OPTIMAL CONTROL OF PERIMETER PATROL USING REINFORCEMENT LEARNING

A Thesis

by

ZACHARY WILLIAM WALTON

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2011

Major Subject: Mechanical Engineering

OPTIMAL CONTROL OF PERIMETER PATROL USING REINFORCEMENT LEARNING

A Thesis

by

ZACHARY WILLIAM WALTON

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2011

Major Subject: Mechanical Engineering

ABSTRACT

Optimal Control of Perimeter Patrol Using Reinforcement Learning. (May 2011)

Zachary William Walton, B.S, Texas A&M University

Chair of Advisory Committee: Dr. Darbha Swaroop

Unmanned Aerial Vehicles (UAVs) are being used more frequently in surveillance scenarios for both civilian and military applications. One such application addresses a UAV patrolling a perimeter, where certain stations can receive alerts at random intervals. Once the UAV arrives at an alert site it can take two actions:

1. Loiter and gain information about the site.
2. Move on around the perimeter.

The information that is gained is transmitted to an operator to allow him to classify the alert. The information is a function of the amount of time the UAV is at the alert site, also called the dwell time, and the maximum delay. The goal of the optimization is to classify the alert so as to maximize the expected discounted information gained by the UAV's actions at a station about an alert. This optimization problem can be readily solved using Dynamic Programming. Even though this approach generates feasible solutions, there are reasons to experiment with different approaches. A complication for Dynamic Programming arises when the perimeter patrol problem is expanded. This is that the number of states increases rapidly when one adds additional stations, nodes, or UAVs to the perimeter. This in effect greatly increases the computation time making the determination of the solution intractable. The following attempts to alleviate this problem by implementing a Reinforcement Learning technique to obtain the optimal solution, more specifically Q-Learning. Reinforcement Learning is a simulation-based version of Dynamic Programming and requires lesser information to compute sub-optimal solutions. The effectiveness of the policies generated using Reinforcement Learning for the perimeter patrol problem have been corroborated numerically in this thesis.

To My Family

ACKNOWLEDGMENTS

Thank you to the Air Force Research Lab for providing the opportunity to do this work. To Dr. Swaroop and Dr. Rathanam for always being there to answer any questions I had, minor or great. To my family who although we have been through our ups and downs we always love each other and find the good from our situations. Last but not least to the great state of Texas and the proud traditions of Texas A&M University.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

A. Perimeter Patrol Problem

Figure 1 displays an example of a base that needs to be protected.

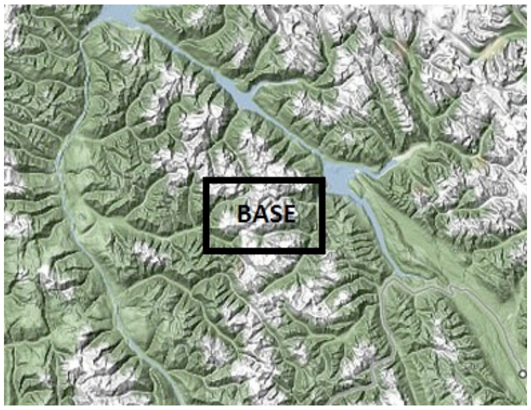


Fig. 1. Base Needing Protection

In order to do this a perimeter is placed around the base as shown in Figure 2.

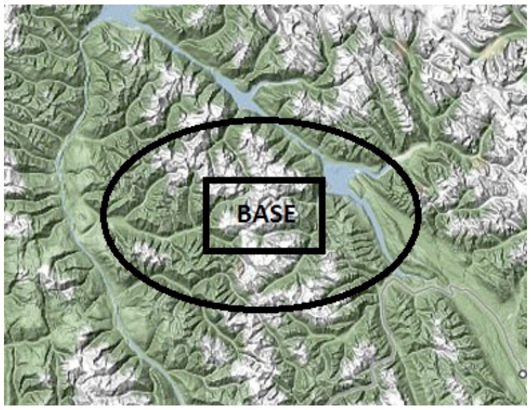


Fig. 2. Base Needing Protection with Imposed Perimeter

The journal model is *IEEE Transactions on Automatic Control*.

The goal is to protect the assets inside the perimeter from possible incursions. These incursions can be either a threat or a nuisance. A threat is something that wishes to do harm to the assets inside the perimeter. A nuisance is something that means to do no harm. Nuisances dominate the problem since most of the time the incursion will be an animal or something harmless crossing the perimeter rather than a real threat. Unattended Ground Sensors (UGS) are deployed on the perimeter. Meanwhile, a collection of UAVs patrol the perimeter. If there is an incursion, an alert is generated at that spot on the perimeter by a UGS. A UAV then proceeds to this spot on the perimeter to service the alert. The UAV has surveillance equipment and streams video from the alert site to a remotely located operator. The operator's job is to accurately classify if the incursion is a threat or a nuisance.

The operator is modeled as a non-ideal sensor or classifier. A sensor is characterized usually by missed detections and false alarms. A missed detection is when the operator classifies an incursion as a nuisance when it is actually a threat. A false alarm is when the operator classifies an incursion as a threat when it is actually a nuisance. Since nuisances dominate the problem it is hoped to reduce the false alarm rate through the use of UAVs in the perimeter patrol system. To judge the effectiveness of the operator as a classifier an operator error model is constructed. This operator error model consists of two probabilities. The first is the probability that the operator correctly classifies an incursion as a threat when it is actually a threat. The second is the probability that the operator correctly classifies an incursion as a nuisance when it actually is a nuisance. These probabilities are a function of the amount of time the UAV has spent at the alert site streaming video.

Ideally, the UAV would want to stay at the alert site indefinitely thus increasing the probability that the operator correctly classifies the incursion, thereby lowering the false alarm rate. However, while the UAV is at the alert site streaming video there could be alerts on the perimeter that have not yet been attended to. These incursions must also be classified within an adequate amount of time for the information gathered to be relevant. For this reason, a penalty associated with unattended alerts is put in place. The goal is to decide whether a UAV must loiter at an alert site for the next interval of time or traverse the perimeter to the next alert site. The decision should give the operator enough information from the alert site to classify the incursion and service unattended alerts in a reasonable amount of time. To simplify the problem we assume that the incursions can only occur at certain areas on the perimeter (which we also refer to as stations). The incursions will be modeled as a random process, more specific a Poisson process, with known average time between incursions.

B. Dynamic Programming Solution and Limitations

The perimeter patrol problem was solved using a Stochastic Dynamic Programming (SDP) methodology [1]. The result was an optimal policy for the UAV to take, meaning the action a UAV should take depending on a current state and a stochastic input. Though the SDP worked well, generated an optimal solution and passed a flight test, the limitations of this methodology still lays reason to experiment with other approaches. The most notable limitation is having to construct a mathematical model.

C. Reinforcement Learning

Reinforcement Learning is a simulation-based version of Dynamic Programming [2]. Reinforcement Learning does not seem to require the probability (state transition) matrix but instead computes them empirically. In this sense, it trades lack of knowledge about the model with additional off-line computational effort. This method also claims to make complex problems with larger state and action spaces tractable by finding the expectation of random variables empirically rather than analytically. Using the Dynamic Programming solution as a baseline, a Reinforcement Learning technique can be implemented and compared. The hope is that generalizations can be made to the perimeter patrol problem without having to construct a mathematical model. The specific Reinforcement Learning technique that was used in this thesis is Q-Learning.

CHAPTER II

PERIMETER PATROL PROBLEM

Many surveillance scenarios for UAVs face the problem of how best to allocate the UAVs' resources. These problems can sometime be solved using a stochastic optimization controller. A previous construction of a stochastic control optimization problem is detailed for the COUNTER scenario [3]. In this scenario a team of UAVs, one SAV and four MAVs, loiter over an urban area while streaming video is relayed to an operator. Once objects of interest are chosen by the operator to view more closely, the four MAVs are assigned a tour from a task assignment algorithm. The objects of interest are then inspected by the MAVs to allow the operator a chance to see any distinguishing features. Obviously the task of monitoring multiple MAV video streams while trying to discern any useable information is too much for any human operator to handle. Due to this a stochastic controller was created to choose when a MAV should revisit in an attempt to gain more information [4, 5]. This approach introduces an operator error model, also referred to as a confusion matrix. In sequence, the MAVs view the objects [6]. An information gain analysis is performed by the controller given the operator's observations. This analysis gives an expected reward for an MAV's revisit, therefore attempting to maximize the expected information the MAV can gain.

Stochastic control optimization has been greatly used in perimeter patrol problems [1]. Here a UAV and an Unattended Ground Sensor (UGS) are used. The UGS will generate an alarm if a disturbance is detected while the UAV patrols the perimeter. In the case of an alarm, the operator can then assign a UAV to assess the area. The goal is to find out the number of times the UAV should loiter before it can be reasonably assumed that a false alarm has occurred. One approach is to have the UAV fly over each UGS at a regular interval. The goal is to minimize the expected response time to an alarm by continuously being on patrol. An originally abandoned method was to consider the max response time Quality of Service (QoS). This idea was abandoned due to the intractability that arose when alert queues were expanded. The final approach was to put a weight on the expected information gained by a loitering UAV versus the expected wait time of the alerts in the queue. A key component to the problem is the alert rate. Nuisance trips can dominate the problem as they occur more often than actual incursions. Serving these nuisances quickly will act to preserve resources.

A. Problem Set-Up

The Perimeter Patrol Problem is set up as follows [1]. The perimeter is uniformly discretized into N segments. There are n number of UGS, or stations. These are the places on the perimeter where incursions can happen. Time is also discretized as $t = [0, 1, \ldots, k]$ with each discretization of time corresponding to F minutes. Alerts arrive according to a Poisson process with a known average time between any two incursions. There is a probability of one and a probability of zero incursions happening in time interval F. All incursions in the interval F are lumped into one big incursion so only one incursion can take place on the perimeter during a time step. The problem will be modeled as a Markov Decision Process.

1. States and Inputs

The states are described next. The first is $x(k)$ which is the position of the UAV at a discrete time k. $S_i(k)$ is the binary status of an alert at a station where $i = [1, 2, \ldots, n]$. This state takes a value of 1 if an alert has yet to be serviced at the

i^{th} station and 0 otherwise. The number of times the UAV has opted to loiter at an alert site is denoted $d(k)$. The delay time is simply $d(k) * F$. The amount of time in minutes an alert has been active at a station without being serviced is the delay time, $\tau_i(k)$. The outcome state is the length of the queue. This is denoted $q(k)$ and is the sum of the binary status of alerts at the stations at a given time k.

There are two inputs to the problem. The first is the control input $u(k)$. This is a binary decision for the UAV to either patrol or loiter. If the UAV opts to patrol it moves one time unit in its direction of travel and $u(k) = 0$. If loitering is chosen the UAV stays at a station for one time unit to inspect an alert site and $u(k) = 1$. The second input is the stochastic input $Y_i(k)$. This is the random variable that indicates whether an alert has arrived at the i^{th} station at the discrete time k.

The following are the state update equations.

$$\begin{aligned}
x(k+1) &= (x(k) + (1 - u(k)))modN, && (2.1)\\
S_i(k+1) &= (1 - \delta(x(k) - x_i)u(k))max\{Y_i(k), S_i(k)\},\ i = 1, 2, \ldots, n, && (2.2)\\
d(k+1) &= [d(k) + 1]u(k), && (2.3)\\
\tau_i(k+1) &= (\tau_i(k) + F)S_i^k,\ i = 1, 2, \ldots, n && (2.4)
\end{aligned}$$

2. Constraints

To keep the problem tractable the following constraints are imposed. The first states there is a maximum amount of loiters the UAV can opt to take at a station with an alert:

$$d(k) \leq d_{max}$$

The next constraint states that the decision to patrol or loiter is binary and $u(k)$ can only equal 1 when the UAV is at a station:

$$u(k) \leq \sum_{i=1}^{n} \delta(x(k) - x_i), u(k) \in 0, 1$$

The final constraint is a multiple part decision constraint:

$$u(k) \leq \sum_{i=1}^{n} \delta(x(k) - x_i) S_i(k) + d(k)$$

This constraint first states that the UAV must patrol if it is not at a station. If the UAV is at a station and its corresponding status is zero two things can happen. If the UAV has loitered at least one time it can choose to loiter or patrol. If the UAV has not loitered at least once it must patrol. A constraint that must be relaxed is to constrain the maximum delay time to alerts as this would make the problem intractable.

3. Reward Structure

Previous work has introduced an operator error model to judge how effective the operator is as a sensor/classifier [4, 5]. The operator error model can be classified by the following four possibilities as shown in Table I.

Table I. Operator Error Model Possibilities.

Operator's Classification	Actual Incursion Type	Description
Threat	Threat	Correct Classification
Threat	Nuisance	False Alarm
Nuisance	Threat	Missed Detection
Nuisance	Nuisance	Correct Classification

The model is a function of the number of loiters the UAV has taken and is presented below:

$$P_{TR}(d) = a + b(1 - e^{-\mu_1 d}), \tag{2.5}$$

$$P_{FTR}(d) = c + g(1 - e^{-\mu_2 d}) \tag{2.6}$$

Here P_{TR}, and P_{FTR} represent the probability that an alert was correctly characterized as a threat and a nuisance, respectively. d is the number of loiters the UAV has taken and a, b, $\mu 1$, c, g and $\mu 2$ characterize the performance of the operator as a sensor. In this formulation $a = c = 0.5$ corresponds to an operator who can not distinguish between a threat or a nuisance without seeing video footage of the alert. This means the operator is unbiased when he has not seen any streaming video from the UAV. Also, $b = g = 0.45$ and $\mu 1 = \mu 2 = 1$.

Using the operator error model the information gained by an operator is a function of p and $d(k)$. p is the *a priori* probability of target density and is set to $p = 0.01$. This corresponds to a $\frac{1}{100}$ probability of an incursion being an actual threat. Using P_{TR}, p and P_{FTR} the information gained by an operator is as follows:

$$\begin{aligned}
I = pP_{TR}log&\frac{P_{TR}}{pP_{TR} + (1-p)(1-P_{FTR})} + \\
p(1-P_{TR})log&\frac{1-P_{TR}}{p(1-P_{TR}) + (1-p)P_{FTR}} + \\
(1-p)(1-P_{FTR})log&\frac{1-P_{FTR}}{pP_{TR} + (1-p)(1-P_{FTR})} + \\
(1-p)P_{FTR}log&\frac{P_{FTR}}{p(1-P_{TR}) + (1-p)P_{FTR}}.
\end{aligned} \tag{2.7}$$

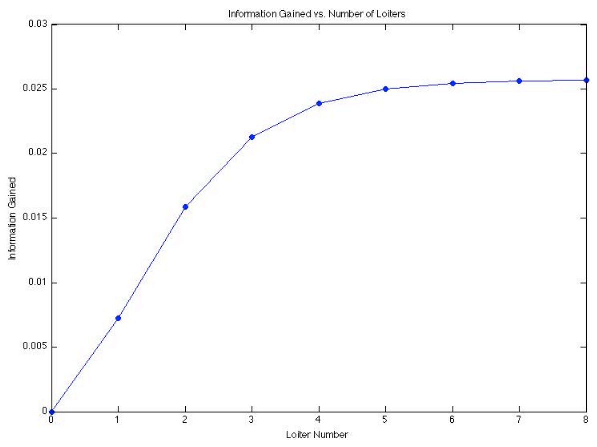
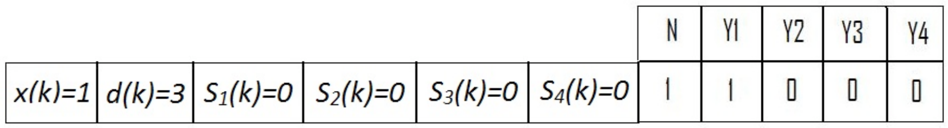


Fig. 3. Information Gained vs. Number of Loiters

The information gained as a function of number of loiters is shown in Figure 3. Though the amount of information the UAV gains increases with the amount of loiters the rate of increase in information gained decreases. This means that the amount of future information eventually levels off with a large number of loiters. If the UAV loiters a large number of times the probability that the operator will correctly classify the incursion will be very large thus reducing the false alarm rates. As stated previously, a penalty associated with unattended alerts will also be enforced.

B. Overall Goal

The objective function is to maximize the expected discounted information gained by the UAV's actions at a station about an alert. The goal is to find an optimal policy for the UAV to follow.



						N	Y1	Y2	Y3	Y4
$x(k)=1$	$d(k)=3$	$S_1(k)=0$	$S_2(k)=0$	$S_3(k)=0$	$S_4(k)=0$	1	1	0	0	0

Fig. 4. Example of a Working Policy

Figure 4 shows the real time implementation structure of the policy. This example is of a perimeter with four stations. It displays the decision the UAV should employ depending on the stochastic input. Here the UAV is at the first station and has already completed three loiters. There are no alerts in the queue, meaning the values for $S_1 - S_4$ are zero. If no alert comes in or an alert arrives at the first station the UAV will choose $u(k) = 1$ and continue to loiter. If an alert arrives at stations two, three or four the UAV will choose $u(k) = 0$ and patrol.

CHAPTER III

PROBLEM SOLUTION

A. Perimeter Specifics

To implement the Dynamic Programming methodology and Q-learning algorithm the perimeter problem must first be characterized . The perimeter is discretized into 15 segments with 4 stations. These stations are located at $x_{1-4} = [1, 4, 8, 12]$. A layout of the perimeter is shown in Figure 5.

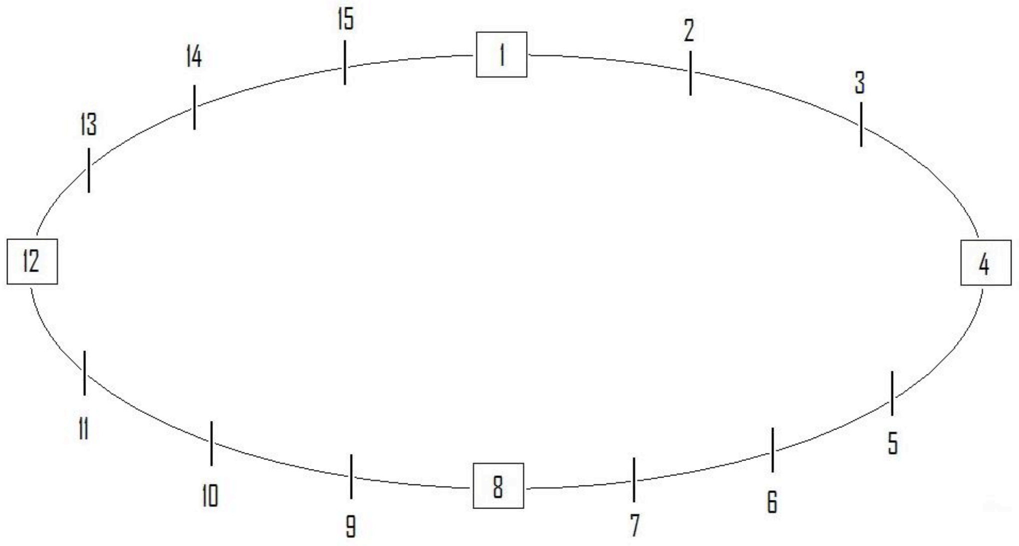


Fig. 5. Perimeter Diagram Showing 15 Discretized Nodes and Four Stations Located at $x_{1-4} = 1, 4, 8, 12$

The UAV traverses the perimeter in a clockwise direction. The variable F is set to 2, meaning each time-step is 2 minutes. The alert rate is $\alpha = \frac{1}{60}$, corresponding to an alert arriving every two perimeter orbits on average. All constraints presented earlier are adhered to. The maximum number of times a UAV can opt for loitering is set to five, d_{max}=5.

B. Dynamic Programming

The following is the objective function:

$$V(x(0), d(0), S(0)) = max\mathrm{E}_Y \left[\sum_{k}^{\infty} \lambda^k [\Delta\mathrm{I}(x(k), d(k), u(k)) - \sum_{i=1}^{n} \beta_i(x(k), d(k), u(k), S_i(k)) S_i(k)] \right] \tag{3.1}$$

$\Delta\mathrm{I}(x(k), d(k), u(k))$ is the information gained by taking action $u(k)$. The penalty associated with unattended alerts on the perimeter is $\beta_i(x(k), d(k), u(k), S_i(k))S_i(k)$, where β_i is a knob. Using a Poisson process $Y_i(k)$ is a stochastic input with a probability $e^{-\alpha F}$ that a value of zero is taken and a probability $(1 - e^{-\alpha F})$ that a non-zero value is taken. The expectation is taken over $Y_i(k), k \geq 0$. The discount factor is $\lambda \in [0, 1)$, thus allowing for convergence.

1. Dynamic Programming Implementation

Using the steady state dynamic programming equation, the conditional value function as a function of the initial state, $X(0) = (x(0), d(0), S(0))$, can be computed.

$$V(X_0|Y_0) = \max_{u \in U_{allowable}} \{\Delta\mathrm{I}(X_0, Y_0, u) + \lambda V(X_1(X_0, Y_0, u))\}, \tag{3.2}$$

Here $U_{allowable}(k)$ is the set of allowable control inputs $u(k)$ that satisfy the constraints. When u is chosen as the input at $k = 0$, $X_1(X_0, Y_0, u)$ is the state obtained at $k = 1$. The initial condition for the state is X_0 and Y_0 is the random input.

The following shows how to calculate the value function $V(X_0)$:

$$V(X_0) = e^{-\alpha F} V(X_0|Y = 0) + (1 - e^{-\alpha F}) \sum_{j=1}^{n} f_j V(X_0|Y_j(0) = 1)] \tag{3.3}$$

Here f_j is the probability that an alert has come in at the j^{th} station. The following "value iterations" are used to solve for the value function V.

$$V_k(X_0|Y_0) = \underset{u \in U_{allowable}}{max} \{\Delta \mathrm{I}(X_0, Y_0, u) + \lambda V_k(X_1(X_0, Y_0, u))\}, \tag{3.4}$$

$$V_{k+1}(X_0) = e^{-\alpha F} V_k(X_0|Y = 0) + (1 - e^{-\alpha F}) \sum_{j=1}^{n} f_j V_k(X_0|Y_j(0) = 1)]. \tag{3.5}$$

Once the value function is computed, the optimal decision can be computed as follows:

$$u^* = argmax_{u \in U_{allowable}} \{\Delta \mathrm{I}(X_0, Y_0, u) + \lambda V(X_1(X_0, Y_0, u))\} \tag{3.6}$$

u^* is then broken up into a decision table for real time implementation in the form of Figure 4. The Dynamic Programming method was performed in Matlab.

C. Q-learning

1. Q-Learning Overview

The specific Reinforcement Learning algorithm that will be implemented is Q-Learning. This algorithm finds an optimal policy by keeping track of two variables, Q and W, while the problem is simulated. The simulation continues until a maximum iteration limit is reached or the Q values converge. The Q value is the learned action-value function meaning the effect an action has on a state in the long run. A state's Q values change each time an action is used in that state. There are m number of Q values for each state, m being the number of actions. W is an incremental counter that keeps track of how many times a particular state-action has been visited. This variable along with the step size constant, A, is used to calculate the learning rate, denoted α_Q, which influences the rate of learning and allows for convergence. There are also m number of W values for each state. The mathematics of the algorithm will be discussed further in a later section. Discussed now will be the generalized Q-learning algorithm.

2. Generalized Q-Learning Algorithm

We will classify the state space as S and the action space as $A(l) = \{0, 1\}$. The following are the steps involved in the algorithm [7]:

```
1: For all $(l, u)$ where $l \in S$ and $u \in A(l) \rightarrow$ Set $Q(l, u)$ and $W(l, u)$ to 0.
2: Initialize $k$ and set $k_{max}$.
3: Set A, the step size constant, to a positive number less than 1.
4: Start the system simulation at any arbitrary state.
5: Let the current state be $i$ and select action $a$ with a probability of $\frac{1}{|A(i)|}$.
6: Simulate action $a$ and let the next state be $j$.
7: Let $r(i, a, j)$ be the immediate reward, determined by the simulator, earned in
   the transition to state $j$ from $i$ under the influence of action $a$.
8: Increment $W(i, a)$ by 1.
9: increment $k$ by 1.
10: Calculate $\alpha_Q = \frac{\mathrm{A}}{W(i,a)}$.
11: Update $Q(i, a)$ by the following,
    $Q(i, a) \leftarrow (1 - \alpha_Q)Q(i, a) + \alpha_Q[r(i, a, j) + \lambda \max_{b \in A(j)} Q(j, b)]$.
12: If $k < k_{max}$ set $i \leftarrow j$ and go to Step 5. Otherwise go to Step 13.
13: For each $l \in S$, select
    $d(l) \in \arg\max_{b \in A(l)} Q(l, b)$.
14: The policy generated by the algorithm is $\hat{d}$. Stop.
```

Algorithm 1: Steps in Q-Learning

CHAPTER IV

ADAPTING Q-LEARNING TO THE PERIMETER PATROL PROBLEM

To illustrate how the Q-learning algorithm will be adapted to the perimeter patrol problem one iteration of the algorithm will be discussed in detail. The simulator starts at a random state and progresses from there. This is done by randomly generating an ID number between one and the total number of states. For the characterized perimeter there are 400 states. A method is needed to find what state the corresponding ID number refers to. The following equations with $M(0) = 0$ do this [1].

if $d(k) = 0$:

$$ID = (M(0) - 1)2^{n-1}[2N + d_{max}n] + (x(0) - 1)2^{n} + dec2bin(Status(0)) + 1 \quad (4.1)$$

otherwise:

$$ID = (M(0) - 1)2^{n-1}[2N + d_{max}n] + N2^{n} + (d - 1)2^{n-1} + dec2bin(Status(0)) + 1 \quad (4.2)$$

The algorithm also requires an action to be taken at random. Sticking to the algorithm, there are only two possible actions so the procedure would be to essentially flip a coin and take the resulting action. Here heads is patrol and tails is loiter. This leads to trouble however because Q-learning requires all of the states to be visited. If the mentioned procedure is used the only way the algorithm would reach the states where the UAV has loitered up to d_{max} times is if the action for loitering was randomly chosen d_{max} times in a row. This event is very rare and poses a problem. The fix is to allow the action to loiter, if this is the action generated, to be carried out for multiple time steps in a row thus visiting these rare states. The new procedure would allow the random action to be either to patrol or loiter up to d_{max} time steps. The resulting action or sequence of actions are then performed in the simulator. This forces the simulation to visit states that otherwise would have taken far more computation time to reach. A check must however be performed to make sure the proposed random action or sequence of actions do not violate any of the constraints. An example of a possible scenario is displayed for clarity. The UAV is at a station and has already loitered once. A random number is then generated from zero to one. Table II shows what the action would be and for how many time steps it is to be carried out.

Table II. Random Action Generation Example for UAV at a Station where $d(k) = 1$.

Condition	Action	# Time-Steps
rand< 0.2	$u(k) = 1$	1
$0.2 \leq rand < 0.4$	$u(k) = 1$	2
$0.4 \leq rand < 0.6$	$u(k) = 1$	3
$0.6 \leq rand < 0.8$	$u(k) = 1$	4
rand ≥ 0.8	$u(k) = 0$	N/A

From here the state and action is given to the simulator. The simulator is comprised of three parts. The first part is updating the state. This is done using Equations 2.1-2.3 and the generated $u(k)$. The second part is to generate the stochastic input, or the alerts coming in at each time step. This is done using a Poisson process with a known average time between alerts of $\frac{1}{60}$ as stated earlier. The final item in the simulator is the reward function which is displayed below [1]:

$$R = \Delta\mathrm{I} - \beta q(k) \tag{4.3}$$

Again ΔI is the information gained by the UAV loitering, $q(k)$ is the queue length and β is a knob used to give weight to the alerts not yet serviced.

Upon each subsequent state being visited through the chosen actions the corresponding Q and W values are updated accordingly. Since there are two control inputs and five stochastic inputs there are ten Q values and ten W values for each state. Once the actions have been simulated the resulting state is set to the current state and the procedure continues at choosing a random action. The simulator is set to run for thirty time steps, or two full orbits. At the end of the thirty time steps one iteration of the algorithm has passed and the algorithm starts from generating a random state. The process continues until k_{max} is reached. If by the end of the algorithm there are states that have not been visited then some Q's will have no value. For these occurrences a random decision will be chosen.

The total number of time steps in the simulator will be denoted by N_S and the state of the system will be $X(k)$. A step by step procedure for this implementation is displayed below:

The algorithm was implemented in MATLAB.

```
1: Set A, $k_{max}$,$\lambda$,$N_S$ .
2: Initialize Perimeter Patrol Parameters
3: For all $(l, u, y)$ where $l \in S$, $u \in A(l)$ and $y \in Y(l) \rightarrow$ Set $Q(l, u, y)$ and $W(l, u, y)$
   to 0.
4: for $k = 1$ to $k_{max}$ do
5:    Generate random ID
6:    Use equation 4.1-4.2 to obtain $X(k)$ from ID.
7:    for $i = 1$ to $N_S$ do
8:       Generate random number from 0 to 1.
9:       Determine sequence of actions, $A_S$.
10:      if Sequence of actions violate constraints then
11:         Go to Step 8.
12:      end if
13:      for $j = 1$ to length($A_S$) do
14:         Determine $Y_i(k)$
15:         Obatain $X(k+1)$ using equations 2.1-2.3.
16:         Obtain reward for progressing to $X(k+1)$ using 4.3.
17:         Increment $W(X(k), u(k), Y(k))$ by 1
18:         Calculate $\alpha_Q = \frac{A}{W(X(k),u(k),Y(k))}$
19:         Update $Q(X(k), u(k), Y(k))$ by the following,
            $Q(X(k), u(k), Y(k)) \leftarrow (1 - \alpha_Q)Q(X(k), u(k), Y(k)) + \alpha_Q[r(X(k), u(k), X(k+1)) + \lambda \max_{u \in A(j)} Q(X(k+1), u(k), Y(k))]$.
20:         Set $X(k) = X(k+1)$.
21:      end for
22:   end for
23: end for
24: For each $l \in S$, select
    $d(l) \in arg\max_{u \in A(l)} Q(l, u, y)$.
25: The policy generated by the algorithm is $\hat{d}$. Stop.
```

Algorithm 2: Adapted Q-Learning Algorithm for Perimeter Patrol Problem

CHAPTER V

RESULTS

Once the policy was solved for using Q-Learning it was tested against the policy obtained by Dynamic Programming. This was done by generating multiple policies in Q-Learning and simulating them alongside the policy from Dynamic Programming. This way the polices will have the same alerts coming in and create an ideal comparison. The simulator was run 1000 times. The discount factor for both Q-Learning and Dynamic Programming was set to $\lambda = 0.9$. The number of sequential loiters the UAV opted to take along with the maximum delays for each simulator iteration were then plotted in a histogram fashion. This will give a metric to how fast the UAV was able to service the alerts as well as how much information was gained.

A. Varying Step Size Constant (A)

The first set of policies were generated with different step size constants. The variable k_{max} was set to 150000. The variable A was set to four different values 0.1, 0.3, 0.6 and 0.9.

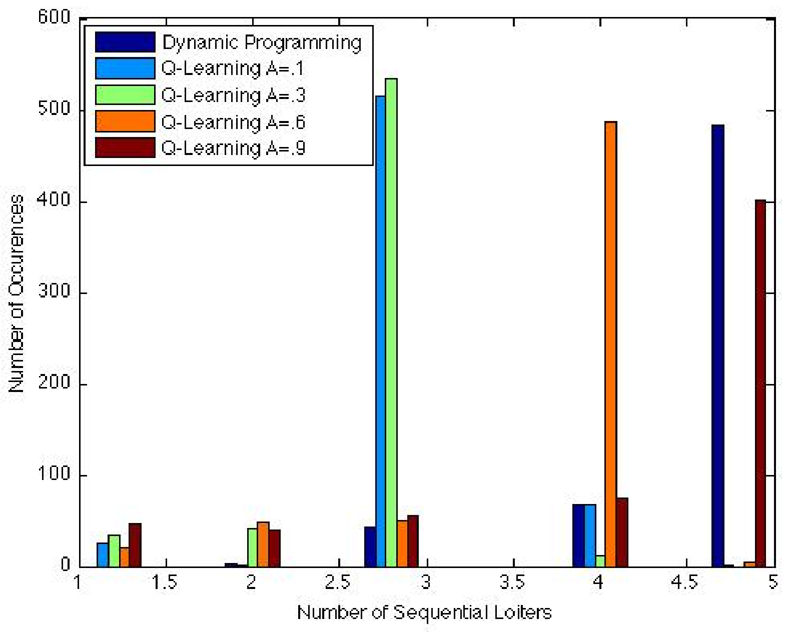


Fig. 6. Number of Sequential Loiters with Varying Step Size Constant

Figure 6 shows that as the step size constant increases the UAV dwells more thus gaining more information. Since the alert rate is set to one every thirty time steps it is a rare occurrence for more than one alert to be generated per simulation. This means that loitering five times should happen the majority of the time. The policy given by Dynamic Programming shows this trend. For the policies generated by Q-Learning as the step size constant was increased the amount of sequential loiters started to match that of the Dynnamic Programming policy's. This makes sense since the step size rate effects the learning rate and the higher the learning rate the more weight the optimization gives to the new reward from the simulation thus increasing the learning. From now on the step size constant will be set to 0.9.

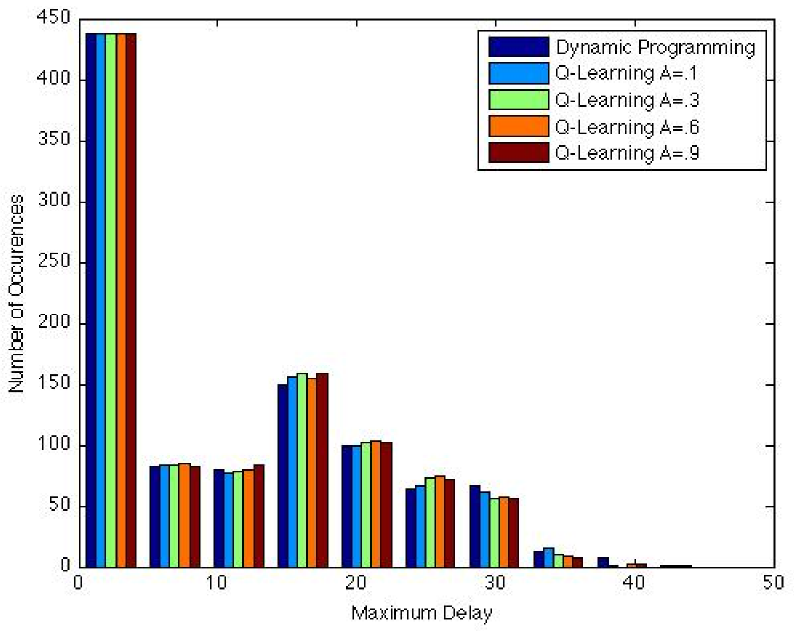


Fig. 7. Maximum Delay per Simulation with Varying Step Size Constant

Figure 7 shows the maximum delays for these policies. The majority of max delays present during the simulation are 30 minutes and below and all policies have generally the same amount. None of the policies have a maximum delay of greater than 45 minutes. The majority of alerts were serviced relatively quickly. This again is due to only one or no alert being generated for the majority of the simulations. However, in some instances more than one alert is generated increasing the max delays.

Figure 8 shows the area of Figure 7 where these multiple alerts per simulation take place. The policy generated by Dynamic Programming has the most of these maximum delay occurrences. The policy generated with a step size constant of 0.9 has the fewest.

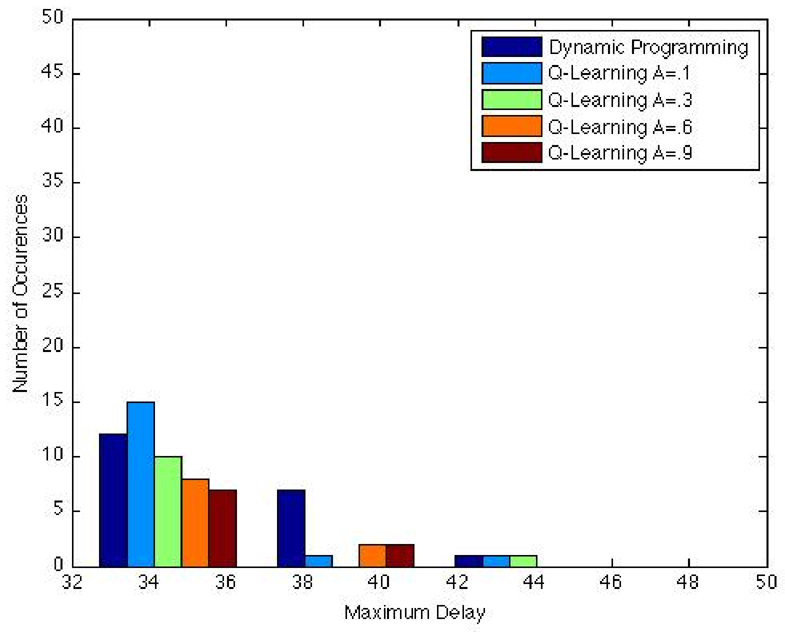


Fig. 8. Maximum Delay per Simulation with Varying Step Size Constant-Expanded

B. Varying Max Iteration (k_{max})

The next set of policies were generated with different k_{max} values. This value was set to $50,000$, $150,000$, $250,000$ and $500,000$.

Figure 9 shows as k_{max} is increased the number of sequential loiters opted by the UAV starts to match up with the results from the Dynamic Programming policy. This is until around 100000 iterations. Once this threshold is reached there is minimal improvement to matching the Dynamic Programming policy. Next is the plot for maximum delays.

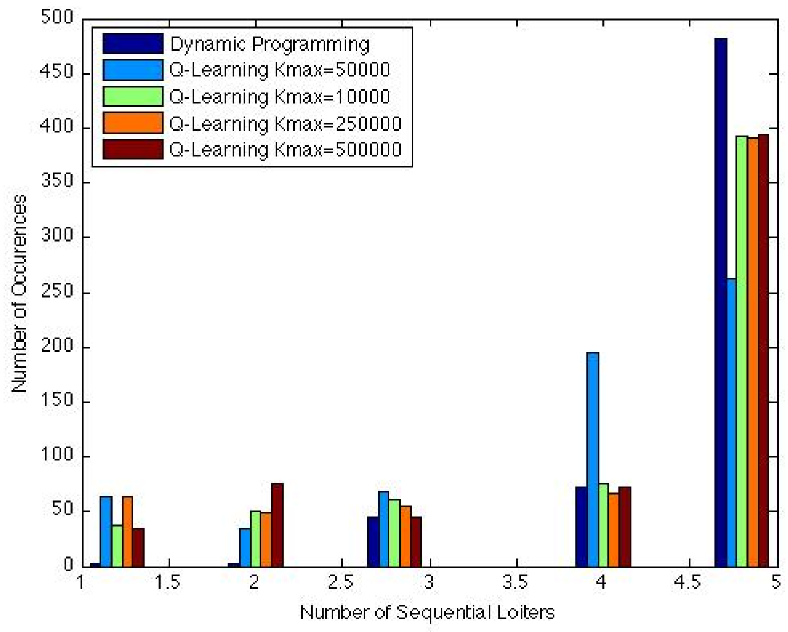


Fig. 9. Number of Sequential Loiters with Varying Max Iteration

Figure 10 shows similar happenings to that of Figure 7 with low maximum delays for the majority of simulation runs. The few times where multiple alerts were generated creating a large maximum delay are shown in more detail in the following plot.

Figure 11 shows that when Figure 10 is expanded to show the region with large maximum delays the policy given by Dynamic Programming has the most occurrences. There is little discrepancy between the policies generated by Q-Learning in this region.

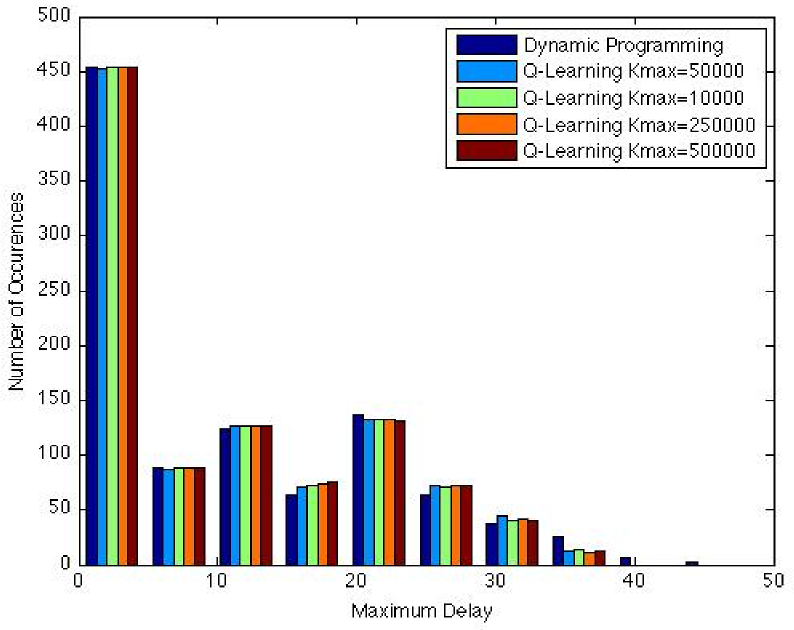


Fig. 10. Maximum Delay per Simulation with Varying Max Iteration

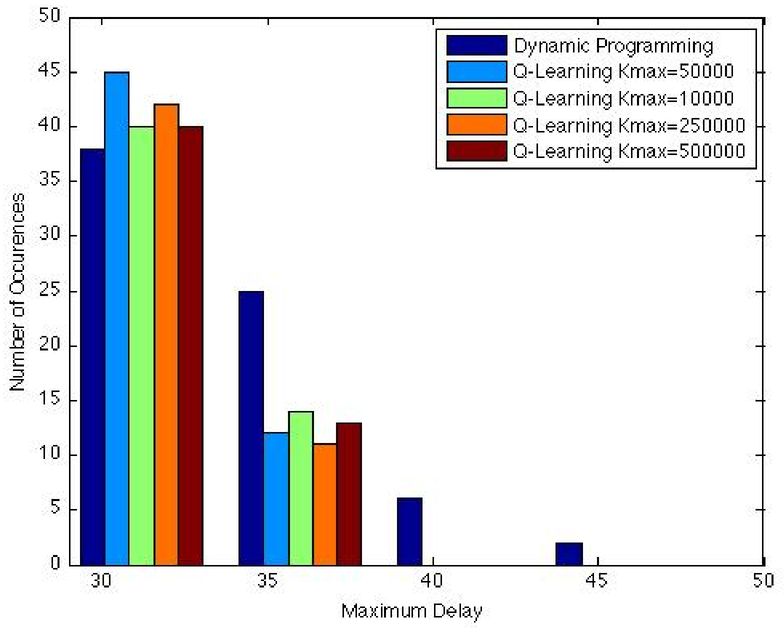


Fig. 11. Maximum Delay per Simulation with Varying Max Iteration-Expanded

CHAPTER VI

CONCLUSION

Upon completing the Q-Learning algorithm many things came to light. The first was that the Q-learning policies that were generated never fully matched that of the Dynamic Programming construction. One reason could be the way the algorithm was implemented. When choosing the random action the method used was to choose not just whether to patrol or loiter but how many times to loiter as well. During these forced loiters, information when a stochastic input is generated could be lost. This method does however have the advantage of trying to reach rare states. It is because of this that the forced loiter method was chosen. Another source of error from the Dynamic Programming policy could result from a rare state being visited when the policies are tested. If this state was not visited in the algorithm and a decision was randomly chosen as stated in the implementation the decision might not be the optimal one. A final discrepancy arises from whether to have the simulator in the algorithm as a finite or infinite horizon. The simulator was designed to run for thirty time steps but to strictly adhere to the Q-Learning algorithm the simulator would have no limit on it and run indefinitely. The problem that arises is again rare states would only get visited if the algorithm was run for a very long time. To make the implementation feasible the finite horizon method was used. This could lead to some sources of error.

It was found that the step size constant that yielded results similar to the Dynamic Programming implementation was A=0.9. This is because the step size constant is directly related to the learning rate which determines how important new information is. The number of implementations for the algorithm to run was found to be $k_{max} > 100000$ to start resembling the results generated by the Dynamic Programming implementation. In order for these two methods to match perfectly a very large amount of iterations would need to be taken.

Overall the decision of which implementation strategy performed the best comes down to computation time. The computation time for Q-Learning was on order of three times more than that of Dynamic Programming for A=0.09 and $k_{max} = 150000$, increasing when k_{max} increases. The reason is during the simulation stage of the Q-Learning algorithm the state must be kept tract of. The initial random state must also be found. This takes a tremendous amount of computation time.

REFERENCES

[1] AIAA Guidance, Navigation and Control Conference, *Optimal Perimeter Patrol Alert Servicing with Poisson Arrival Rate*, Chicago, IL, August 2009.

[2] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA, 1998.

[3] D. Gross, S. Rasmussen, P. R. Chandler, and G. Feitshans, "Cooperative operations in urban terrain (counter)," *2006 SPIE Defense & Security Symposium*, April 2006.

[4] M. Pachter, P. Chandler, and S. Darbha, "Optimal mav operations in an uncertain environment," *International Journal of Robust and Nonlinear Control*, vol. 18, pp. 248–262, 2008.

[5] M. Pachter, P. R. Chandler, and S. Darbha, "Optimal control of an atr module equipped mav-human operator team," in *Lecture notes in Economics and Mathematical Systems*, vol. 588. Springer:Berlin, 2007.

[6] C. Derman, G. J. Lieberman, and S. M. Ross, "A sequential stochastic assignment problem," *Management Science*, vol. 18, no. 7, pp. 348–355, 1972.

[7] A. Gosavi, *Simulation-Based Optimization: Parametric Optimization Techniques and Reinforcement Learning*, Kluwer Academic, Norwell, MA, 2003.

VITA

Zachary William Walton received his Bachelor's degree in Mechanical Engineering from Texas A&M University in the summer of 2008. He went on to work for Caterpillar Inc. before returning to Texas A&M University for graduate school in 2009. During the summer he was a Graduate Intern at the Air Force Research Lab in Dayton, OH. He was also a Teaching Assistant under Dr. Dara Childs for one semester in Dynamics and Vibrations.

Zach may be reached by writing to Zach Walton; Texas A&M University—Mechanical Engineering; Mail Stop 3123; College Station TX 77845.