering and Applied Mechanics

Under the Supervision of Professor Justin Bradley

Lincoln, Nebraska

May, 2024

SLIDING MARKOV DECISION PROCESSES FOR DYNAMIC TASK PLANNING ON UNCREWED AERIAL VEHICLES

Trent Wiens, MS

University of Nebraska, 2024

Adviser: Justin Bradley

Mission and flight planning problems for uncrewed aircraft systems (UASs) are typically large and complex in space and computational requirements. With enough time and computing resources, some of these problems may be solvable offline and then executed during flight. In dynamic or uncertain environments, however, the mission may require online adaptation and replanning. In this work, we will discuss methods of creating MDPs for online applications, and a method of using a sliding resolution and receding horizon approach to build and solve Markov Decision Processes (MDPs) in practical planing applications for UASs. In this strategy, called a Sliding Markov Decision Processes (SMDP), the underlying state space is regularly rediscretized according to its informational proximity and utility while a receding horizon algorithm allows us to consider immediate next steps while keeping the primary goal state in mind. This approach allows for dynamic decision making and replanning by a UAS in an uncertain and dynamic environment in which mission objectives or the environment could change. The SMDP method shows an ability to create recursively optimal policies, under conditions of limited computing power and time, that perform similarly to the optimal policy of the associated fully-modeled flat MDP.

Acknowledgements

My special thanks goes to my advisor and committee members, Dr. Justin Bradley, Dr. Carl Nelson and Dr. Bhuvana Gopal, for their guidance on this thesis. Specifically, I would like to thank Dr. Bradley for his guidance throughout my Master's Degree, pushing me to improve as both a student and researcher. In addition, I would like to thank the NIMBUS Lab, for giving me a great environment to work on research I enjoy. Finally, I would like to thank my parents and siblings for their continuous support and motivation throughout my education.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Uncrewed Aircraft Systems (UASs) have become invaluable tools in a variety of mission types. Militaries have been utilizing such systems for years [10], scientists have begun widely exploring their use in research and data collection [24, 44], and multicopters have become a mainstay in film and photography. As the proliferation of UASs increases new techniques to increase automation in all facets including mission planning have become important. In most of these cases the decisions made are in the face of uncertainty as conditions and mission objectives can be dynamic. Decision making in stochastic environments is well explored and frameworks such as Markov Decision Processes (MDPs) have received wide attention [28, 37]. MDPs though, are not without limitation. State space explosion hampers their usefulness, particularly on Size, Weight, and Power (SWaP) constrained applications as the state space can grow exponentially with each variable upon which it is dependent. In these cases, researchers have developed tools to reduce the state space and therefore computing resources necessary to solve for the optimal policy [30, 36]. For example, factoring the MDP [12] allows independent variables to be considered separately, while reward gradients allow policies to be estimated [1].

MDPs first appeared in the 1960s [23] and have been applied to decision making and planning algorithms successfully [2]. MDPs are powerful because they produce

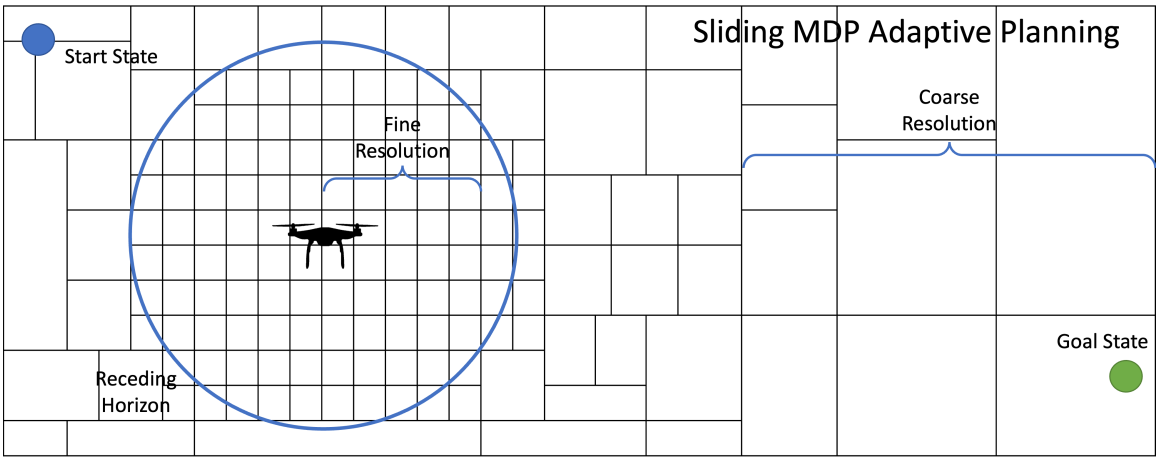


Figure 1.1: Sliding MDP rediscretizing a physical state space for a UAV.

optimal and easily executable policies in stochastic environments. An MDP, $\Sigma = \{S, A, P, R\}$, is described as a set of states (S), actions (A), transitions (P), and rewards (R). For each $s \in S$ and $a \in A$ there is a transition probability from one state to another represented by $P(s\prime|s, a)$ where $s\prime$ is the resulting state of taking action a in s. In the most general case, for each state and transition, a reward function, $R(s, a, s\prime)$, defines the reward given when the system transitions from s to $s\prime$ when taking action a. Rewards can be negative or positive representing costs or benefits to the system for taking an action. In traditional MDP design, solvers, such as policy or value iteration, iterate over the Bellman equation [3, 23] producing an optimal value function describing how advantageous each possible state is to visit [38]. From the value function and transition probabilities the model's optimal policy can be derived which is then implemented as a simple lookup table for runtime decision making. The optimal policy, π^*, maps states to actions, and is easily executed as long as the vehicle can observe its current state.

This is notably different than reinforcement learning, where the policy is found directly by rewarding behaviors that are beneficial and punishing behaviors that are detrimental. One advantage of MDPs is the development of a model of the system,

which a reinforcement learning program does not find. This model can be difficult to create initially, but can generally be easily adapted to other situations. Reinforcement learning, however, requires re-learning when the situation is changed.

MDPs that have the level of resolution needed to make high fidelity, optimal policies in a real-world environment are complex, and obtaining the optimal policy is computationally demanding. Typically, when MDPs are used to model these environments the optimal policy is calculated offline, where there is plenty of computing power, and then the optimal policy uploaded to the robot to be executed at runtime. This approach has several distinct disadvantages: 1) because the policy is decoupled from the model it becomes static making it unable to adapt to changes in the environment without updating the model and recomputing the policy; and 2) even if the model were updated the computation required to compute the optimal policy may be unrealizable for SWaP-constrained robotic systems. We address these two disadvantages by coupling the model and computation of the policy onboard the UAS, and developing a solver for use in any computational environment no matter how limited.

To do this we developed a hybrid strategy we call Sliding MDPs (SMDPs) wherein the classes of states of the MDP model are regularly rediscretized according to informational proximity and utility. SMDPs therefore shrink and expand the model to maintain higher local fidelity as needed, while a receding horizon keeps the MDP solvable with available onboard computing resources. SMDPs accomplish this by using a moving horizon window that only considers states currently accessible by the agent and subsequently building a more refined (in discretization) state space within it (see Figure 1.1). The sliding resolution algorithm determines appropriate discretizations of the classes of states in the MDP by leveraging the current value function. This allows us to make a series of local decisions from a time-varying MDP instead of solving a larger, fixed or "flat" MDP. This approach shares some similarities with [20], which

breaks MDPs into a series of subgoals and solves them individually. In our SMDP algorithm, each local computation produces an optimal local policy, which then can be pieced together to create a piece-wise optimal policy for the system, which we call "recursively optimal" [20]. By keeping the resolution low outside the horizon we shrink the total size of the state space making computation of the optimal policy much easier. Because the SMDP algorithm rediscretizes the classes of state at each step it can add and remove enumerated states making it possible to easily incorporate new information about the states on the fly.

Our research is motivated by a multi-flight, single-agent mission requiring a UAS to land and recharge its battery via a solar cell. We assume possible landing sites are known a priori and each site has an associated cost/reward. At each landing site the UAS must consider battery charge and time of day to make the next decision of when and where to fly. The state space associated with this complexity is much too large for online solving of the MDP onboard the UAS when a fine discretization is used to consider battery charge and time of day. If a more coarse discretization is used, the problem is easily solvable online but is far less useful. In either case, traditional methods do not allow for dynamic adjustment or adaptivity as the policy is determined before flight. Because SMDPs grow and shrink the space, adding or removing states as needed, they can be dynamically adjusted, and also easily solved at runtime to produce optimal policies on the fly.

1.1 Contributions

This research has vast implications as it improves the ability for a MDP to be used in contexts that may have not been feasible before. MDPs have the strength of being able to make decisions in stochastic environments, but it is difficult to find the policy

offline and generally intractable on SWaP contrained vehicles for all but the simplest models. In addition, the MDP is not adaptable as the environment changes, creating rigid policies that can quickly become not useful. The SMDP algorithm allows for the policy to be found at runtime, while still creating policies that are recursively optimal. This means that the model can be changed as an agent moves through the environment and becomes more adaptable to new information being gained. The best applications of this method are places where the next decision is most important, but decisions in the future are less important, or rely on information found from previous decisions. The SMDP algorithm allows for an agent to gain information from its environment and feed it back into the model to create policies that are well informed.

Overall, we contribute the following:

- A novel technique to reduce state space size of flat MDPs using a sliding resolution window within a receding horizon. We call this Sliding MDPs.
- The ability to automatically, at runtime, add/remove enumerated states (thus changing the SMDP) based on information obtained during the mission and in conjunction with the current value function.
- A solver for SMDPs that finds recursively optimal policies at runtime which approximate the policy from a high-resolution flat MDP given any available computation.
- An online, updateable multi-flight planner for solar-supplemented UAS.

Chapter 2

Related Work

Decision making and planning for autonomous UAS missions has taken many forms including Dubins path [9], optimal control [45], search [16], or more typically, a combination of these. For example, in typical UAS missions such as surveillance or data collection, planners often take the approach of a shortest path problem [15, 29] where paths are planned in the most efficient way to cover an area using one or multiple agents. Other solutions such as [35] use combinatorics to solve more complex problems such as multiple goal UAS swarm routing. This technique is effective but is computationally costly and therefore computed offline. Other strategies have explored the novel use of Bézier curves in multi-agent simultaneous arrival and continuous monitoring [43]. To deal with uncertainty some researchers [25] have taken a different approach opting for the power of MDPs in these situations. This approach suffers from the explosion of the state space as dimensions are added. One solution is to reduce the action space in the problem by eliminating, or only considering specific scenarios as the mission progresses [25]. However, this strategy may not be generalizable to all MDPs and missions if there is a need for a complex action space. Similar to our strategy, some optimal control formulations have reduced state space size while retaining accuracy by using a variable resolution for discretization of the state space [21, 32, 33]. In our work we build on the work of those who use MDPs to

solve planning and decision making problems by applying ideas learned from MDP and optimal control research to mitigate the state space explosion and provide a strategy generalizable to any MDP which can be rediscretized.

In small UASs, Size, Weight, and Power (SWaP) is highly constrained, resulting in limited computational resources. The work reported in [15, 25, 29, 35] experience this problem and either develop tools to reduce the required resources or compute solutions offline where more resources are available. Our decision maker's/planner's purpose is online solution to problems modeled as MDPs while considering changes in environmental and mission uncertainty. Reducing computational requirements in order to produce **a** solution (as opposed to ideally optimal) is imperative to our success. Resource reduction typically comes in two ways for MDPs, reduction of the state space and/or more efficient solvers. To improve efficiency of MDP solvers some algorithms seek to approximate the optimal policy using dynamic programming, linear programming, and Monte Carlo simulation [4, 39, 42]. These solvers are powerful and can be used effectively in conjunction with shrinking state spaces to further reduce computational resources necessary. Our SMDPs are meant to **reduce state space**, thereby reducing the size of the MDP handed to any available solver. So for the purposes of this paper we ignore differences in solvers and consider reducing state space size as the primary way to reduce computational resources.

Research focused on reducing MDP state space often comes in one of two forms: removing states that aren't useful for the goal; or using a receding horizon to limit the problem to smaller more manageable chunks. For example, in [13, 26] the state space is reduced by finding and removing hidden structures within the problem that will not lead to a meaningful goal. Implementing a receding horizon is a common strategy in reducing state space size in a broad range of fields. Under this strategy, a sliding window is used to focus on a smaller sub-problem which is then solved.

The solutions to each sub-problem are typically concatenated to produce a piece-wise total solution [31]. This technique is used solve problems such as adaptive control for vision-based navigation of a UAS [18], model-predictive control of a UAS [11], and for UAS flight trajectory control in mixed integer linear programming [27]. In [7, 8], receding horizons are used to solve MDPs with large state spaces on small processors. The receding horizon strategy, though powerful, can sometimes fail to produce an accurate, or safe policy due to its limited scope or incomplete information [14, 40].

More recently, directly applicable to MDPs is a method to abstract them into a series of smaller "Abstract MDPs" [20]. This solution works well for MDPs with a clearly hierarchical goal structure lending itself to levels of abstraction. A related method, hierarchical constrained MDPs, aggregates states of the MDP to produce a less state-dense MDP that is more easily solved, but is suboptimal [17].

Considering a system as discrete can also be a strategy to reduce computation and state space size, and naturally lends itself to the discrete nature of computation. In control theory, multi-resolution discretization approaches for this purpose have been applied in economic models as well as vehicle control [21, 32, 33]. Both [21] and [33] use a flexible grid scheme to adjust resolution in dynamic programming applications. Munos and Moore view each state as a node on a decision tree, splitting each node under certain conditions [32]. For continuous Markov processes they propose a "general towards specific" approach in which the problem begins as a coarse representation, and is strategically refined according to splitting criteria based on singularities of the value function. They propose "corner value difference" and "value non-linearity" theorems which determine the conditions under which splitting can occur to create a higher-resolution state space while not overloading the computational resources. Our SMDP strategy is based on this "corner value difference" theorem – specifically, at each time step, we determine the discretization of the state space by assuming that

large value function differences between neighboring states imply important information is missing between them.

Chapter 3

Background

Here, to set the stage for our innovations, we introduce key considerations in MDP solvers and the quality of MDP solutions given problem scope and discretization of state and action spaces.

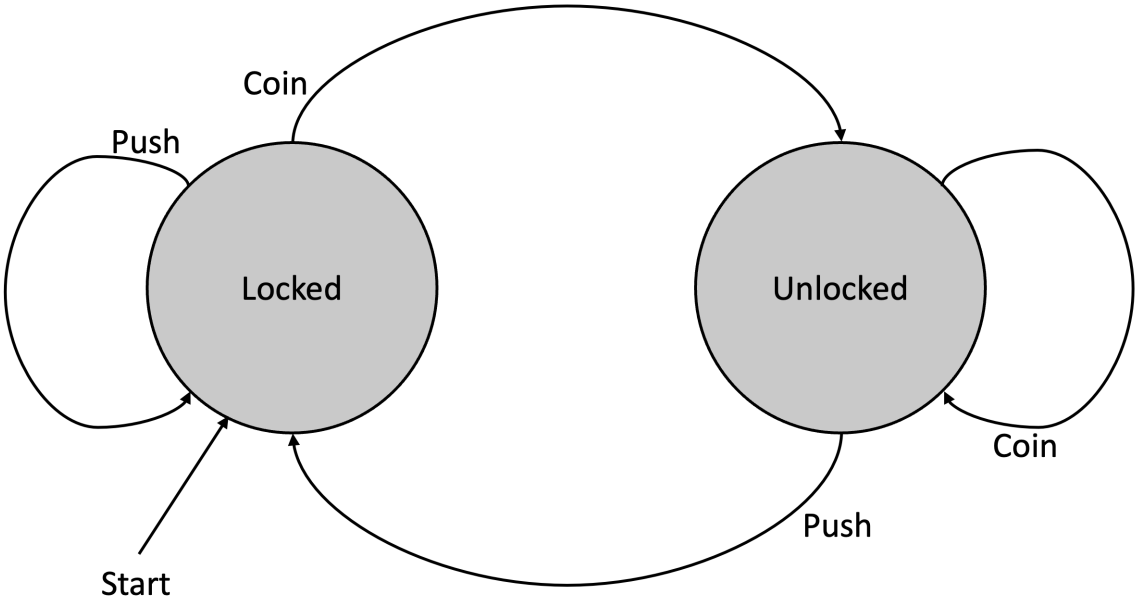


Figure 3.1: State-transition diagram of a coin operated turnstile.

Current State	Input	New State
Locked	Coin	Unlocked
	Push	Locked
Unlocked	Coin	Unlocked
	Push	Locked

Table 3.1: The state-transition table of a coin operated turnstile.

3.1 Automata Theory

First, it is imperative to have an understanding of automata theory. An automaton is a self-operating machine, which consists of a set of states and a transition function. The set of states are the different configurations the automaton can be in. The automaton exists in a specific state and can move to another state based on the transition function. The transition function dictates which state the automaton will go to given some input. A common example of an automaton is a coin-operated turnstile. The turnstile can either be locked or unlocked, starting in the locked state. Inserting a coin will cause the turnstile to unlock and let the patron through. Once the patron is through the turnstile, it should lock again. This can be represented in a state diagram in Figure 3.1. The set of states is either locked or unlocked. The transition function takes two inputs into account, depositing a coin (*coin*) and pushing the arm (*push*). While the turnstile is locked, depositing a coin will change its state to unlocked, and pushing will not change its state. While the turnstile is unlocked, depositing a coin will not change its state and pushing will change the state to a locked state. A state-transition table can be created, see Table 3.1

The coin operated turnstile is an example of a deterministic automaton, but a non-deterministic option is also possible. A non-deterministic automaton either have transitions that are not only determined by the current state and inputs (meaning the state you arrive in after taking an action is unknown) or an input is not required

at each state transition (meaning the state of the automaton can change without an input). Stochastic automaton are a subset of these non-deterministic automaton, where the transition function is defined by probabilities of getting from one state to another. This would mean that the transition function shows that there is a probability of going to a given state from a starting state and action. Adapting the turnstile example, the turnstile could be faulty, and 10% of the time it will remain in an unlocked state after being pushed. Moving on to an even more general case, each action could have an associated probability, even containing some states that will lead to failure of the system. Then, it becomes a 'game' of taking the action that has the best expected outcome for a given state. This is the foundation of Markov Decision Processes.

3.2 MDP Formulation

Markov Decision Processes contain a state space (S), an action space (A), transition probability matrix (P) and reward function (R). In the context of robotic automation, an MDP is created to mimic the operation of the robot within the environment. When creating the MDP, design decisions are made at every step and will impact the accuracy of the policy created. This section will be geared towards those with limited experience in creating Markov Decision Processes.

3.2.1 State and Action Space

The state space, S, is every state that the agent, the robot operating in the environment, can exist in. In discrete examples, creating this state space is trivial. Many applications of MDPs are situations where the environment is continuous and therefore, the environment must be discretized into discrete states. This discretization of

states is similar to the resolution of the state space, and the impact of changing the discretization is explored in Sections 3.3 and 3.4. States come in many different forms and can be abstracted outside of physical locations an agent exists in. Each type of these states we call *classes* of states, C. In a MDP created to model a drone delivering a payload, the location of the drone, l_n and the status of the payload, p_{status} can be used as classes of states implying $C = \{l_n, p_{status}\}$. Each state in the state space would include enumerations of both of these classes, leading to a state space

$$\begin{aligned} S &= l_n \times p_{status} \\ &= \{\langle l_1, p_{true}\rangle, \langle l_1, p_{false}\rangle, \langle l_2, p_{true}\rangle, \langle l_2, p_{false}\rangle, \dots\}. \end{aligned}$$

It is advantageous to have few classes of states, as each additional class adds another dimension of complexity to the MDP, because the total number of states will be the product of the number of states in each class (in the drone payload example, $S = n \times 2$, where n is the number of location states, and 2 is the number of payload states.)

The action space, A, consists of every action that can be taken from the agent. The actions are related to the capabilities of the agent and generally cannot be changed on the fly, unless there is some way to adapt the abilities of the agent during a mission. Actions are the driving factor that is able to allow the agent from moving from one state to another in an MDP.

3.2.2 Transition Probability Matrix

The transition probability matrix, $P(s'|a, s)$, is a matrix containing the probability of getting to every other state in the state space, $s' \in S$, based on a given starting state, s, and action, a. Many times, this matrix will initially be created with deterministic probabilities and randomness will be introduced later. For example, if the robot is in

an environment where it can go up, down, left, or right, initially each action would have a 100% chance of happening as intended. Then, to encourage a more robust solution, the transition probabilities may change to 80% chance to go in the intended direction and a 20% chance of going in a different direction. Additionally, a failure state could be used instead of going in a different direction. This state would be heavily disincentivized, resulting in more conservative policies. This failure state in the context of drones may mean that the drone crashes on the way to the location, resulting in a total mission failure.

In some applications of MDPs, the transition probabilities can be found from the probability or the rate or effectiveness of the actions. [19] explores finding the transition probabilities when modeling the progression of a disease based on medical studies that report different metrics of the effectiveness of treatments and [5] uses historical data to find the transition probabilities of land changing from one use to another. Both of these cases leverage historical data to create these matrices. This can be difficult to find for robotic systems, especially new ones, but can be found through testing as in [34], where the autonomous digging UAS was ran many times to obtain the success rate in different soil types, and this data was used to train an MDP to predict digging success.

3.2.3 Reward Function

The reward function, $R(s'|a, s)$, contains the 'reward' for getting to another state, s', from a starting state, s, and an action a. When an agent is operating in an environment, it aims to maximize the positive reward obtained. Therefore, the reward function impacts the actions an agent will take. The specific number chosen to be the positive or negative reward is arbitrary, but must be compared to the other values within the problem. A common strategy is to assign a constant negative reward to

all states that do not contain the goal, and assign a positive reward to the goal. This strategy is easy to implement and is the strategy used when creating the MDP used for the motivating example in Section 5.1. There can be benefits, however, to creating a reward function that is variable across the state space. In the context of a ground robot, the reward function may want to become more negative when the robot approaches a cliff or dangerous area. This would cause the robot to keep away from that area, if possible, to prevent accumulation of a larger negative reward.

Additionally, when using an on-board solver, the reward function can be updated as the agent moves through the environment and learns new information. A ground robot may not be aware of an obstacle but, if it can detect it, the reward function can be updated to disincentivize travel back to that area in the future. Designing a robust reward function is important to obtaining an accurate and intelligent policy.

3.2.4 Value Function and Policy Creation

After the MDP is created, the MDP must be solved, meaning a policy must be created that will map the states in the state space to actions from the action space. For every state in the state space, there must be a corresponding action. Generally, there are two main methods of obtaining the policy, value iteration and policy iteration.

Value iteration uses the Bellman equation to calculate a value (sometimes called utility) for every state in the state space. This is calculated using the reward function, $R(s)$, transition probability matrix, $P(s'|a, s)$, and a discount or "forgetting" factor, γ,

$$V_{i+1}(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|a, s) V_i(s'). \tag{3.1}$$

Where the algorithm will stop when $V_{i+1} = V_i$. This is guaranteed to converge, since

Equation (3.1) is a contraction mapping. Once converged, the value function and transition probability matrix can then be used to calculate the policy, $\pi(s)$,

$$\pi(s) = \underset{a \in A(s)}{\operatorname{argmax}} \sum_{s'} P(s'|a, s)V(s'). \tag{3.2}$$

Policy iteration, on the other hand, alternates between two steps to obtain the optimal policy. First, the policy set to be random and the value function is found using a simplified version of Equation (3.1),

$$V_{i+1}(s) = R(s) + \gamma \sum_{s'} P(s'|\pi_i(s), s)V_i(s'), \tag{3.3}$$

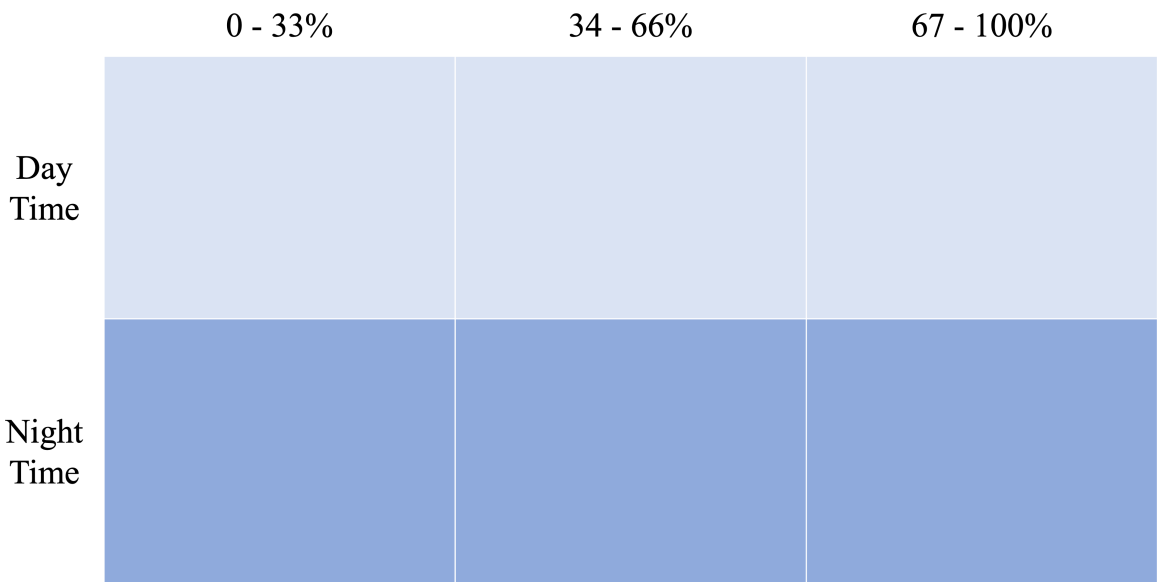
where the action for each iteration is chosen from the policy, not the maximum reward of any action. Then, the policy is updated, using Equation (3.2) and the newly created V_i to obtain a new policy. These are repeated until $\pi_{i+1} = \pi_i$. This is also guaranteed to converge, as it is also a contraction mapping.

Either of these methods will result in a globally optimal policy, where the action that maximizes cumulative expected reward is taken in each state. The largest difference in these two methods is that the value function may not be accurate for all states in policy iteration, which means that, for reasons explained in Section 4, value iteration is better for our application. In traditional applications of MDPs, this policy is found offline, where more computational resources are available, and then uploaded to the robot as a lookup table. It is assumed that the robot has knowledge of its current state, so the optimal action will be chosen from this table and executed at run time.

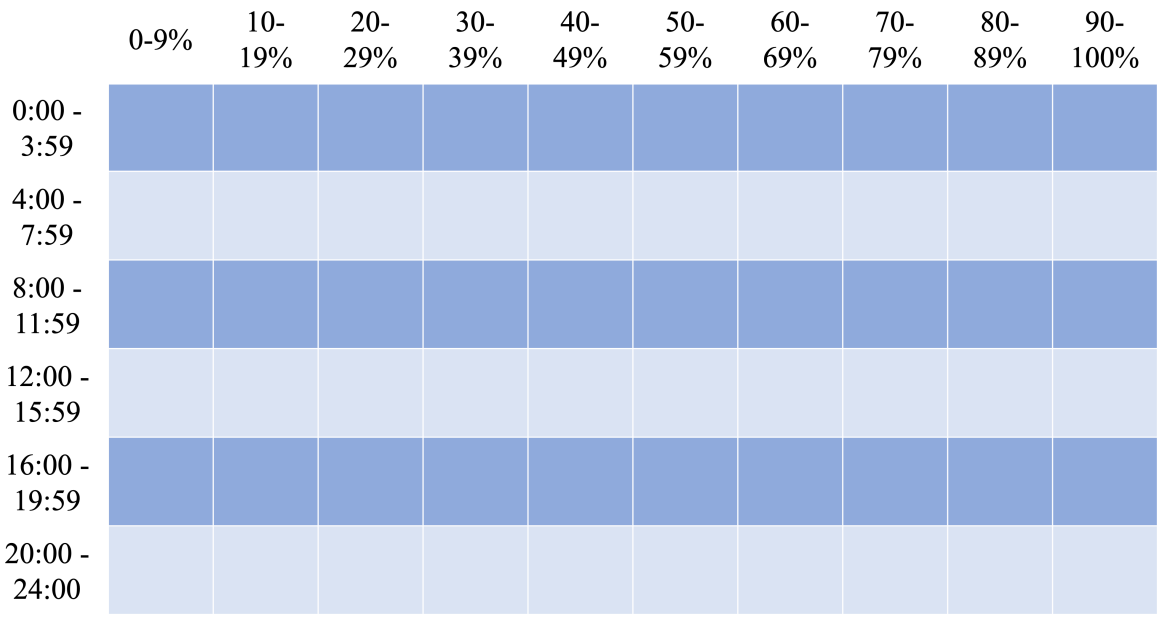
3.3 Discretization of Variables

Creating a complete, high-fidelity state space in real world problems can be difficult due to the large number of independent variables that are often represented as continuous functions. The discretization of the variables can be made more coarse or fine, depending on the requirements of the problem. Consider a possible MDP created to represent our test problem, a solar-supplemented UAS, which can take off, travel to, and land in a new location autonomously. The distance from the starting location to the goal location is too large for the UAS to travel to the goal without charging, so the UAS must chose when to stay and charge, or move to another location. Additionally, the UAS can only charge its battery during the day. In this example, three classes of states exist: landing spots, battery charge, and time of day. Battery charge and time of day are continuous variables that will need to be discretized in order for the problem to be solved, whereas landing spots are discrete. The possible discretizations of the continuous variables are explored in Figure 3.2. The coarse discretization consists of 3 charge states and 2 time states, while the fine discretization has 10 charge and 6 time states. The total size of the state space increases from 6 to 60, going from the coarse to fine discretization. In addition, the number of states created in charge and time would be multiplied by the number of landing locations, further increasing the state space.

In the continuous domain, the theoretical limiting factor of our discretization would be related to the sensing capabilities of the UAS. If we assume the UAS can land with 10 m accuracy, can sense battery charge changes within 1%, and useful time precision is 1 s, the number of states becomes very large. More precisely, if we confine the UAS to a 10 km $\times$ 10 km area, potential landing locations $l_n = 1{,}000{,}000$, battery states $b = 100$, and time states $d = 86{,}400$, yields total states $S = l_n \times b \times d =$



(a) Coarse discretization



(b) Fine discretization

Figure 3.2: Coarse discretization vs. fine discretization of battery charge and time of day state classes in the multi-flight, single-agent UAS example.

8,640,000,000,000. In addition to the large number of states created, the number of actions that can be taken by the UAS increases the size of the MDP. In this example, the UAS is able to travel to any of the landing locations, or it can stay and charge. This means that the number of actions, a, is equal to $l_n + 1$. Therefore, the transition probability matrix would contain $S^2 \times a$ elements. Solving this MDP on a SWaP constrained robot may be impossible, and when using more powerful, offline computing sources, would be time consuming at best.

One way to look at discretization is the boundaries created in the action space based on resolution of the states. When an MDP is solved, the policy maps sets of continuous states to actions. For example, Figure 3.3 shows a 1-D problem mapping battery charge to two actions: 'stay and charge' and 'move to another location'. The MDP policy creates two ranges, with the threshold to change the optimal action from one to another at $k\%$ charge. By discretizing the continuous domain, this threshold is necessarily moved to a state boundary, thereby discarding potentially useful information. While there are some specific discretizations especially close to the continuously optimal decision threshold, see $n = 10$, the only way to ensure the threshold is close to a discrete boundary is to increase the resolution of discretized states, and thereby recovering the lost information. However, increasing the resolution of the discretization, as discussed, can lead to large state spaces that become unsolvable. Additionally, this threshold becomes more complicated as more classes of variables are added. In Figure 3.4, both battery charge and time of day are included, with the same actions. Now the threshold creates areas, instead of ranges. In general, the theoretical continuous threshold has no constraints on the linearity or smoothness and there is no limit on the number of different areas created, even with only two actions. The hypervolume created by the theoretical continuous thresholds becomes more complicated as the number of classes increases, increasing the 2-D area into an

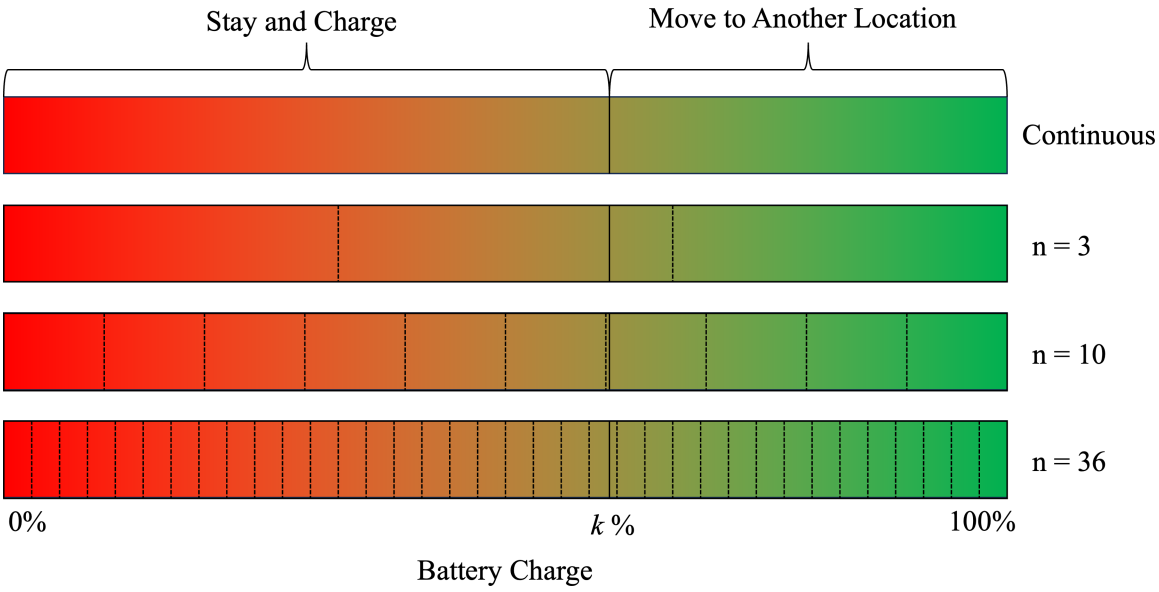


Figure 3.3: Battery charge shown continuously, then with different discretizations, n. The policy maps sets of continuous states onto the action space which necessarily fall on a state border in a particular discretization, thereby discarding useful information.

N-dimensional volume, where N is the number of distinct classes of variables. In the strategy we describe in this paper we exploit this phenomenon to recover information lost by the discretization process, and utilize it to improve the policy.

3.4 Impacts of Increasing Discretization

One way to reduce the state space size is to reduce the discretization of the state space. However, the impact on the quality of the solution to the MDP must be fully understood, and is problem dependent. To showcase this, the solar-supplemented multi-flight UAS MDP mentioned above was solved at different discretizations of charge and day states (see Section 5.1 for full problem details). In Figure 3.5 we showcase the time to solve the MDP for each combination of charge and day states between 2-20 states. As expected, the time to solve increases exponentially as the number of total states increases.

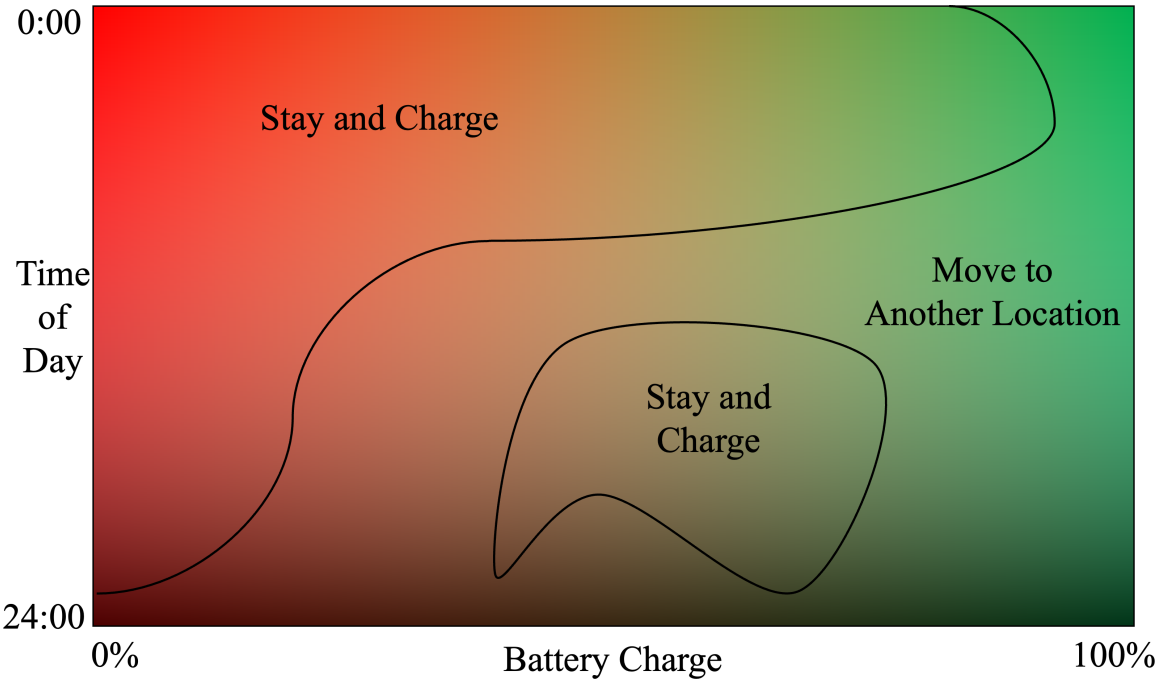


Figure 3.4: 2-D representation of continuous battery charge and time of day, with the threshold between actions 'stay and charge' and 'move to another location' creating multiple areas.

The Sliding MDP method described in this paper is an on-board solver which assumes every action chosen by the agent is successful. In contrast, a flat MDP is an offline solver, which will create a policy that contains the optimal action for each state in the state space, independent of the state of the agent. Therefore, to properly compare the flat MDP to an on-board solver, we create a 'deterministic policy' for the flat MDP. The deterministic policy contains every action a robot would take, as if the agent executes the policy in a deterministic manner. The total expected value for every state visited in the deterministic policy was then found from the value function. Figure 3.6 shows the relationship between increasing the discretization and the expected reward across the deterministic policy. There are some combinations of lower resolutions that give particularly good policies but, generally, increasing the number of charge states improves the reward of the deterministic policy. Increasing

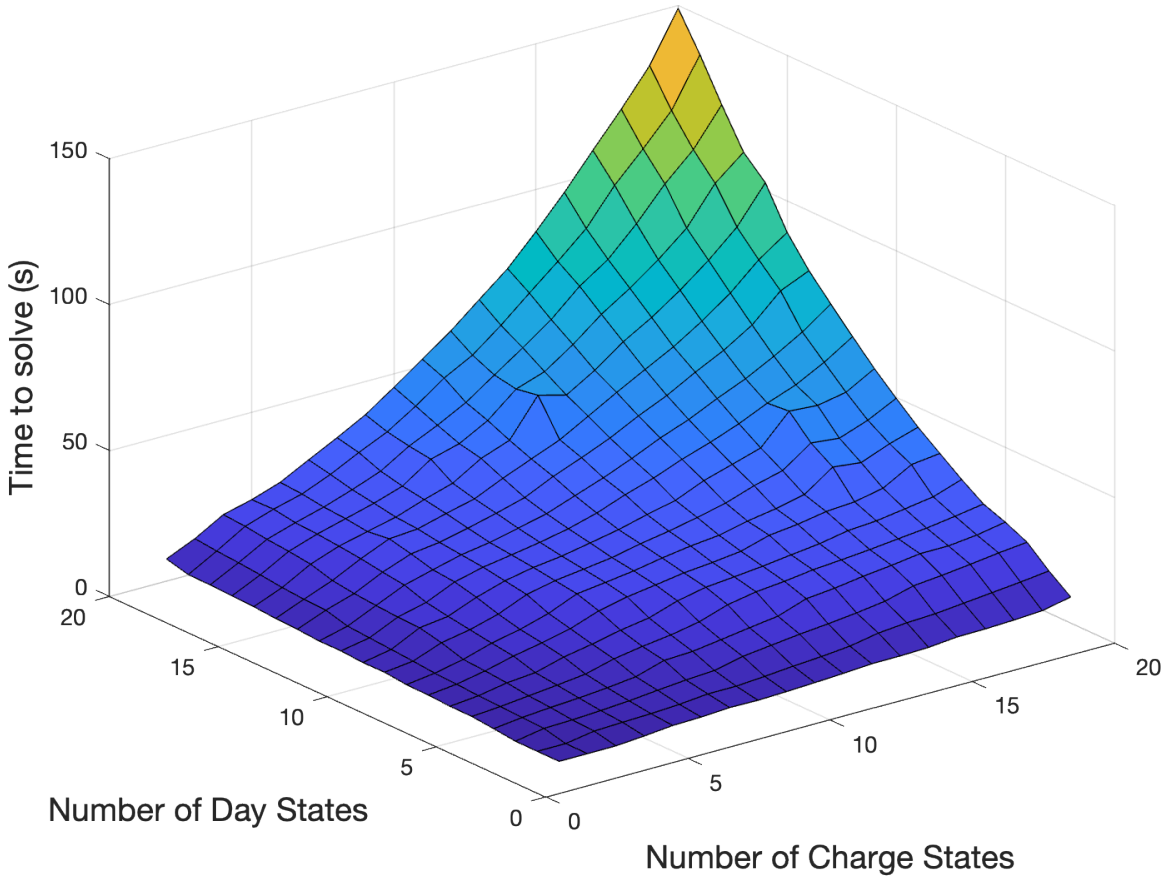


Figure 3.5: Effect of increasing the number of charge and day states on the time required for a policy to be calculated in the test problem.

the number of day states, however, has very little impact on the reward. This is because the additional information gained by increasing the resolution of the day states is less relevant to the drone, it only matters whether it can charge or not (whether it is daytime or nighttime). If this problem had a more detailed solar charging model, similar to [16], where the amount of energy gained from charging is dependent on the time of day, more day states could improve the value of the policy.

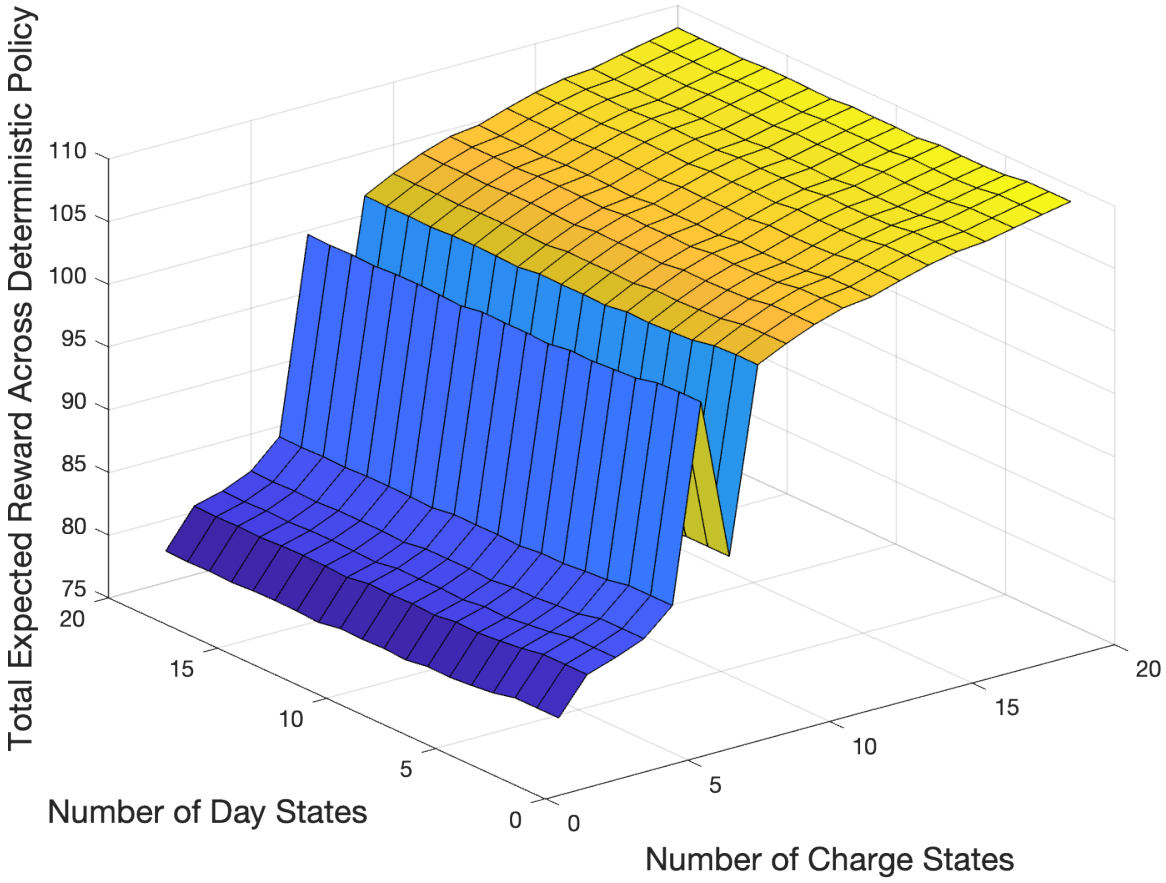


Figure 3.6: Total expected reward for the deterministic policy as the number of charge and day states increases.

Chapter 4

Sliding MDPs

Here we introduce and combine 3 key innovations to take advantage of the insights just presented and reduce the time and space complexity required to solve large planning and decision making problems in robotic systems. First, we use a receding resolution horizon, with the assumption that detail in the state space is less advantageous the farther the agent is from that state. We then solve the sub-MDP created within the receding horizon. Next, we use a value difference, or gradient approach to decide where increasing the discretization of the state space will reveal more information. Finally, we change the discretization of the state space, adding states into the MDP where necessary to create a more accurate policy. These innovations allow the agent to compute new policies dynamically while it moves through the state space, only increasing the discretization of the state space when there is more information to be gained. States outside the horizon are left in a coarse discretization and are considered unreachable. Each policy found is optimal within the horizon and, when combined with the policies from the horizons created as the robot moves through the state space, will result in a policy that we call recursively optimal across the path the robot takes through the state space.

4.1 Receding Resolution Horizon

In stochastic problems, receding horizon approaches have been successful when dealing with infinite horizons or state spaces that are too large to compute. This approach can also be used to decide where to increase the resolution of the discretization.

Deciding what should be included in the horizon varies case by case and is dependent on the size of the state space and computing power available. To calculate the horizon from a state, s_0, the states included would be such that transition probability $P(s|a, s_0) > B$ for any $a \in A(s)$, where B is the lowest "reasonable" transition probability. The states that fit this criteria are considered reasonable steps. Deciding what is considered "reasonable" can be difficult due to the many variables that can arise, and must be handled on a problem by problem basis. For example, one may consider any action with a transition probability that is greater than 50% "reasonable". This, however, can lead to a larger state space than appropriate for the computing power available and could cause the agent to take aggressive actions that are unlikely to lead to beneficial outcomes. On the other hand, if one wanted only high probability actions, choosing B to be 90% or higher, the receding horizon could be too small, and not take into account enough states for a high fidelity discretization. As B is increased, the number of states in the horizon will also increase.

4.2 Value Difference

The discretization of the state space within the receding horizon impacts the quality of the policy, and must be adjusted as part of the SMDP. The higher the discretization of the state space, the closer the performance will be to a high fidelity, optimal policy as there is more information available for the MDP to consider. The goal is to decide where there may be useful information not available at the current discretization and

increase the discretization in that area to include new information, without exploding the number of states in the state space.

To determine where useful information is likely to be we use a gradient based approach leveraging differences in value, $V(s)$, between neighboring states. Higher differences in value imply more information may be found with a higher resolution discretization in this area of the state space. This is similar to the *average corner-value difference* approach from [32]. In that work the space is split at the top $h\%$ of differences in value, whereas we split the space at differences that are above a splitting parameter, Z. This has the benefit of providing an easy-to-tune parameter that prevents the solver from infinitely refining the resolution, and hence blowing up the state space. This splitting process performed better in our tests than the uniform grid in the control problem posed in [32].

As an example, consider two neighboring states, s_1 and s_2, where $s_1 = (a_1, b_1)$ and $s_2 = (a_1, b_2)$. In this case, variable b is a coarsely discretized continuous variable. If the difference between the values of these states is large, $|V(s_1) - V(s_2)| > Z$, then we find that there is information that could be found from increasing the resolution and rediscretizing the b variable. A new state would be added, s_{new} and would have a value equal to the average of the surrounding values $V(s_{new}) = (V(s_1) + V(s_2))/2$.

This process will converge to a discretization of the state space that results in a value function with a constant gradient across the state space, meaning the difference between any two neighboring states will be at most the splitting parameter, Z. This will always converge, as long as $Z > 0$. This is because the difference between $V(s_1)$ and $V(s_2)$ will always be larger than the difference between $V(s_1)$ and $V(s_{new})$ or $V(s_2)$ and $V(s_{new})$. This leads to an algorithm that has an upper limit to the number of iterations that will be conducted, explored in Section 6.2. This is different than the process in [32], where there is no upper limit on iterations, creating finer

discretizations infinitely, possibly without gaining useful information.

After a constant gradient is achieved in the value function, a new transition probability matrix and reward matrix would then be created at the new discretization and the MDP would be solved again. The value iteration solver can be "warm started" by using the value function that is created from filling new states with the neighboring average values.

4.3 Example Problem

To provide intuition, we demonstrate how a small problem would be solved. The problem consists of a 1 dimensional physical domain in which the "UAS" can move either to the right or to the left as visualized in Figure 4.1. The UAS starts on the left side and must reach a goal state on the right side. The box contains all the states that are within the receding horizon, meaning the probability of getting to every state s' from the current state s is larger than the horizon boundary condition, $P(s'|a, s) > B$. When the value difference between its current state and the next state is greater than the splitting criteria $|V(s_1) - V(s_2)| > Z$ ($Z = 1.1$ in this example), the value splitting algorithm will split the continuous physical domain to add a new possible state s'_{new}. The policy would be recalculated, including the new state and the next action would be chosen based on this policy. After the UAS completes this action, it will "forget" the added state and continue on. Then, it would create a new receding horizon at the next state. This problem allows a very controlled environment for us to test the effectiveness of SMDPs. The problem splits in predictable places and has an easy to solve flat MDP for comparison.

The full state space, visualized in Figure 4.2, has a finer discretization across the entire state space. It contains all of the states seen in the coarse discretization and

all states that could be added if the state space was fully rediscretized, or added a new state between each existing state. We use this high resolution flat MDP as the best representation of the problem and hence the one with the highest quality policy.

In this example, an MDP would need to be created and solved containing all the states in the boxes. This means that the largest sub-MDP created in the sliding MDP would have 5 states, while the flat MDP would contain 7 states. This reduction in states will reduce the time to solve and spatial complexity exponentially. The fully expanded problem has a transition probability matrix with 98 elements ($7^2 \times 2$), while the SMDP representation has 50 elements ($5^2 \times 2$) in its largest transition probability matrix.

4.4 Algorithmic Implementation

Algorithm 1 demonstrates the implementation of SMDPs. Before running the algorithm, the coarsely discretized MDP is solved to obtain the value function, $V(s)$. The coarsely discretized MDP consists of the lowest reasonable resolution version of the MDP problem being solved given computational limitation. In general, similar to solving a flat MDP, starting with a higher resolution discretization will lead to higher quality policies. The state space, S; the transition probability matrix, P; and the action space, A, from the coarsely discretized MDP are also included as inputs. The algorithm will loop until the current state, s_c of the robot is the same as the goal state, s_g. First, the receding horizon is applied, constricting the state space to only reasonable states. Next, the difference between neighboring states is found and an additional state is added where the difference is large enough. The new state will be added to the state space and the process will repeat. After the state space is fully expanded, the goal state will be added to the state space if it is not already included,

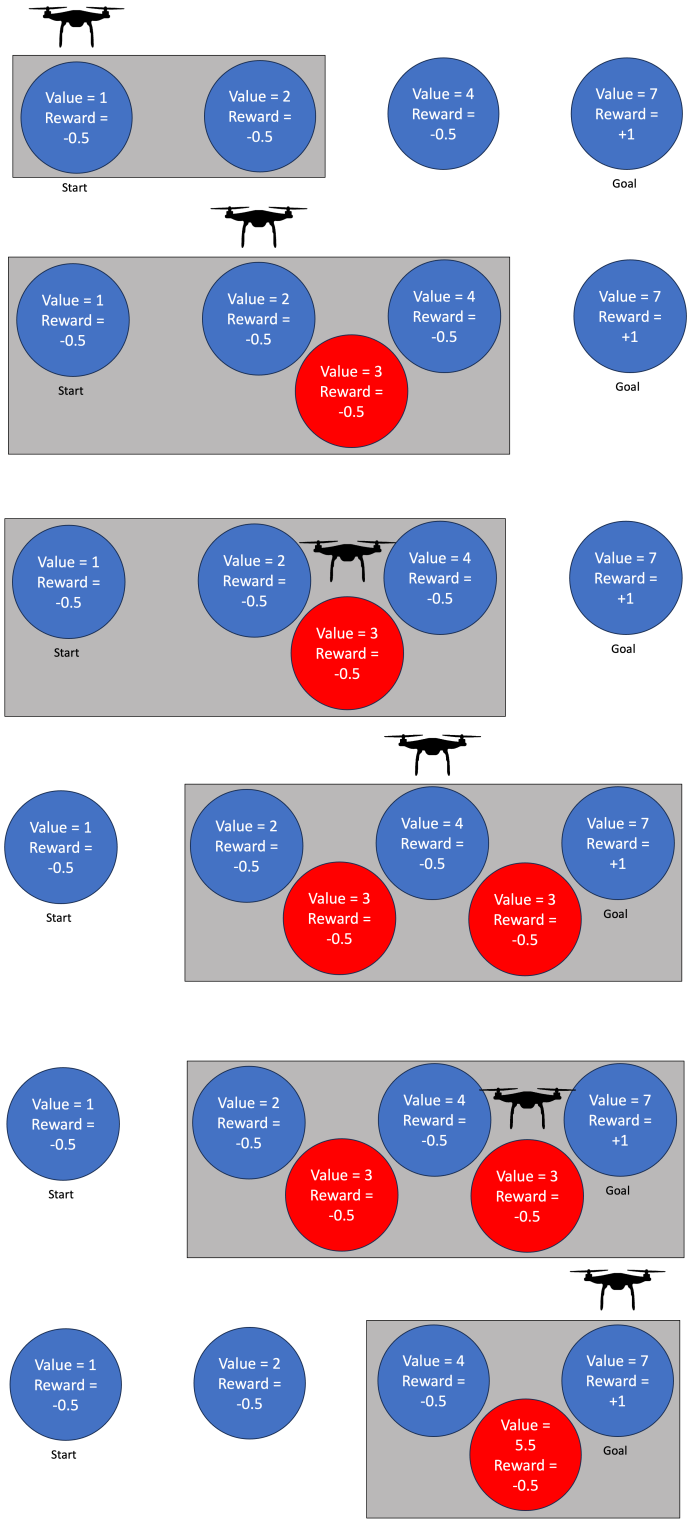


Figure 4.1: Visualization of a test problem using an SMDP to discretize physical space. ($Z = 1.1$)

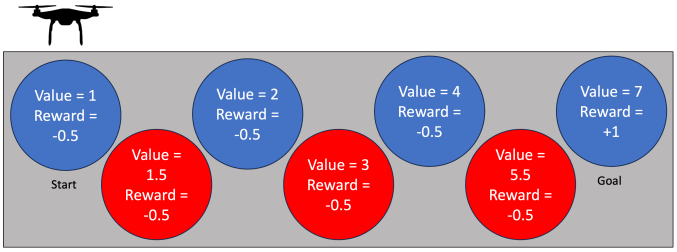


Figure 4.2: Full Discretization of the test problem, as would be used in a flat MDP.

and the transition probability matrix and reward matrices are created. Then, the MDP is solved using value iteration, with a 'warm-start' that will jump start the iteration process and results in a policy for receding horizon. The optimal action for the current state is found and executed, changing s_c to the state after the action is completed, s_a.

Algorithm 1 Sliding Markov Decision Process

Input: $V(s), S, P, A$
while $s_c \neg s_g$ **do**
  //Begin receding horizon
  forall $s \in S$ **do**
    if $P(s'|s, a \in A) > B$ **then**
      //make sub-state space only containing states in the horizon
      $S' = S' + s'$
    end
  end
  //Count all the states in S'
  $N =$ **count**(S')
  //Begin sliding resolution via value splitting
  while $addedStates \neg 0$ **do**
    $addedStates = 0$
    forall $s_i, s_{i+1} \in S'$ **do**
      if $|V(s_i) - V(s_{i+1})| > Z$ **then**
        //Value difference is large enough, add new state
        $N = N + 1$
        for $j = i + 1$ to N **do**
          //Move all the values forward to put new state between the states
          $V(s_j) = V(s_{j+1})$
        end
        //Change the value of the added state the to average of the surrounding states
        $V(s_{i+1}) = (V(s_i) + V(s_{i+2}))/2$
        $addedStates = addedStates + 1$
        $S' = S' + s_{i+1}$
      end
    end
  end
  if $s_g \notin S'$ **then**
    //if the goal state is not in S', add it
    $S' = S' + s_g$
  end
  //create reward and transition probability matrices
  $R =$ **function** rewardMatrix(S', A)
  $T =$ **function** transitionProbabilityMatrix(S', A)
  //Solve the MDP with value iteration
  $\pi =$ **function** valueIterationMDPSolver$(T, R, V(s))$
  //find the action for the current state
  $a =$ **function** findAction(s_c, π)
  //execute action
  $s_a =$ **function** executeAction(s_c, a)
  //set the current state to the state after executing the action
  $s_c = s_a$
end

Chapter 5

Results

5.1 Multi-flight Planning Problem

The motivating problem is a multi-flight, solar supplemented UAS, with a goal. On the way, the UAS will need to stop and charge to complete the mission. We built a simulation of this problem and solved it in totality with three different solving methods. The first is a flat MDP, which contains the entire state space with no variability and includes every state within its horizon. The second is a constant receding horizon MDP (CMDP) that changes the horizon as it moves through the space, without changing the discretization. The final is the Sliding MDP (SMDP), which has a receding horizon and changes the discretization according to the value difference approach. The flat MDP results in a policy for every state in the state space, while the CMDP and SMDP result in piece-wise policies along the path of travel for the UAS. The constant resolution MDPs (flat and CMDP) were both solved at different discretizations based on the assumption that the SMDP will fully rediscretize state space, or added states between all neighboring states, going from $2 \rightarrow 3 \rightarrow 5 \rightarrow 9 \rightarrow 17 \rightarrow 33 \ldots$ states. The lowest resolution state space consists of $b = 3$ charge states, (battery charge of 0-33%, 34-66% and 67-100%) and $d = 2$ day states ('day'

and 'night'). The flat and coarsely discretized MDPs are defined as follows:

$$
\begin{aligned}
\mathbf{MDP} &= \{S, P, R, A, \gamma\} \\
S &= l_n \times b \times d \\
P(s'|a, s) &= \text{landingProbability}(s, s', distance(s, s')) \\
R(s) &= \begin{cases} -0.04 & \text{if } s \neq s_g \\ +1 & \text{if } s = s_g \end{cases} \\
A(s) &= \begin{cases} s \rightarrow s' \\ \text{charge} \end{cases} \\
\gamma &= 0.95
\end{aligned}
\tag{5.1}
$$

The problem space consists of a 4×4 square grid with $l_n = 16$ possible landing spots, see section with solid border and numbered locations in Figure 5.4. The UAS starts in the top left corner, and the goal is in the bottom right corner. The probability of going from one location to another location is chosen using a normal cumulative distribution function, based on the distance between two locations and the charge of the battery. If the UAS has full battery charge, the probability is 50% to get to a neighboring location and 50% to stay in its current location. Moving two spaces has a probability of 15.8% and a 84.2% of staying in its current location, decreasing as the destination location is moved farther away. There is a negative reward, or cost, for every action taken to a non-goal state, and a positive reward for taking an action to the goal state. The UAS is allowed to go to any location, or stay in place and charge. The discount factor, γ in Equation (3.1), was set to 0.95. Finally, we assume that the system is fully observable, that is, the UAS has full knowledge of its current battery charge, the time of day and location within the grid.

5.2 Implementation

The SMDP algorithm was implemented in MATLAB on a 2021 Macbook Pro, with 32 GB of RAM and a M1 Max CPU. MATLAB limits the size of arrays to prevent running out of memory[1]. Rediscretizing the MDP to very high resolutions creates transition probability matrices that are too large for MATLAB to store (5 full rediscretizations with 65 charge states and 33 day states creates a transition probability matrix that is 149.2 GB) so, a policy cannot be found. This means a flat MDP cannot be compared against SMDPs for higher resolutions. Notably, the SMDP method creates much smaller incremental MDPs, which allows us to solve MDPs past this upper limit of resolution.

5.3 Metrics

The effectiveness of the algorithm was judged on three factors: spatial complexity, time to obtain policy and quality of the policy. When comparing these factors, there is not a definitive one-to-one comparison between the on-board solvers (CMDP and SMDP) and the flat MDP. This is because the on-board solvers do not result in a policy for every state in the state space, it will only create a policy within each horizon created on the path to the goal. Solving the flat MDP, however, results in a policy containing all actions for every state within the state space, independent of the movement of the robot. This also means the flat MDP has global knowledge of the entire state space, while the on-board solvers only contain knowledge of the states within the receding horizon.

[1] https://www.mathworks.com/help/matlab/matlab_prog/resolving-out-of-memory-errors.html

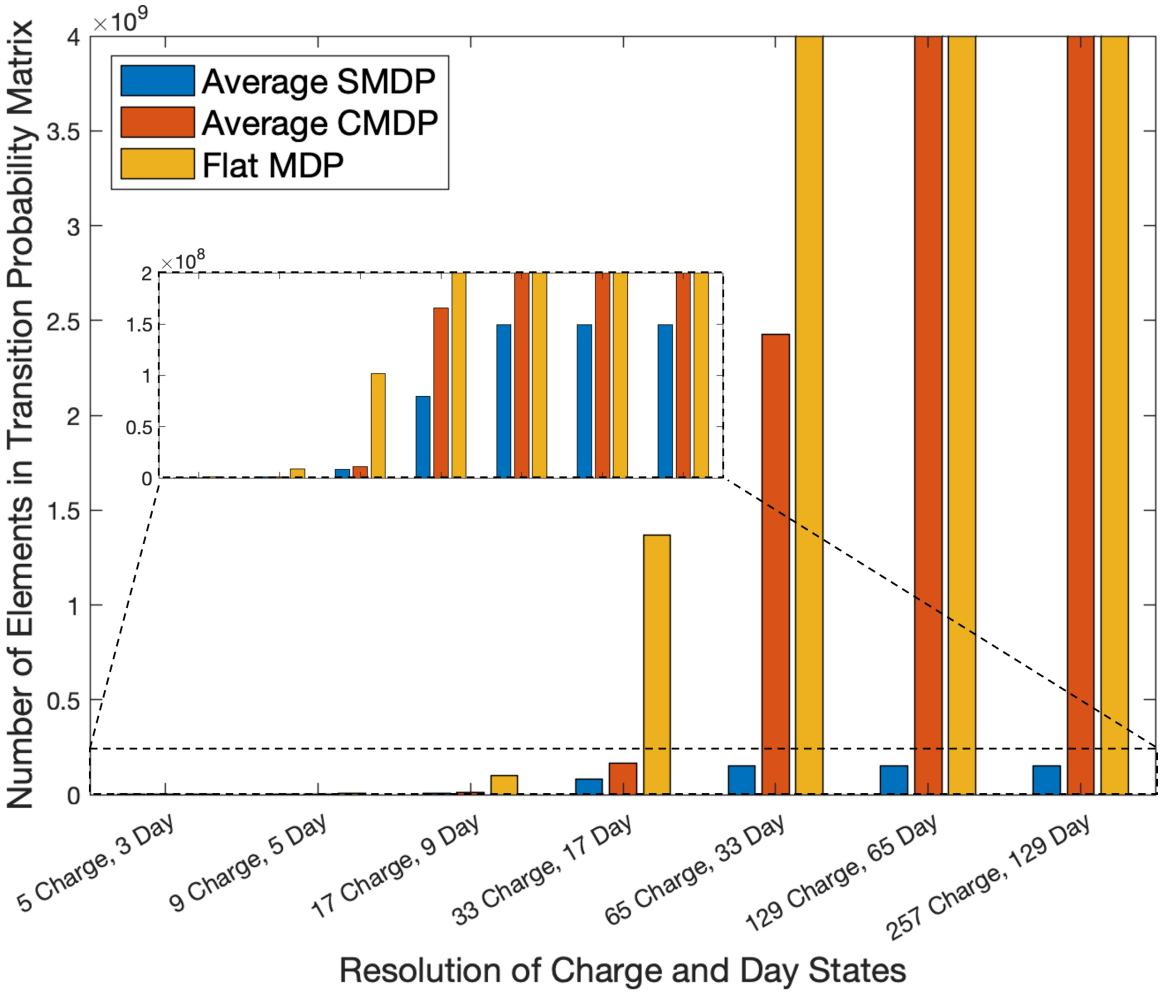


Figure 5.1: Number of elements in the transition probability matrix for a SMDP, CMDP and flat MDP of different resolutions.

5.4 Spatial Complexity

Spatial complexity is an important measure as the size of the MDP is heavily related to the amount of time and processing power needed to solve for an optimal policy. The size of the transition probability matrix for a flat MDP has complexity of $S^2 \times A$. This size was compared to the average size of the transition probability matrix for the CMDP and SMDP, as the number of states will change as the agent moves through the state space. Figure 5.1 shows this complexity measure for our example

problem. The graph cuts off all of the discretizations that have too many elements for MATLAB to store, 4×10^9 elements. The flat MDP is larger than both the CMDP and SMDP at every resolution and has an exponential increase. The CMDP also increases exponentially, but at a slower rate as the receding horizon will limit the size of the state space. The SMDP is equal in size to the CMDP at lower discretizations, but diverges at higher resolutions. This is what we expect from the SMDP algorithm, it converged on a discretization where the difference between all neighboring states in the value function is less than the splitting parameter.

5.5 Time

The total time to solve the flat MDP was compared to the time it would take the on-board CMDP and SMDP solvers to get to the goal assuming the current action takes less time to complete than the calculation of the next action.

Figure 5.2 shows the impact of increasing the resolution on the time to solve. The graph is shown with a logarithmic time axis. The solid lines show the experimental results and the dashed line shows a line of best fit, extrapolated to the final resolution possible with the SMDP algorithm. Both the CMDP and flat MDP show exponential increases, with no limit on the time it will take to get the optimal policy. The SMDP, however, initially increases exponentially, but flattens out as it converges.

The SMDP algorithm with the chosen splitting criteria will take 1375 s on average, assuming the algorithm is allowed to converge. It takes 8 actions to get to the goal in the deterministic policy, meaning each decision takes 171.875 s, assuming it moves instantly. Therefore, the robot would be able to operate optimally if it takes more than 171.875 s to complete each action.

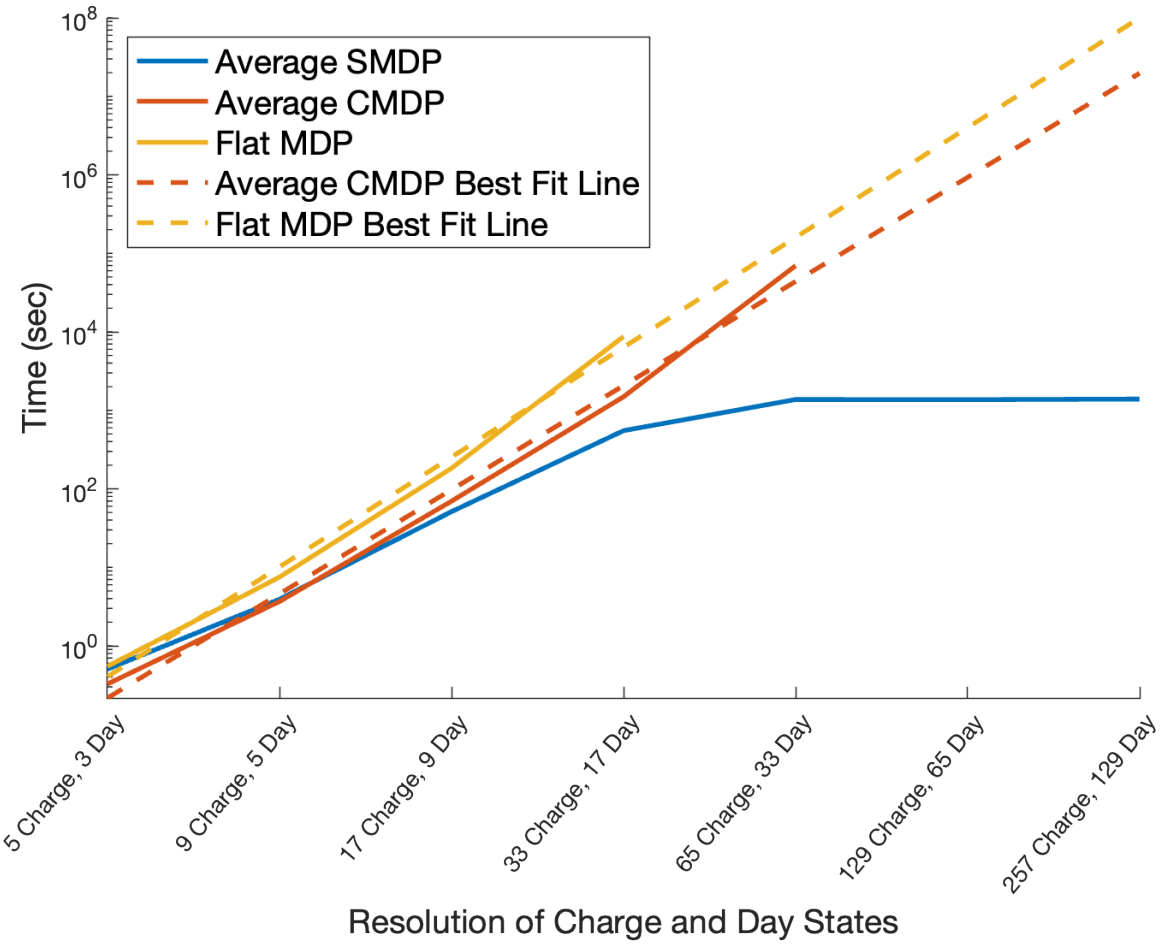


Figure 5.2: Time taken to solve a flat MDP or get the deterministic policy for a SMDP and CMDP.

5.6 Policy Quality

While "recursive optimality" might be the best an online solver can attain, it is critical to ensure that SMDP and CMDP solvers find policies as close to the globally optimal flat MDP policies as possible. To do this we use the expected reward, from the value function, to compare the accuracy of the policies. The sum of the expected reward across the policy, executed deterministically at different resolutions is shown in Figure 5.3. This shows that both on-board solvers have lower expected rewards across the deterministic policy. This is because the flat MDP has global knowledge of the entire

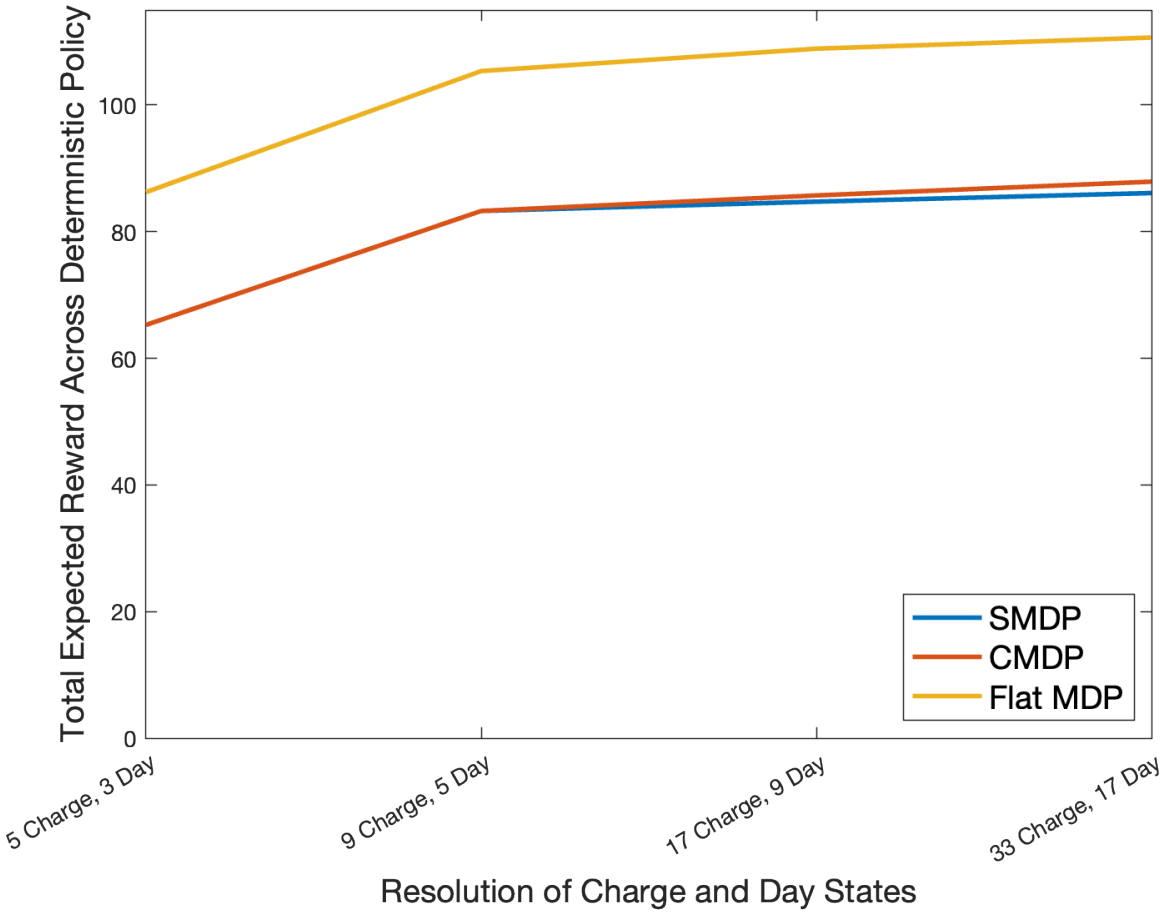


Figure 5.3: Expected total reward across the deterministic policy as the resolution increases for the SMDP, CMDP and flat MDP.

state space, but the on-board solvers only have knowledge within the horizon. The expected reward follows a similar pattern in all three MDP solvers however, with an initial increase, diminishing as the resolution gets more fine.

The actual policies themselves can also be compared across each method. Table 5.1 shows the deterministic policy for the SMDP, CMDP, and Flat methods. Each of the numbers means the UAS will move from it's current location to the new location, as labeled in Figure 5.4 and the robot will fill it's battery to full when completing the 'Charge' action if it is daytime. Each policy across the methods are identical, except at the highest resolution, where the flat MDP policy swaps actions 5 and 6.

This agreement shows that although the SMDP has a faster time to solve and smaller size complexity, the policy that is obtained is the same or very similar. Obtaining a recursively optimal policy is sufficient in most applications, including this problem; the robot will be able to arrive at the goal from any of the deterministic policies presented.

5.7 Adaptability

SMDPs and CMDPs additionally allow the MDP to be changed and the state space to be expanded dynamically. For example, in Figure 5.4, the original problem is shown in the numbered 4×4 grid with a solid border. The original goal is shown on space 16. As the UAS is on the way to the goal, there may be some previously unknown area that is made available to the UAS, shown with a dashed border. Because the MDP is being solved onboard, the new area can be included within the next horizon, and thus the new area could be taken into account when creating the next policy. A new goal could also be created in this new area, allowing more adaptable and resilient mission planning compared to flat MDPs, where the MDP would need to be re-solved for the entire state space. This process can be repeated as many times as necessary, allowing for an original state space of a very limited area to be explored and expanded, with a recursively optimal policy throughout the process.

Resolution	**5 Charge, 3 Day States**			**9 Charge, 5 Day States**			**17 Charge, 9 Day States**			**33 Charge, 17 Day States**		
Method	SMDP	CMDP	Flat	SMDP	CMDP	Flat	SMDP	CMDP	Flat	SMDP	CMDP	Flat
Action Step 1	2	2	2	2	2	2	2	2	2	2	2	2
2	3	3	3	3	3	3	3	3	3	3	3	3
3	4	4	4	Charge	Charge	Charge	Charge	Charge	Charge	Charge	Charge	Charge
4	Charge	Charge	Charge	4	4	4	4	4	4	4	4	4
5	8	8	8	8	8	8	8	8	8	8	8	Charge
6	12	12	12	Charge	Charge	Charge	Charge	Charge	Charge	Charge	Charge	8
7	16	16	16	12	12	12	12	12	12	12	12	12
8				16	16	16	16	16	16	16	16	16

Table 5.1: Deterministic policies using all three methods, across all resolutions. Numbers refer to moving to the spaces as labeled in Figure 5.4 and 'Charge' allows the UAS to charge to a full battery during the day time.

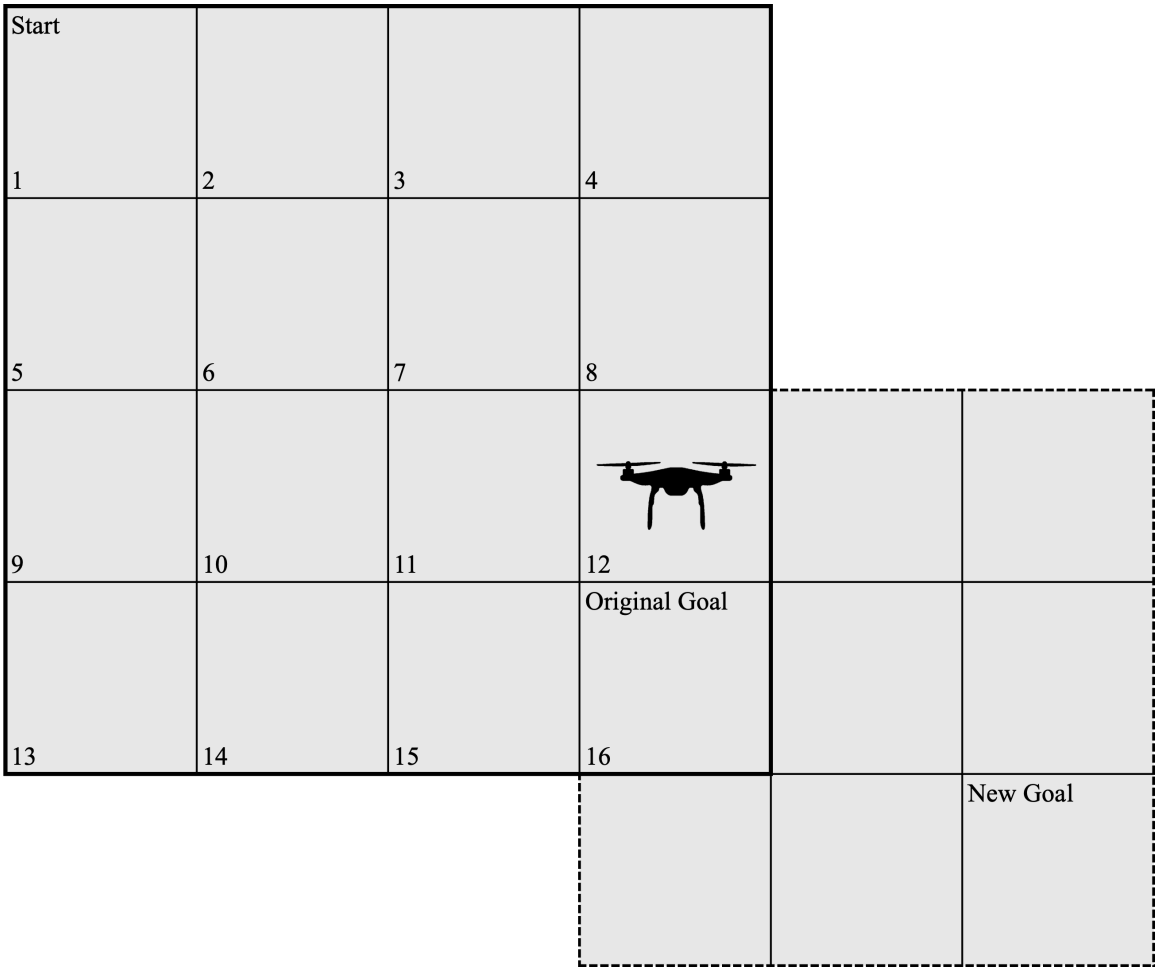


Figure 5.4: UAV operating in a state space that will be expanded as the agent moves. The solid border is the original state space, and the dotted border shows a previously unknown state space for the agent to operate in.

Chapter 6

Generalization

This algorithm can be widely applied, but many of the parameters chosen for this problem were tuned to our specific problem and application. Here we discuss how our approach can be generalized, focusing on the two tuned parameters, receding horizon limit, B, and the splitting parameter, Z.

6.1 Receding Horizon Limit

The receding horizon limit, B, dictates which states are within the current horizon. Most straightforwardly, this parameter is the probability below which transitions to a state are not considered, thereby eliminating that action/state from the horizon. This can be chosen based on insight into the problem space, acceptable risk levels, and limitations on computation.

The first method to choose this limit is a 'forward' method. This involves observing the state space and associated actions that are reasonable given intuition about the problem space, then averaging the transition probabilities that align with those state/action pairs. In the example space shown in Figure 5.4, a ground robot may be limited to moving one space in any direction making it a potentially good horizon. In this case the transition probability matrix would be analyzed and the probability as-

sociated with moving one location in any direction would be chosen. This is favorable because the decided limit is well informed by the domain and dynamical limits of the agent, however, extensive knowledge of the transition probability matrix is needed.

The next method is a 'backwards' method. This involves choosing a probability based on assessing the risk associated with the mission. In the example problem described in Section 5.1, because of flight, any state/action pair is possible (as opposed to a ground robot) but limited by the energy requirements to get there. In this case if the UAS needed to take few risks, a high limit would be chosen, filtering out low transition probabilities to discourage risky behavior. This may be more widely applicable, as a risk assessment may be easier to conduct than analyzing the transition probability matrix. This limited knowledge of the problem could lead to horizons that are too restricted, causing inaccurate policies, or a horizon that is too open, with a state space that is too large to solve onboard the UAS. As a result, computational limitations should be considered in tuning this parameter. In general, the horizon limit should be as low as possible, allowing as many states as possible, without causing the state space to become too large for the given computational capabilities.

6.2 Splitting Parameter

The difference in value between neighboring states, or value gradient, is the metric used split the state space. The goal of the value difference splitting algorithm is to create a value gradient that is less than the splitting parameter, Z, for the entire value function. Intuitively, if the gradient of the value function is constant no new information can be found by increasing the resolution of the state space and the resolution should not be increased further. The value function (see Equation (3.1)) determines the value of each state depending on the reward function, $R(s)$, of the problem. The

absolute number assigned to these rewards is arbitrary and only relevant within the context of the problem. Therefore, the value of the splitting parameter must be normalized within each problem. Some initial options for this parameter include the upper quartile, median, and lower quartile of the value differences, where the top $3/4$, $1/2$ and $1/4$ of value differences would be rediscretized, respectively. Any of these could be a good choice, depending on the computational capabilities and specifics of the problem.

Additionally, a helpful metric when choosing this parameter is N_{max}, the upper limit of states that could be created from the rediscretization process. This helps ensure the SMDP will remain small enough to accommodate the chosen computing hardware. N_{max} can be calculated from the value gradient of the initial coarse MDP. To start, we find the largest difference in value between neighboring states, ΔV_{max}. Because we split two states via arithmetic mean, ΔV_{max} will decrease by $1/2$ each iteration, meaning no new states will be created when

$$\frac{\Delta V_{max}}{2^i} < Z. \tag{6.1}$$

In this case, i is the number of iterations needed to converge to a value function with a constant gradient, which can be found by solving

$$i > \log_2 \left(\frac{\Delta V_{max}}{Z} \right). \tag{6.2}$$

Finally, this can be used to solve for N_{max},

$$N_{max} = 2^i N - (2^i - 1), \tag{6.3}$$

where N is the starting number of states. This can either be used to restrict the

maximum size of the state space (assuming a particular Z) or can be used in a 'backwards' method, where the maximum discretization is chosen and Z is calculated accordingly. For example, the number of charge states could be limited to 100, since the measurement of changes in the battery charge have precision of only 1%. This would mean solving Equation (6.3) for i, then solving Equation (6.1) for Z to create a lower limit for the splitting parameter. Increasing the splitting parameter will cause fewer states to be created.

Additionally, the splitting parameter may be poorly scaled compared to the value function of the specific application. If a splitting parameter is chosen that is larger than ΔV_{max}, the state space will be split at every opportunity, resulting in a higher resolution state space. This can be beneficial, if the state space is very small and a large number of states needs to be added to make an accurate policy but, it can also cause the state space to become too large to solve.

6.3 Partially Observable MDPs (POMDPs)

POMDPs are similar to a regular MDP, but direct knowledge of the current state of the agent is not known. Instead, the current state of the agent is informed by observations of the current state. The strength of a POMDP is that the observations do not need to be directly related to the states, while a regular MDP has state observations directly related to the state. Strong use-cases for POMDPs include medical diagnosis [22], where the subject's symptoms will play a role, but do not directly dictate the subject's diagnosis, or the allocation of resources to manage areas containing species that may be extinct [6], where one can either get rid of threats to the species, survey the area looking for members of the species, or surrender control and allocate resources to other species or areas.

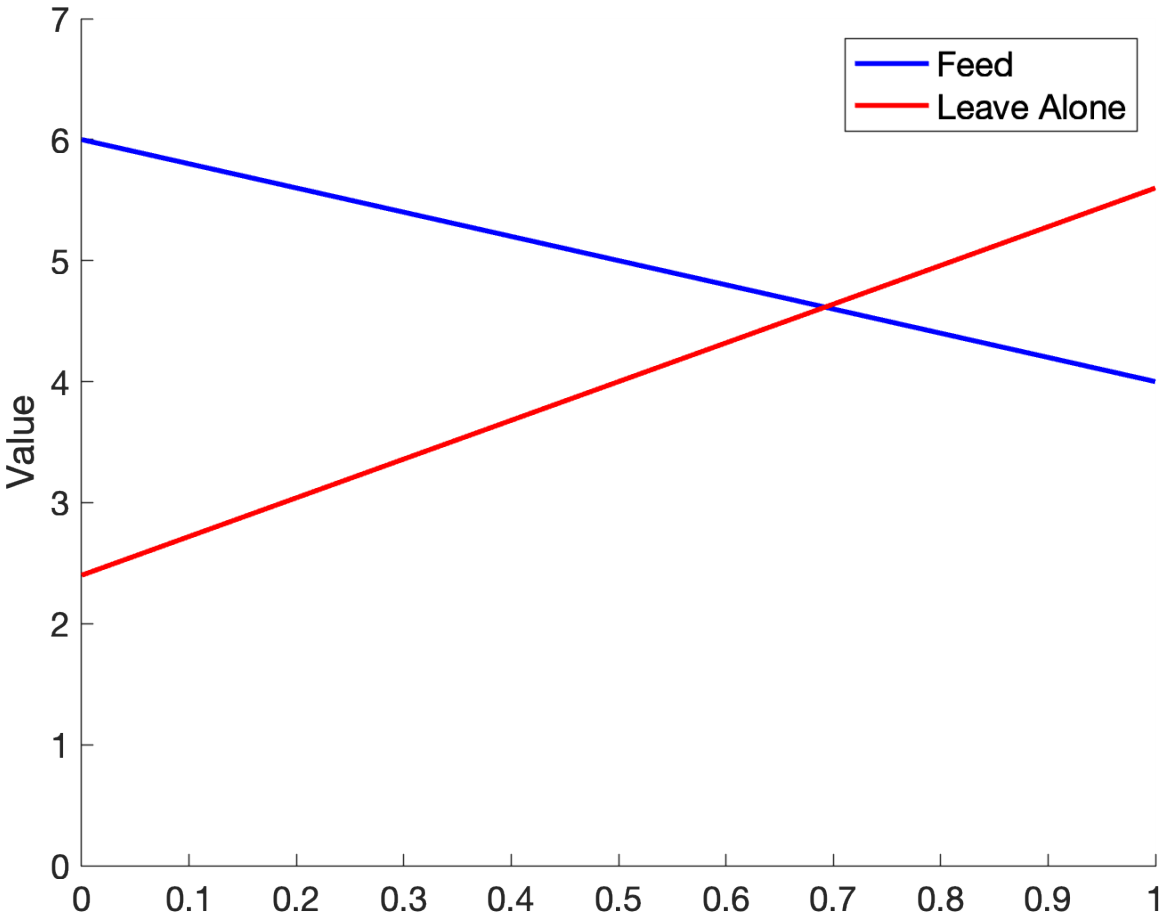


Figure 6.1: Value function of the crying baby problem modeled as a POMDP.

A simple example of a POMDP is the crying baby problem. In this example, the baby has two states, hungry or full, and the agent has two possible actions, feed or leave alone. The agent cannot directly find the state of the baby, it can only make observations of whether it is crying or not. This means that the current state is dependent on the observations made. This creates a 'belief space' which, in this example, would be a 1-D line between the hungry and full states, seen in Figure 6.1, where 0 means the baby is definitely hungry, and 1 means the baby is definitely full. Our 'belief state' is somewhere in between 0 and 1, informed by the observation function, which maps the observations made to a change in the probability that the baby is either hungry or full. This also means that the value function is different

for each action, and varies across the belief state. These lines, or hyperplanes in higher dimensions, form a piece-wise linear and convex set across the belief space, seen in Figure 6.1. The optimal action is simple to find once these lines are formed, simply decide what belief state you are in depending on the observations made and take the action with the highest associated reward. However, creating the vectors to represent this hyperplane created is very complicated and necessitates taking future actions into account, similar to the creation of the value function for regular MDPs (Equation 3.1).

A simple solution framework for POMDPs includes finding the most likely state, and solving for the associated regular MDP. The SMDP solving method would be easily adapted into this solving method. Additionally, there are point-based POMDP solvers [41], where value function is restricted to a finite subset of the belief state, only allowing local value updates. This is a similar method to SMDPs, however, using the resulting value function to decide where to rediscretize the state space may be difficult as it will be a set of hyperplanes, rather than one value. Finally, it is possible to learn the policy using reinforcement learning methods but, this does not create a model for us to use and thus, the SMDP algorithm is not helpful.

Chapter 7

Discussion

Our Sliding MDP approach shows many benefits over the traditional approach to solving flat MDPs. The spatial complexity and time to solve the SMDP do not grow exponentially, making it more reasonable to solve decision making problems. This is beneficial for an application on a UAS, which are generally SWaP-constrained systems with limited processing abilities. The SMDP could also be used to recalculate the policy at runtime, which could lead to more flexible, adaptable policies, able to dynamically change along with the environment. These benefits are achieved while still having a recursively optimal policy that is similar or identical to the flat MDP. In addition, many of the other techniques attempting to shrink the state space [13, 17, 20, 25, 26] could be applied in conjunction with SMDPs to get increasingly solvable MDPs. Factoring MDPs and alternate MDP solvers do not interfere with the SMDP approach and could further decrease the time to solve and size of the MDP with similarly accurate policies.

Another use-case for the SMDP approach would be to make large, complex MDPs solvable onboard constrained hardware by pruning unneeded states. This process would involve removing states that have a value difference smaller than some splitting parameter. This process would lead to a smaller number of states analyzed, meaning that it could be re-solved onboard SWaP-constrained hardware. This process still

requires the solving of the initial flat MDP to obtain the value function but this could be done offline and then uploaded to the SWaP-constrained vehicle so it could be re-solved onboard when conditions change in the environment.

SMDPs still face limitations when used on a low capability processor. Solving each sub-MDP may take longer than expected requiring optimizing the horizon, B, splitting parameter, Z, and planning lookahead of the agent. In extreme circumstances if not carefully designed, this can put systems at risk if policies are not computed on time. SMDPs, however, would still perform better in these situations than recalculating a flat MDP, where the response time could be much longer. SMDPs make this planning and re-planning possible at the expense of global optimality. We have found that high level mission planning problems are good candidates for applying SMDPs as corresponding flat MDPs can take especially long to solve when modeled well.

There is potential for SMDPs to be used for applications beyond single agent decision making. Heterogeneous swarms of SWaP-constrained vehicles operating in the same environment could obtain recursively optimal policies tailored for each vehicle, while still having the capability of re-planning if the environment changes.

Additionally, the ability to reconfigure the MDP on-the-fly allows feedback into the system. A specific application could be the creation of a soil moisture map over a large area. Initially, water content could be measured at a constant resolution across the area, then the reward function can be adapted to favor areas of interest. This moisture map could contain more information, concentrated on the areas of interest without requiring extensive measurement of the entire area.

Chapter 8

Conclusion

We have described a novel algorithm for solving complex decision-making and planning problems formulated as an MDP. This is done by reducing the state space to a local receding horizon and increasing the discretization of the states within the horizon. Then, a new sub-MDP is formed and solved at a higher resolution, resulting in a recursively optimal policy for an agent as it moves through the environment. The performance of the SMDP algorithm in our test situations and its ability to solve the complex problem in our motivating example is very encouraging. This, in addition with its ability to adapt to changing conditions, makes it ideally suited to increase autonomy and decision making on UASs.

In the future, SMDP's ability to incrementally solve MDPs can be leveraged to be updateable as new information from the environment is learned. If the agent must complete a task multiple times, information about how well the agent was able to complete the task the first time can be fed back into the system to adjust the transition probabilities and rewards in subsequent completions of the task. In addition, because the MDP can be changed on the fly, it could be possible to add new classes of states to the problem, for example, we may expect that the mission will be completed in less than a day, so the time of day class is not needed but, if suddenly the mission gets expanded to a multi-day mission, the model can be updated

to include the time of day class.

The goal of the mission can also be updated online, during execution. For example, if the target is moving, the agent would be able to adjust its path to head towards the target every time much like a traditional control reference. This strategy could be used in higher level non-flight planning problems as well. An example is the tasking of actions within a heterogeneous group of agents, allowing accurate policies to be computed for the states surrounding each agent, leveraging their individual strengths and obtaining an accurate policy for each agent.

Exploration on the limits of this method also should be conducted as well, specifically, application of this method to problem domains without a physical interpretation of the receding horizon could improve the robustness of this method, discovering methods to find receding horizon limits without relying on specific qualities of the classes within the state space. Additionally, there could be problems where there are some states need to be kept within the receding horizon, even if the agent is far away from that state. Finally, research into an adaptable receding horizon could be another path, where the horizon is larger when the discretization is coarse, and the horizon shrinks as the discretization is increased. This could be beneficial in keeping the number of states per horizon similar, while increasing the information that is included in the horizon.

Bibliography

[1] Douglas Aberdeen et al. Policy-gradient algorithms for partially observable markov decision processes. 2003.

[2] Oguzhan Alagoz, Heather Hsu, Andrew J Schaefer, and Mark S Roberts. Markov decision processes: a tool for sequential decision making under uncertainty. Medical Decision Making, 30(4):474–483, 2010.

[3] Richard Bellman. A markovian decision process. Journal of Mathematics and Mechanics, pages 679–684, 1957.

[4] Craig Boutilier, Raymond Reiter, and Bob Price. Symbolic dynamic programming for first-order mdps. In IJCAI, volume 1, pages 690–700, 2001.

[5] Ademola Braimoh and Paul Vlek. Land-cover dynamics in an urban area of ghana. Earth Interactions - EARTH INTERACT, 8, 01 2004.

[6] Iadine Chadès, Eve McDonald-Madden, Michael A McCarthy, Brendan Wintle, Matthew Linkie, and Hugh P Possingham. When to stop managing or surveying cryptic threatened species. Proc Natl Acad Sci U S A, 105(37):13936–13940, September 2008.

[7] Hyeong S Chang and Steven I Marcus. Approximate receding horizon approach for markov decision processes: Average award case. Technical report, MARYLAND UNIV COLLEGE PARK INST FOR SYSTEMS RESEARCH, 2002.

[8] Hyeong Soo Chang and Steven I Marcus. Two-person zero-sum markov games: receding horizon approach. IEEE Transactions on Automatic Control, 48(11):1951–1961, 2003.

[9] Hamidreza Chitsaz and Steven M LaValle. Time-optimal paths for a dubins airplane. In 2007 46th IEEE conference on decision and control, pages 2379–2384. IEEE, 2007.

[10] Judy G Chizek. Military transformation: intelligence, surveillance and reconnaissance. LIBRARY OF CONGRESS WASHINGTON DC CONGRESSIONAL RESEARCH SERVICE, 2003.

[11] Stefano Colautti and Henk Haverdings. Small scale rotorcraft uas flight control using mpc. In ICCAS 2010, pages 2529–2532. IEEE, 2010.

[12] Thomas Dean, Robert Givan, and Sonia Leach. Model reduction techniques for computing approximately optimal solutions for markov decision processes. In Proceedings of the Thirteenth conference on Uncertainty in artificial intelligence, pages 124–131. Morgan Kaufmann Publishers Inc., 1997.

[13] Thomas L Dean, Robert Givan, and Kee-Eung Kim. Solving stochastic planning problems with large state and action spaces. In AIPS, pages 102–110, 1998.

[14] Tao Ding, Mario Sznaier, and Octavia Camps. Receding horizon rank minimization based estimation with applications to visual tracking. In 2008 47th IEEE Conference on Decision and Control, pages 3446–3451. IEEE, 2008.

[15] Pavichaya Eaungpulswat and Herbert Werner. Area coverage algorithms for multiagent surveillance tasks. 2012.

[16] Ryan D Eubank, Justin M Bradley, and Ella M Atkins. Energy-aware multi-flight planning for an unattended seaplane: Flying fish. Journal of Aerospace Information Systems, pages 73–91, 2016.

[17] Seyedshams Feyzabadi and Stefano Carpin. Risk-aware path planning using hirerachical constrained markov decision processes. In 2014 IEEE International Conference on Automation Science and Engineering (CASE), pages 297–303. IEEE, 2014.

[18] Eric W Frew, Jack Langelaan, and Sungmoon Joo. Adaptive receding horizon control for vision-based navigation of small unmanned aircraft. In 2006 American Control Conference, pages 6–pp. IEEE, 2006.

[19] Risha Gidwani and Louise B. Russell. Estimating transition probabilities from published evidence: A tutorial for decision modelers. PharmacoEconomics, 38(11):1153–1164, August 2020.

[20] Nakul Gopalan, Michael L Littman, James MacGlashan, Shawn Squire, Stefanie Tellex, John Winder, Lawson LS Wong, et al. Planning with abstract markov decision processes. In Twenty-Seventh International Conference on Automated Planning and Scheduling, 2017.

[21] Lars Grüne and Willi Semmler. Using dynamic programming with adaptive grid scheme for optimal control problems in economics. Journal of Economic Dynamics and Control, 28(12):2427–2456, 2004.

[22] Milos Hauskrecht and Hamish Fraser. Planning treatment of ischemic heart disease with partially observable markov decision processes. Artificial Intelligence in Medicine, 18(3):221–244, 2000.

[23] Ronald A Howard. Dynamic programming and markov processes. 1960.

[24] Ashraful Islam, Adam L Houston, Ajay Shankar, and Carrick Detweiler. Design and evaluation of sensor housing for boundary layer profiling using multirotors. Sensors, 19(11):2481, 2019.

[25] Byeong-Min Jeong, Jung-Su Ha, and Han-Lim Choi. Mdp-based mission planning for multi-uav persistent surveillance. In 2014 14th International Conference on Control, Automation and Systems (ICCAS 2014), pages 831–834. IEEE, 2014.

[26] Andrey Kolobov, Daniel S Weld, et al. Discovering hidden structure in factored mdps. Artificial Intelligence, 189:19–47, 2012.

[27] Yoshiaki Kuwata and Jonathan How. Three dimensional receding horizon control for uavs. In AIAA Guidance, Navigation, and Control Conference and Exhibit, page 5144, 2004.

[28] Pierre Laroche, François Charpillet, and Rene Schott. Mobile robotics planning using abstract markov decision processes. In Proceedings 11th International Conference on Tools with Artificial Intelligence, pages 299–306. IEEE, 1999.

[29] Alex Lazinica. New developments in robotics automation and control. BoD–Books on Demand, 2008.

[30] Nicolas Meuleau, Milos Hauskrecht, Kee-Eung Kim, Leonid Peshkin, Leslie Pack Kaelbling, Thomas L Dean, and Craig Boutilier. Solving very large weakly coupled markov decision processes. In AAAI/IAAI, pages 165–172, 1998.

[31] Hanna Michalska and David Q Mayne. Robust receding horizon control of constrained nonlinear systems. IEEE transactions on automatic control, 38(11):1623–1633, 1993.

[32] Rémi Munos and Andrew Moore. Variable resolution discretization in optimal control. Machine learning, 49(2-3):291–323, 2002.

[33] Stephen G Nash. A multigrid approach to discretized optimization problems. Optimization Methods and Software, 14(1-2):99–116, 2000.

[34] Adam Plowcha, Yue Sun, Carrick Detweiler, and Justin Bradley. Predicting Digging Success for Unmanned Aircraft System Sensor Emplacement. In Jing Xiao, Torsten Kröger, and Oussama Khatib, editors, Proceedings of the 2018 International Symposium on Experimental Robotics, pages 153–164, Cham, 2020. Springer International Publishing.

[35] Adam J Pohl and Gary B Lamont. Multi-objective uav mission planning using evolutionary computation. In 2008 winter simulation conference, pages 1268–1279. IEEE, 2008.

[36] Pascal Poupart. Exploiting structure to efficiently solve large scale partially observable Markov decision processes. Citeseer, 2005.

[37] Nicholas Roy, Geoffrey Gordon, and Sebastian Thrun. Planning under uncertainty for reliable health care robotics. In Field and Service Robotics, pages 417–426. Springer, 2003.

[38] Stuart J Russell and Peter Norvig. Artificial intelligence a modern approach. London, 2010.

[39] Scott Sanner and Craig Boutilier. Approximate linear programming for first-order mdps. arXiv preprint arXiv:1207.1415, 2012.

[40] Tom Schouwenaars, Jonathan How, and Eric Feron. Receding horizon path planning with implicit safety guarantees. In Proceedings of the 2004 American control conference, volume 6, pages 5576–5581. IEEE, 2004.

[41] Guy Shani, Joelle Pineau, and Robert Kaplow. A survey of point-based POMDP solvers. Autonomous Agents and Multi-Agent Systems, 27(1):1–51, July 2013.

[42] David Silver and Joel Veness. Monte-carlo planning in large pomdps. In Advances in neural information processing systems, pages 2164–2172, 2010.

[43] Shawn Stephens, Satyanarayana G Manyam, David W Casbeer, Venanzio Cichella, and Donald L Kunz. Randomized continuous monitoring of a target by agents with turn radius constraints. In 2019 International Conference on Unmanned Aircraft Systems (ICUAS), pages 588–595. IEEE, 2019.

[44] Adam C Watts, Vincent G Ambrosia, and Everett A Hinkley. Unmanned aircraft systems in remote sensing and scientific research: Classification and considerations of use. Remote Sensing, 4(6):1671–1692, 2012.

[45] Michael D Zollars, Richard G Cobb, and David J Grymin. Simplex optimal control methods for urban environment path planning. In 2018 AIAA Information Systems-AIAA Infotech@ Aerospace, page 2259. 2018.