A Q-Learning Approach to Minefield Characterization from Unmanned Aerial Vehicles

by

Stephen Greyson Daugherty

Department of Mechanical Engineering and Materials Science
Duke University

Date: _____

Approved:

_____
Silvia Ferrari, Supervisor

_____
Josiah Knight

_____
Ronald Parr

Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in the Department of Mechanical Engineering and Materials Science
in the Graduate School of Duke University
2012

Abstract

(Intelligent Control Systems)

A Q-Learning Approach to Minefield Characterization from Unmanned Aerial Vehicles

by

Stephen Greyson Daugherty

Department of Mechanical Engineering and Materials Science
Duke University

Date: _____

Approved:

_____
Silvia Ferrari, Supervisor

_____
Josiah Knight

_____
Ronald Parr

An abstract of a thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in the Department of Mechanical Engineering and Materials Science
in the Graduate School of Duke University
2012

Abstract

The treasure hunt problem to determine how a computational agent can maximize its ability to detect and/or classify multiple targets located in a region of interest (ROI) populated with multiple obstacles. One particular instance of this problem involves optimizing the performance of a sensor mounted on an unmanned aerial vehicle (UAV) flying over a littoral region in order to detect mines buried underground.

Buried objects (including non-metallic ones) have an effect on the thermal conductivity and heat retention of the soil in which they reside. Because of this, objects that are not very deep below the surface often create measurable thermal anomalies on the surface soil. Because of this, infrared (IR) sensors have the potential to find mines and minelike objects (referred to in this thesis as clutters).

As the sensor flies over the ROI, sensor data is obtained. The sensor receives the data as pixellated infrared light signatures. Using this, ground temperature measurements are recorded and used to generate a two-dimensional thermal profile of the field of view (FOV) and map that profile onto the geography of the ROI.

The input stream of thermal data is then passed to an image processor that estimates the size and shape of the detected target. Then a Bayesian Network (BN) trained from a database of known mines and clutters is used to provide the posterior probability that the evidence obtained by the IR sensor for each detected target was the result of a mine or a clutter. The output is a confidence level (CL), and each target is classified as a mine or a clutter according to the most likely explanation

(MLE) for the sensor evidence. Though the sensor may produce incomplete, noisy data, inferences from the BN attenuate the problem.

Since sensor performance depends on altitude and environmental conditions, the value of the IR information can be further improved by choosing the flight path intelligently. This thesis assumes that the UAV is flying through an environmentally homogeneous ROI and addresses the question of how the optimal altitude can be determined for any given multi-dimensional environmental state.

In general, high altitudes result in poor resolution, whereas low altitudes result in very limited FOVs. The problem of weighing these tradeoffs can be addressed by creating a scoring function that is directly dependent on a comparison between sensor outputs and ground truth. The scoring function provides a flexible framework through which multiple mission objectives can be addressed by assigning different weights to correct detections, correct non-detections, false detections, and false non-detections.

The scoring function provides a metric of sensor performance that can be used as feedback to optimize the sensor altitude as a function of the environmental conditions. In turn, the scoring function can be empirically evaluated over a number of different altitudes and then converted to empirical Q scores that also weigh future rewards against immediate ones. These values can be used to train a neural network (NN). The NN filters the data and interpolates between discrete Q-values to provide approximate information about the optimal sensor altitude.

The research described in this thesis can be used to determine the optimal control policy for an aircraft in two different situations. The global maximum of the Q-function can be used to determine the altitude at which a UAV should cruise over an ROI for which the environmental conditions are known *a priori*. Alternatively, the local maxima of the Q-function can be used to determine the altitude to which a UAV should move if the environmental variables change during flight.

This thesis includes the results of computer simulations of a sensor flying over an ROI. The ROI is populated with targets whose characteristics are based on actual mines and minelike objects. The IR sensor itself is modeled by using a BN to create a stochastic simulation of the sensor performance. The results demonstrate how Q-learning can be applied to signals from a UAV-mounted IR sensor whose data stream is preprocessed by a BN classifier in order to determine an optimal flight policy for a given set of environmental conditions.

To a good friend, an exceptional nerd, and a beautiful wife: Liesel.

Contents

List of Tables

List of Figures

List of Abbreviations and Symbols

Symbols

k	Time index
u_k	Control value at time k
x_k	System state at time k
Q	Value function for policy determination
ϵ	Exploration coefficient
G	BN structural training search space size
$\mathcal{X}$	(Control context) State space of a system; (BN context) Set of state variables $X_1, X_2, ..., X_n$
$\mathcal{F}$	(BN context) State space of $\mathcal{X}$; (Sensor simulation context) IR sensor FOV
$\mathcal{P}$	Probability distribution of $\mathcal{X}$ over $\mathcal{F}$
$\mathcal{S}$	(BN context) Set of dependencies among elements in $\mathcal{X}$; (Sensor simulation context) Sensor FOV
Ω	Probability space of the BN, graphically depicted as the network itself
θ_i	Parameter set of variable X_i; specifies the corresponding CPT
Θ	Set of all node parameters $\{\theta_1, \theta_2, ..., \theta_n\}$
π_i	Set of parents of variable X_i
$\mathcal{D}$	Dataset over which a BN is trained
μ_i	Set of children of X_i
$\mathcal{B}$	Set of structural and parameter specifications for a BN

$\mathbf{W}_a$	Input weight matrix for layer a of an NN
$\mathbf{b}_a$	Bias vector for layer a of an NN
$\mathcal{A}$	Action space of a system
$\mathcal{M}$	Markov decision process
T	Transition probability function for a stochastic process
R	Scalar reward function
π	Control policy
V^π	Expected discounted return for policy π
γ	Stepwise discount factor
π^*	Optimal policy
V^*	Optimal value function
$\mathcal{W}$	Minefield ground surface
h	Altitude
x	Generic state-space vector
y	(Context of modern control theory exposition) Generic output vector
η	Control vector
A	Generic plant matrix
B	Generic control matrix
C	Generic plant-related observer matrix
D	Generic control-related observer matrix
x_a	Twelve-dimensional aircraft state vector
u_a	Four-dimensional aircraft control vector
u	Velocity in the x-direction of the UAV body reference frame
v	Velocity in the y-direction of the UAV body reference frame
w	Velocity in the z-direction of the UAV body reference frame

x_r	North position in the terrestrial reference frame
y_r	East position in the terrestrial reference frame
z_r	Altitude indicator; $z_r = -h$
p	Roll rate in the UAV body reference frame
q	Pitch rate in the x-direction of the UAV body reference frame
r	Yaw rate in the x-direction of the UAV body reference frame
ϕ	Roll angle in the terrestrial reference frame
θ	Pitch angle in the x-direction in the terrestrial reference frame
ψ	Yaw angle in the x-direction in the terrestrial reference frame
X_b	Acceleration in the x-direction in the UAV body reference frame
Y_b	Acceleration in the y-direction in the UAV body reference frame
Z_b	Acceleration in the z-direction in the UAV body reference frame
L_b	Angular roll acceleration in the UAV body reference frame
M_b	Angular pitch acceleration in the x-direction of the UAV body reference frame
N_b	Angular yaw acceleration in the x-direction of the UAV body reference frame
$I_x x$	Moment of inertia about UAV body x-axis
$I_y y$	Moment of inertia about UAV body x-axis
$I_z z$	Moment of inertia about UAV body x-axis
$I_x z$	XZ product of inertia
V	Scalar aircraft speed
β	Sideslip angle
γ	Path angle; not to be confused with the discount factor
μ	Aircraft bank angle
δT	Throttle position
δE	Elevator angle

δA	Aileron angle
δR	Rudder angle
L_{IR}	Side length measurement for sensor FOV
H	Height of UAV above the ground
θ_{IR}	Sensor aperture angle
v_{IR}	Sensor altitude mode
y_i	Classification of target i
E	Environmental variables
M_i	Measured target properties
F_i	Actual target properties
d_i, d_{mi}	Target depth, actual and measured
z_i, z_{mi}	Target size, actual and measured
s_i, s_{mi}	Target shape, actual and measured
s_r, s_c, s_u	Soil moisture, composition, and uniformity
g	Vegetation density
w	Weather condition
i	(Non-subscript, non-ordinal context) Illumination
ζ_i	Set of hidden variables for a given cell i
c_i	Classifier confidence level for cell i
$\hat{y}_i$	Classifier description of cell i
y_i^*	Actual classification of cell i
e_i	Classification error for cell i
ρ_i	Scoring weight assigned to a given cell, based on classification and ground truth
W_v	Risk function multiplier for correct classifications
W_e	Risk function multiplier for incorrect classifications

ξ_{IR}	Spatial resolution of the IR sensor, in meters
B	Bias in the target size measurement
σ	Random variable scaling coefficient for introducing noise to the size measurement
$\mathcal{E}$	Flight envelope
$*_h$	Set of all measured bins for a given altitude mode h for a particular flight path over the ROI
r'_h	Sum of rewards for a given flight over a field
$R'(h)$	Average of r'_h over a set of ten trials over the same ROI

Abbreviations

BN	Bayesian Network
DP	Dynamic Programming
ADP	Approximate Dynamic Programming
CPD	Conditional Probability Distribution
CPT	Conditional Probability Table
CL	Confidence Level
DAG	Directed Acyclic Graph
FOV	Field of View
IR	Infrared, typically in reference to a sensor type
NN	Neural Network
PMF	Probability Mass Function
ROI	Region of Interest
UAV	Unmanned Aerial Vehicle
MLE	Most Likely Explanation
MDP	Markov Decision Process
UXO	Unexploded Ordnance

APM	Anti-Personnel Mine
ATM	Anti-Tank Mine
CLUT	Mine-Like Clutter
POMDP	Partially-Observable Markov Decision Process
STOL	Short Takeoff and Landing

Acknowledgements

I would like to thank my advisor, Dr. Silvia Ferrari, for the energy she has invested and the opportunity she has offered me.

I would also like to thank the team from BAE Systems–Dr. Caryl Johnson, Dr. Carla Hagler, and Dr. Glenn Nofsinger–for their interest in our research and the productive discussions resulting from that interest. I would like to thank Guoxian Zhang for his help and advice, and I would like to thank Dr. Jeffrey Scruggs for his wonderfully in-depth classes. Additionally, I would like to acknowledge the frequent assistance of Kathy Parrish, the most organized person I know.

Last, I would like to offer all possible gratitude to my wife, family, and friends for their encouragement.

1

Introduction

1.1 Motivation and Background

The performance of a sensor mounted on an aircraft flying over a field can be quite difficult to anticipate precisely because a number of environmental variables cannot be carefully controlled or precisely measured. These variables include (but are not limited to) weather factors such as wind speed distribution, ambient temperature, humidity, air opacity, cloud cover, time of day, and precipitation, as well as ground variables such as soil type, moisture distribution, and ground temperature. In addition to the impracticality of tracking all the variables that could influence detector performance, their precise relationships to what a sensor will "see" at a given moment from a given altitude are likely to be mathematically complex or incompletely understood. Given this lack of determinism, it is necessary for a system to make online adjustments to its control policy in order to achieve near-optimal performance.

In order to optimize the use of sensor data, it is necessary to distinguish between mines and mine-like objects based on sensor inputs. In the absence of single, ideal (or near-ideal) indicators that could make high-probability or absolute distinctions

between mines and clutters, it is possible to use Bayesian Networks (BNs) to infer what a given target is more likely to be (Cai and Ferrari, 2009). This has been proven possible in Ferrari and Vaghi (2006), where BNs were used both to model what various types of sensors (including IR sensors) would perceive under different environmental and operational conditions, and to infer from those perceptions whether detected targets were more likely to be mines or clutters. This project used examples taken from the Ordata Database, a collection of information about 5000 different mines and 3000 different clutters, along with the measurements taken of them by electromagnetic induction, ground-penetrating radar, and IR sensors and the conditions under which the measurements were taken (Explosive et al., 2006). It then used that information to develop a sensor fusion model, where an inference about the classification of a target was formed based on multiple sensor inputs (Ferrari and Vaghi, 2006). The research performed for this thesis makes use of the same data set and the same IR sensor model.

Given a stochastic model that relates control decisions to sensor performance, one must find a way to optimize future control decision based on this information. In cases where reward is quantified *a priori* as a function of the state of a system, it is sometimes possible to use dynamic programming (DP) to work backwards from a finite planning horizon to determine an optimal control sequence. When the planning horizon is discretized into some number of stages and the state space of the system is discretized into a finite number of accessible states for the given boundary conditions, optimal path planning is assured (White and Jordan, 1992). However, dynamic programming can become computationally unrealistic for multidimensional state spaces (according to the "curse of dimensionality"). When DP implementations are infeasible, various methods in approximate dynamic programming (ADP) can often find near-optimal solutions in a more reasonable amount of time (Werbos, 2004).

ADP methods are also better-suited to many problems where little is known about the consequences of various control actions *a priori*. One such method comes from the doctoral thesis of Christopher Watkins. He used a lengthy set of parallels with animal behavior to explore the general problem of learning optimal behavior when the consequences of an agent's actions are not precisely predictable (Watkins, 1989). Watkins applied this to the "two-armed bandit problem." The idea of the problem is that an agent sitting in front of a slot machine that is equipped with two levers, each of which dispenses a reward according to some probability distribution. There is a dilemma because the agent is unaware of what the probability distributions are, so it must spend some time experimenting in order to determine which lever tends to give a greater amount of reward, but the number of times any lever can be pulled is limited. Ergo, the agent must learn some way to balance the tradeoffs. One way to accomplish this is by modeling the policy by which future actions are chosen as a Markov Decision Process (MDP, described in Bellman (1957)), and then mixing trial and error with exploitation of the best expected action using an algorithm called Q-Learning (Watkins, 1989).

In the context of the problem at hand, Q-Learning can be used to choose whether an aircraft should ascend, descend, or cruise at its current altitude in order to optimize sensor performance. The primary strength of this method is that the airborne agent does not require an *a priori* model about the relationship between the system variables and the sensor performance; based on empirical data, the agent can estimate what the optimal control policy will be.

One problem with the basic Q-Learning algorithm, however, is that estimation of a Q-score requires an algorithm that uses value iteration to compute the relative values of finite (and therefore discrete) numbers of states or state-action pairs. This thesis endeavors to estimate the optimal control policy over a continuous state variable. Ergo, it is necessary to generalize the Q-function over a continuous state

space using a set of empirically measured values. Since Cybenko (1989) proved that two-layer feedforward neural networks (NNs) can serve as universal function approximators, NNs can be adapted for this purpose.

For online Q-Learning, researchers sometimes use a coefficient ϵ (where $0 \leqslant \epsilon \leqslant 1$) to balance between exploration of the state space and exploitation of the expected rewards. In such a case, one may act randomly (or otherwise against the greedy policy) with probability ϵ and follow the greedy policy strictly with probability $1 - \epsilon$ (Watkins, 1989). However, the research completed thus far focuses on offline learning, so the exploration/exploitation tradeoff is not important. This is because the research uses a supervised learning process to determine the values of Q assessed by a neural network. In further research, this information can be used with an ϵ-greedy policy or to initialize an actor-critic architecture (as described in Jensensius (2005); Si et al. (2004)), either of which would adapt online.

1.2 Research Objectives

Airborne mine detection can be used to cover a large land area in a short amount of time. Ground robots often must plan paths that skirt around impenetrable obstacles (Cai and Ferrari, 2009), whereas airborne sensors can pass right over, giving them much more flexibility to explore regions as desired, bounded only by the flight envelope of the platform. However, not all airborne detector systems are created equal, and any way of changing an aircraft's trajectory within its flight envelope in order to improve sensor performance will increase the strategic value of the system. The research presented here endeavors to serve this purpose by showing how neural network Q-learning can be used to find a near-optimal flight policy.

Despite any measure of optimization, airborne demining is bound to be an imperfect undertaking. An aircraft cannot carry as wide an array of sensors as a ground vehicle can; nor can fixed-wing aircraft stop over a field to investigate points of in-

terest more closely. Therefore, this research is of greater interest to someone seeking to form some general characterization of characteristics of a minefield than someone seeking to pinpoint every possible location of a mine or clear a path over which ground vehicles or personnel will travel without further demining efforts. Within this framework of general minefield observation, the research presented here seeks to improve sensor performance by helping the aircraft to find its optimal observation altitude automatically.

Furthermore, the research develops a flexible framework by which different objectives may be served by adjusting the weights given to the performance function. For instance, if an aircraft's mission is intended to seek out mine locations in an area as a preliminary effort to locate mine clusters for removal, then a greater numerical emphasis may be placed on correct detections and false negatives than false positives or correct negatives. In short, different missions may have different optimal policies, and the model presented seeks to optimize not only performance within a situation, but also the spectrum of the system's usefulness.

Within the scope of this research, it is assumed that the ground truth of the objects evaluated is known in order to provide some feedback either in the field or immediately after a set of flights has been made. One might object that if the complete ground truth were known in a given situation, then there would be little reason to deploy the aircraft for additional observations. However, the algorithms explained in the pages that follow can easily be extended to the real world through actor-critic architectures or semisupervised learning.

1.3 Thesis Organization

The main body of this thesis is comprised of six chapters. Chapter 2 includes a basic introduction to the inner workings of Bayesian and neural networks, as well as an overview of Q-Learning. Chapter 3 formally describes the airborne treasure-hunt

problem and outlines the steps used to solve it. Chapter 4 contains some specific details of the methodology implemented in this research. Chapter 5 presents and analyzes the outcomes of the simulated implementations of the algorithms. Chapter 6 contains conclusions of the research and recommendations for the future.

2

Bayesian Networks, Neural Networks, and Q-Learning

2.1 Bayesian Network Interpretation

Fundamentally, Bayesian networks use *a priori* information about relationships between variables to calculate the likelihood of an unknown variable's possible states, given some information about known variables that affect the unknown's likelihood. There are many variations on BNs. The ones used in research for this thesis allow each variable to have a small number of discrete states as inputs. For the example that will follow shortly, the variables will be binary; that is, each one will have two possible states. The example is similar in structure to one that is used for demonstration by Murphy (2007) and Baumgartner (2005).

In a BN, variables are placed into positions called nodes (Murphy, 2007). Each node can be observed or unobserved, meaning that knowledge might exist about the actual state of the variable, or it might not. In either case, it is assumed that prior information has been gathered that provides information on how the state of each variable affects the probability distributions of the others.

Drawing a BN involves depicting nodes and the relationships between them. These relationships are represented by arrows called arcs. If an arc is drawn from one node to another, then the source node is a *parent*, and the end node is a *child*. Portrayal of an arc means that information about the status of a parent will affect the probability mass function (PMF) of the child. Every possible contingency for the PMF given the possible states of the parents is given as a conditional probability distribution (CPD) and, for discrete cases like the ones treated here, can be listed in a conditional probability table (CPT) (Murphy, 2007).

Consider, for instance, an example in which one wishes to describe the probabilistic dynamics between the maintenance of an old car, the functionality of some of its components, and whether a cold start is easily manageable. The graphic relationships among the variables are shown in Figure 2.1.

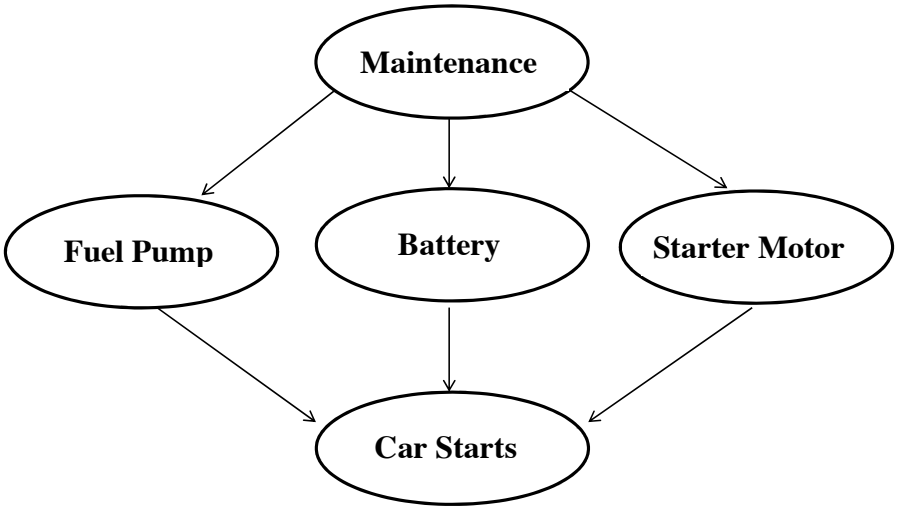


FIGURE 2.1: Simple BN

As stated previously, the arcs identify parents and children. If M is taken to stand for complete and regular maintenance, F for the function of the fuel pump, B for the strength of the battery, S for the function of the starter motor, and C for

whether or not the car will start, then F, B, and S are the children of M, and C is the child of F, B, and S. Examples of CPTs that would describe these variables are given in Tables 2.1, 2.2, 2.3, 2.4, and 2.5.

Table 2.1: CPT for M

$\mathbf{P}(M = \mathbf{T})$	$\mathbf{P}(M = \mathbf{F})$
0.4	0.6

Table 2.2: CPT for F

M	$\mathbf{P}(F = \mathbf{T})$	$\mathbf{P}(F = \mathbf{F})$
T	0.95	0.05
F	0.55	0.45

Table 2.3: CPT for B

M	$\mathbf{P}(B = \mathbf{T})$	$\mathbf{P}(B = \mathbf{F})$
T	0.8	0.2
F	0.7	0.3

Conditional probabilities like those in the tables can be expressed in the form $P(X = x|Y = y)$, where the lowercase letters represent exact states and the capital letters represent variables. In this format, one could write $P(\text{C}= T|\text{F}= T,\text{S}= T,\text{B}= T) = 0.995$ after looking at the CPTs. It is important to make a few observations here. One is that directed cycles are prohibited in BNs. Because of this, BNs are *directed acyclic graphs*, or DAGs. Another is that, though CPTs only show the direct influence of parents on the probability distributions of their children, observed children can affect unobserved parents' PMFs. This can occur through Bayes's Theorem, expressed as

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}, \tag{2.1}$$

Table 2.4: CPT for S

M	$\mathbf{P}(S = \mathbf{T})$	$\mathbf{P}(S = \mathbf{F})$
T	0.85	0.15
F	0.65	0.35

Table 2.5: CPT for C

F	B	S	$\mathbf{P}(C = \mathbf{T})$	$\mathbf{P}(C = \mathbf{F})$
T	**T**	**T**	0.995	0.005
T	**T**	**F**	0.4	0.6
T	**F**	**T**	0.2	0.8
T	**F**	**F**	0.1	0.9
F	**T**	**T**	0.55	0.45
F	**T**	**F**	0.25	0.75
F	**F**	**T**	0.15	0.85
F	**F**	**F**	0.001	0.999

where X and Y are two separate variables. If X is observed but Y is not, then, given information about the prior probability of Y and the conditional probability $P(X|Y)$, it is possible to produce a likelihood of Y conditioned on X (Murphy, 2007).

The architecture just described allows for simple inferences to be made about some variables even when not all the remaining variables in a graph are known. This makes it a highly flexible tool for probabilistic reasoning.

2.2 Bayesian Network Formulation

A Bayesian network is a graph-theoretic construct that describes dependencies among a set of random variables $\mathcal{X} = X_1, X_2, ..., X_n$ when those distributions are interrelated. This thesis will generally use capital letters (e.g., X) to refer to unknown variables and lowercase letters (e.g., x) to refer to particular instances of those variables. Let $\mathcal{F}$ be the space of all possible instantiations of $\mathcal{X}$, and let $\mathcal{P}$ be the probability distribution of $\mathcal{X}$ over $\mathcal{F}$. In addition, let $\mathcal{S}$ be the set of all dependencies among variables in $\mathcal{X}$. Then $\Omega = \{\mathcal{X}, \mathcal{S}\}$ represents the probability space of the BN.

The graphic representation of the Bayesian-network model is given by Ω. That is, $\mathcal{X}$ specifies the nodes of the network, and $\mathcal{S}$ specifies the structure connecting the nodes (i.e., the arcs between the nodes). In order to derive any useful information from the network, it is necessary that we provide specific information about the conditional probability distribution of each node. For each node X_i, this is given by a parameter θ_i. Let π_i represent the set of parents of X_i. Then each θ_i provides information about $P(X_i|\pi_i)$, which specifies a CPT. Examples of CPTs are given in Tables 2.1, 2.2, 2.3, 2.4, and 2.5. The set $\{\theta_1, \theta_2, ..., \theta_n\}$ will be represented by Θ.

Building a BN requires one to process statistical information from some dataset $\mathcal{D}$. A dataset is composed of some number of samples, where each sample provides a single, simultaneous instance of every variable in $\mathcal{X}$. Once a BN is trained (explained in fuller detail below), one can use it to make guesses about the states of variables in the graph whose actual instantiations are not known. In order to relate parents and children to one another, we can use instances of Bayes's rule (shown generally in Equation 2.1) given by

$$P(X_i|\mu_i) = \frac{P(\mu_i|X_i)P(X_i)}{P(\mu_i)}, \tag{2.2}$$

where μ_i represents the set of children of X_i, $P(X_i)$ is the prior probability (i.e., unconditioned probability distribution over all possible states of X_i ($(x_{i,1}, x_{i,2}, ..., x_{i,r(i)})$, where $r(i)$ gives the number of possible states for X_i), and $P(\mu_i)$ is the prior probability of the states of X_i's children.

One might notice that this probability distribution is not especially straightforward to calculate, but if the children of X_i are d-separated whenever X_i is observed (i.e., as long as observations about the state of some children do not affect the PMFs for any of the unobserved children as long as the state of X_i happens to be known), then the distribution can be determined in two steps. First, it is necessary to find a

distribution for $P(\mu_i|X_i)$. Since this is not related to a single variable, one cannot take it directly from a CPT. Instead, one must use the equation

$$P(\mu_i|X_i) = \prod_{j=1}^{p} P(\mu_{i,j}|X_i), \tag{2.3}$$

where p is the total number of X_i's children, and $\mu_{i,j}$ is the value of the jth child of X_i (Jensen, 2001). The quantity $P(\mu_i|X_i)$ is known as the likelihood of μ_i. Note here that the necessary values of $P(\mu_{i,j}|X_i)$ can be taken from the CPTs for μ_i; this means that the likelihood is directly obtainable from $\mathcal{D}$. Once one has calculated $P(\mu_i|X_i)$, one can find $P(\mu_i)$ by marginalizing over all possible states of X_i. This can be done by computing

$$P(\mu_i) = \sum_{k=1}^{r(i)} P(X_i = x_{i,k})P(\mu_i|X_i). \tag{2.4}$$

Once $P(\mu_i)$ is known, $P(X_i|\mu_i)$ quickly follows from Equation 2.2 (Heckerman, 1995; Baumgartner, 2005). This is known as the posterior probability, and it is useful for finding out what value of X_i would have "caused" the values observed among its children. In order to make a best guess about the value of x_i when X_i is unobserved, one would choose a value of k to maximize $P(X_i = x_{i,k}|\mu_i)$. This process is known as inference.

2.3 Training Bayesian Networks

2.3.1 Training and Validation Sets

When a Bayesian-network methodology is applied to a set of raw data, the structure and parameters of the network (represented as $\mathcal{B} = (\mathcal{S}, \Omega)$) are often not known *a priori*. Therefore, it is necessary to train the network to model the data provided. But first, it is necessary to divide the data to be modeled into a training set and a

validation set. The reason is because the training process requires a search of a large space of possible instances of $\mathcal{B}$, and some scoring heuristic is helpful in this search. To that end, one set will be used to estimate the dependencies among the variables, and the other set (i.e., the validation set) will be used to compare the quality of those estimates with one another. The reason that the two sets must be separate is because BNs are intended to be used for extrapolation to cases not contained in the training set. If the predictions of a BN were scored against the same data used to train it, then the network would almost surely overfit the training data, and would therefore generalize poorly (Murphy, 2007).

2.3.2 Structural Training

For any grouping of variables taken from a dataset, it is necessary to determine what configuration of nodes best represents the dependencies among the variables. In other words, it is necessary to find a directed acyclic graph (DAG) that maximizes the statistical significance of variable correlations. This can be difficult, given that the number of possible DAGs grows quickly as a function of the number of input variables, according to the equation

$$G(n) = \sum_{k=1}^{n} (-1)^{k+1} \binom{n}{k} 2^{k(n-k)} G(n-k) \tag{2.5}$$

According to this equation, only 3 different DAGs can be formulated using 2 variables. However, 10 variables can have 4.2 quintillion configurations (Murphy, 2007). Scoring functions exist for estimating the likelihood of a dataset, but because of Equation 2.5, an exhaustive search of the space of possible DAGs is often not possible. In order to fix this, there are a few common search algorithms. One, called K2, adds node connections according to a greedy policy; at each step, the algorithm generates all possible graphs that result from the addition of a single arc to the

current graph. Each DAG is then judged by a Bayesian scoring criterion, and the best in the group becomes the new reference point (Cooper and Herkovits, 1992).

Another algorithm that is commonly used for searching the DAG structure space is hill climbing. This method defines "neighbors" as all DAGs that can be generated by modifying one arc (or lack of arc) (Murphy, 2007). This algorithm (as well as K2 and practically all other non-exhaustive techniques) is likely to wind up in a local maximum. With the hill-climbing algorith, users can often edge closer to the global maximum by running each algorithm a large number of times with randomly initialized structures. In that case, the highest-scoring result subject to the acyclicity constraint would become the working DAG structure.

Other search-and-score techniques exist, but there is one more thing to keep in mind: Though the direction of a BN arc is often conceptualized as the parent "causing" the children, empirical search-and-score structural training can only be expected to produce a result that is Markov-equivalent to a "correct" model. In other words, while BNs are excellent tools for identifying correlations among variables and using those correlations for purposes of statistical inference, correlation does not determine causation (Murphy, 2007).

2.3.3 Parameter Training

Once a BN structure is determined, it is necessary to estimate the prior and conditional probabilities that will populate the CPTs. Given the structural dependencies of the BN, an algorithm can search for the position in the parameter state space that maximizes the likelihood of the dataset provided for training. The result (ideally) is the most likely explanation (MLE) of the training set. More realistically, a parameter training algorithm will search the parameter space and return the most likely instance of Ω that it finds (Murphy, 2007).

2.4 Neural Network Structure

Neural networks (NNs) commonly serve one of two purposes: classification or function approximation. In the former case, networks called perceptrons can be trained to distinguish between linearly separable groups of data points (Demuth et al., 2010), but these will not be discussed in detail here. The latter case is more important for the purposes of this thesis, for reasons that will be described shortly.

First, it is necessary to describe the workings of a single neuron. Figure 2.2 depicts a single neuron. Here, $\mathbf{x}$ represents a single, two-component input vector, b is a scalar bias, and y is the output. The first block performs a weighted summation using a weight vector $\mathbf{W}$. The linear graph at the right side of the figure indicates that the neuron has a linear transfer function, so its output will be given simply by

$$y = \mathbf{W}\mathbf{x} + b \tag{2.6}$$

(Demuth et al., 2010).

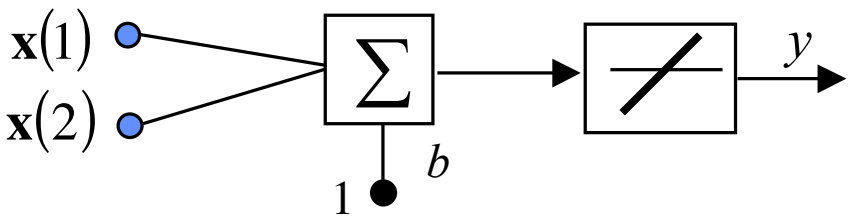


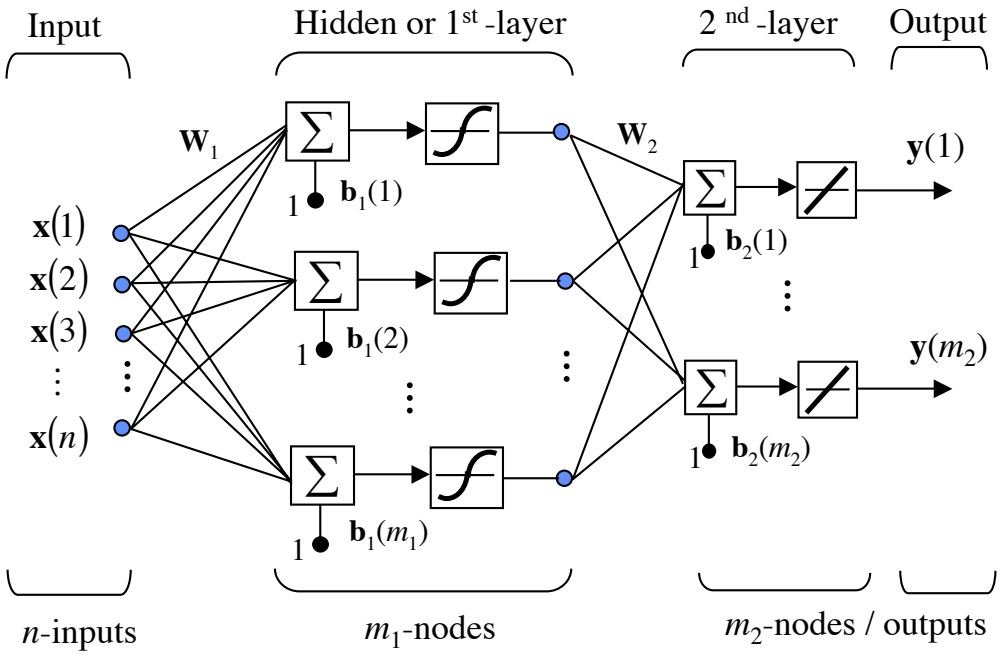
FIGURE 2.2: Single neuron with linear transfer function (taken from Ferrari (2000)).

Not all neurons are the same. Most importantly, different transfer functions are possible. The most important alternative to the linear transfer function (for the purposes of this thesis) is the sigmoid transfer function. The two types of sigmoid transfer function are tan-sigmoid and log-sigmoid. Their respective functions are given in Equations 2.7 and 2.8 (Demuth et al., 2010).

$$y = tanh(\mathbf{W}\mathbf{x} + b) \tag{2.7}$$

$$y = \frac{1}{1 + e^{-(\mathbf{W}\mathbf{x}+b)}} \tag{2.8}$$

If inputs are fed to a set of sigmoid neurons in parallel, whose outputs are in turn connected to linear neurons, the result is referred to as a two-layer feedforward neural network. This architecture is depicted in Figure 2.3. The layer that is not connected directly to the outputs of the network (in this case, the array of sigmoid neurons) is referred to as the "hidden" layer.



$\mathbf{W}_1$ = input weight matrix of 1st -layer; $\mathbf{b}_1$ = vector bias of 1st -layer

$\mathbf{W}_2$ = input weight matrix of 2 nd -layer; $\mathbf{b}_2$ = vector bias of 2 nd -layer

FIGURE 2.3: Two-layer feedforward neural network (from Ferrari (2000)).

2.5 Neural Network Training

Cybenko (1989) proved that a feedforward network containing a sufficient number of sigmoidal neurons in a single hidden layer connected to a linear output layer can approximate any function that has a finite number of discontinuities. Hence, NNs can be very useful as function approximators. In particular, they can be used to estimate the shape of an unknown function that underlies an observed set of data. Since empirical observations are necessarily discrete and finite, NNs can help to interpolate between data points. Also, since observational data often suffers from some amount of noise, NNs can work as filtering systems.

Neural networks can only accomplish these tasks if they are given the right parameters to do it. That is, the vectors $\mathbf{b}_1$ and $\mathbf{b}_2$, as well as the matrices $\mathbf{W}_1$ and $\mathbf{W}_2$, must be populated with the correct values to approximate the unknown function $\mathbf{T} = f(\mathbf{x})$, where $\mathbf{T}$ is taken from the set of observationally determined targets and is related (by unknown dynamics) to the set of input vectors. Finding these parameters requires a training algorithm to search the parameter space for a good match.

However, given the needs of plausible interpolation and noise filtering, NN training is not a simple matter of good search algorithms; size matters. If the sigmoidal layer contains too few neurons, then the range of possible function outputs may not be sufficiently dynamic to model the underlying function with acceptable accuracy. On the other hand, using too many neurons can lead to overfitting, where noise is not filtered out, and the user receives little or no new and useful information. In severe cases of overfitting, arbitrary curves may appear between empirical data points that have no connection to reality. Preventing this systematically is not simple. Some rules of thumb exist, such as using approximately one-fifth as many sigmoid neurons as one has input-target pairs. More formally, a few algorithms address the issue

probabilistically. For the purposes of this thesis, a technique known as Bayesian regularization is notable; given a larger-than-necessary NN, it uses Bayesian-statistical methods to formalize Occam's Razor and determine how many network parameters are necessary, nullifying the rest (MacKay, 1992a,b).

2.6 Q-Learning

Approximate dynamic programming (ADP) methods, such as Q-Learning, are valuable tools for solving optimal control problems online, subject to partial or imperfect knowledge of the system state and models (Ferrari and Stengel, 2004). Optimal control problems involve a dynamic system (or process) that is either stochastic or deterministic. Although various notations are in use in the ADP literature (Si et al., 1992), in this thesis, we will adopt the notation that is typically used in the optimal control and dynamic programming community (see Bertsekas (1995) for a detailed description and introduction). Assuming time can be discretized and indexed by k, a deterministic dynamical system may be modeled by the difference equation,

$$x_{k+1} = f(x_k, u_k, k) \tag{2.9}$$

where the state x_k at time k is an element of the *state space* $\mathcal{X}$, and the control u_k at time k is an element of the space $\mathcal{A}$ of admissible actions or decisions. If the dynamical system is stochastic, then it may be modeled as an MDP (Bertsekas and Tsitsiklis, 2002). An MDP is a tuple $\mathcal{M} = \{\mathcal{X}, \mathcal{A}, T, R\}$ representing a random and sequential decision process. In this case, the state space is a finite set of possible state values, denoted by $\mathcal{X} = \{s_1, \ldots, s_n\}$, and the space $\mathcal{A} = \{a_1, \ldots, a_m\}$ is a finite set of admissible actions or decisions. T is the transition probability function, $T : \mathcal{X} \times \mathcal{A} \to P(\mathcal{X})$, which describes the MDP state transitions, such that whenever the state at time k has value $x_k = s_i$ and the decision is $u_k = a_j$, there is a probability

$P(x_{k+1} = s_l \mid x_k = s_i, u_k = a_j)$ that the next state value is $x_{k+1} = s_l$. In many real-world applications of optimal control, however, the exact form of the difference equation (2.9) or the transition matrix T are unknown or approximate, and can only be determined online.

In optimal control problems, there exists a reward associated with the dynamic system that may be represented by the reward function, $R : \mathcal{X} \times \mathcal{A} \rightarrow \mathbb{R}$, specifies the value of the immediate reward, $r_k = R(\ x_k, u_k)$, received after executing the action decision u_k in state x_k. A policy is a mapping of state values to actions, $\pi : \mathcal{X} \rightarrow \mathcal{A}$. Let the value function $V^\pi(x_k)$ denote the expected discounted return of a policy π, defined as:

$$V^\pi(x_k) = E\left\{\sum_{i=0}^{\infty} \gamma^i r_{k+i} \mid \pi,\ x_k\right\} \tag{2.10}$$

where r_{k+i} is the reward received i steps into future, and the discount factor $0 \leqslant \gamma < 1$ modulates the effect of future rewards on present decisions, with small values emphasizing near-term gain and larger values emphasizing later rewards. Then, an *optimal policy* π^* is one that maximizes $V^\pi(x_k)$ for all possible states $x_k \in \mathcal{X}$. The Markov property guarantees that an optimal policy exists, though it may not be unique, and, thus, it is associated with an *optimal value function* $V^*(x_k) = \max_\pi V^\pi(x_k)$. The optimal policy of an MDP, $\mathcal{M}$, is a fixed point of Bellman's equation, which can be determined iteratively using policy iteration or value iteration algorithms (Russell and Norvig, 2003, Chapter 5).

In value iteration, the value of a state $V(x_k)$ is the total expected discounted reward accrued by a policy starting at $x_k \in \mathcal{X}$. The Q function of a state-action pair, $Q(x_k, u_k)$, is the total expected discounted reward accrued by a policy that produces $u_k = \pi(x_k)$ (Russell and Norvig, 2003, Chapter 5). The Bellman equation can be formulated in terms of the aforementioned functions, such that the state-action value

function is

$$Q(x_k, u_k) = E\left\{R(x_k, u_k) + \gamma V(x_{k+1})\right\} \tag{2.11}$$

$$V(x_{k+1}) = \max_{u_{k+1} \in \mathcal{A}} Q(x_{k+1}, u_{k+1}). \tag{2.12}$$

If two functions $Q(\cdot)$ and $V(\cdot)$ satisfy the above Bellman equation, then they specify an optimal *greedy* policy

$$\pi^*(x_k) = \arg\max_{u_k \in \mathcal{A}} Q(x_k, u_k) \tag{2.13}$$

Value-iteration algorithms use eq. (2.11) to iteratively determine $Q(\cdot)$ and $V(\cdot)$ and, subsequently, determine $\pi^*(\cdot)$.

The primary difference between pure value iteration and Q-learning is versatility. Value iteration can be used to determine the optimal policy of an MDP, $\mathcal{M}$, provided that the rewards to be gained by following a given policy from a given state are predictable. If this is not the case, then Q-Learning can be utilized to learn an approximate state-action value function $Q(x_k, u_k)$ that is iteratively updated by the rule,

$$\begin{aligned} Q(x_k, u_k) &\leftarrow (1-\alpha)Q(x_k, u_k) \\ &+ \alpha\big[r_k + \gamma \max_{u_{k+1} \in \mathcal{A}} Q(x_{k+1}, u_{k+1})\big] \end{aligned} \tag{2.14}$$

where α is the learning rate, and $0 < \alpha \leqslant 1$. In this thesis, Q-Learning is implemented using NNs in order to solve a new sensor planning problem described in the next section, which consists of obtaining the optimal guidance policy for an IR sensor deployed onboard an UAV for mine detection and classification.

3

Problem Formulation

3.1 Environmental Model

The problem considered in this thesis consists of learning an optimal guidance policy for a UAV with an onboard infrared (IR) sensor that flies over a minefield $\mathcal{W} \subset \mathbb{R}^2$, or region of interest (ROI), for the purpose of detecting and classifying buried landmines and unexploded ordnance (UXOs). Each bin in $\mathcal{W}$ is randomly selected to be either empty or seeded with a mine or a piece of minelike clutter (CLUT). Any subset of the environmental conditions E (enumerated in Table 3.1) can vary from one bin to another, or from one set of bins to another. However, for the purposes of the research performed for this thesis, it is assumed that the environmental conditions are uniform across the entire ROI.

It is assumed that all components of the state of E are reliably knowable. It is also assumed, for training purposes, that the "ground truth" of the bin classification (i.e., whether a bin contains a mine, a clutter, or nothing), can be determined *after* a flight has been completed for purposes of evaluating the scoring function. However, during a flight, target existence, characteristics F_i, and classification are not assumed to be

known and are subject to measurement errors. These errors are described further in Section 3.4.

3.2 Modern Control Theory and Linear State Space Decomposition

In recent decades, microprocessors have revolutionized control system design. In modern control theory, various aspects of a system's behavior are formulated as a set of vectors and matrices. These mathematical constructs lend themselves particularly well to automated computations (Nelson, 1998). As Nelson (1998) describes, we can describe relevant components of a system's state as a vector, x. The vector x can include a number of specific system state metrics; but in order for the model to be linearizable, the evolution of variables included in x needs to be determinable (or nearly determinable) based on information about the states of and controls applied to the rest of the variables in x. In order to make this possible for a wide variety of different systems, x often includes both a variable and one or more of its derivatives with respect to time (Nelson, 1998).

In order to describe the natural evolution of the system with time (that is, with neutral control inputs), it is necessary to use a *plant matrix*, A. In particular, when we have neutral contol inputs, the differential equations governing the system dynamics can be written as

$$\dot{x} = Ax \tag{3.1}$$

Given this information, it is then possible to use the additivity of linear systems to account for the effect of control inputs. This requires that we first quantify the way in which each control input affects each of the state variables. We can manage this by formulating the control inputs themselves as a vector η and their effects on the state variables as a matrix B. This yields the equation

$$\dot{x} = Ax + B\eta \tag{3.2}$$

The vector y represents a generic output of the system (as described in Nelson (1998) and Scruggs (2010)), which relates the actual state of the system to the measured output according to the equation

$$y = Cx + D\eta \tag{3.3}$$

In cases where the values of the state variables are assumed to be precisely knowable (and where these variables are exactly what we want to observe), then C is an identity matrix and D is populated with zeroes.

The reason that this modeling technique is so applicable to a wide variety of systems is because the state-space decomposition method can accommodate any set of linear differential equations with constant coefficients. This includes differential equations of arbitrarily high order; as Nelson (1998) shows, any equation of the form

$$\frac{d^n c(t)}{dt^n} + a_1 \frac{d^{n-1} c(t)}{dt^{n-1}} + a_2 \frac{d^{n-2} c(t)}{dt^{n-2}} + ... + a_{n-1} \frac{dc(t)}{dt} + a_n c(t) = r(t) \tag{3.4}$$

can be transformed into a state space formulation by using $c(t)$ as an output and $r(t)$ as an input, then choosing the state variables such that $x_1(t) = c(t)$, $x_2(t) = \frac{dc(t)}{dt}$, $x_3(t) = \frac{d^2 c(t)}{dt^2}$, etc. This yields the decomposition

$$
\begin{aligned}
A &= \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \dots & 1 \\ -a_n & -a_{n-1} & -a_{n-2} & -a_{n-3} & -a_{n-4} & \dots & -a_1 \end{bmatrix} \\
B &= \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} \\
C &= \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \end{bmatrix} \\
D &= 0
\end{aligned} \tag{3.5}
$$

Given this, any set of differential equations that can each be written in the form of Equation 3.4 can therefore be used to construct a linear state space. Therefore, any system whose behavior is describable as a system of such differential equations can be modeled in this way.

3.3 UAV Dynamics

Given a state-space decomposition, there are a few ways that the dynamics of a system can be simulated. One way (a rather simplistic way) is to use some small timestep Δt and the definition of the derivative (i.e., $\dot{x} = \lim_{\Delta t \to 0} \frac{f(x + \Delta t) - f(x)}{\Delta t}$) to estimate the evolution of the system (for the time series $t = 0, t = \Delta t, t = 2\Delta t, ...$) for which the initial conditions (i.e., the initial state of the system) are known, using linear interpolation:

$$
x_{t+\Delta t} \approx x_t + (\Delta t)\dot{x}_t = x_t + (\Delta t)(Ax_t + B\eta_t) \tag{3.6}
$$

As one might imagine, for any linearizable system, this modeling technique becomes increasingly precise as the timestep Δt decreases; errors due to linear interpolation disappear in the limit as $\Delta t \to 0$.

The trouble with simulations based on linearizations of the state space composition (as described above) is that, for any value of Δt, there exists some small inaccuracy in each step of the simulation, and this effect will be compounded over a large number of timesteps. Reducing the size of the timestep improves the accuracy of a simulation but increases the computational burden. Alternatively, it is possible in many cases to solve the relevant initial-value problems of the ordinary differential equations governing a system.

The flight dynamics of a UAV, for instance, can be simulated using the state-space decomposition method to model the kinetics of the airframe. In particular, we can apply a six-degree-of-freedom equation of motion derived from Newton's second law using an inertial- and a body-reference frame (Stengel, 2004). The full aircraft state consists of the 12-dimensional vector $x_a = [u\ v\ w\ x_r\ y_r\ z_r\ p\ q\ r\ \phi\ \theta\ \psi]^T$ where u, v, w, and p, q, r are the UAV velocities and angular rates in the reference frame of the UAV body, respectively, and x_r, y_r, z_r, and ϕ, θ, ψ, are the UAV translational and angular positions in the terrestrial inertial frame, respectively. The body state accelerations, denoted by X_b, Y_b, Z_b, L_b, M_b, and N_b are a function of the available thrust, and of the aerodynamic force and moment coefficients produced by the controls for the present aircraft state and wind field. The model estimates low-angle-of-attack Mach effects, power effects, and moments and products of inertia by using available full-scale wind tunnel data and physical characteristics, according to the methods described in Stengel (2004). The moments of inertia I_{xx}, I_{yy}, I_{zz}, and product of inertia I_{xz} are estimated using simplified mass distributions, and are held fixed at all times. Then, using the classical aircraft angles definitions and coordinate transformations described in Stengel (2004), the following UAV equation of motion

can be obtained:

$$
\begin{aligned}
\dot{u} &= X_b + g_{_x} + rv - qw \\
\dot{v} &= Y_b + g_{b_y} + pw - ru \\
\dot{w} &= Z_b + g_{b_z} + qu - pv \\
\dot{x}_r &= u\cos\theta\cos\psi + v(\sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi) \\
&+ w(\cos\phi\sin\theta\cos\psi - \sin\phi\sin\psi) \\
\dot{y}_r &= u\cos\theta\sin\psi + v(\sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi) \\
&+ w(\cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi) \\
\dot{z}_r &= -u\sin\theta + v\sin\phi\cos\theta + w\cos\phi\cos\theta \\
\dot{p} &= \frac{q}{(I_{xx}I_{zz} - I_{xz}^2)}\{I_{zz}L_b + I_{xz}N_b - p[I_{xz}(I_{yy} - I_{xx} \\
&- I_{zz})] + r[I_{xz}^2 + I_{zz}(I_{zz} - I_{yy})]\} \\
\dot{q} &= \frac{(M_b - pr\,(I_{xx} - I_{zz}) - I_{xz}\,(p^2 - r^2))}{I_{yy}} \\
\dot{r} &= \frac{q}{I_{xx}I_{zz} - I_{xz}^2}\{I_{xz}L_b + I_{zz}N_b + r[I_{xz}(I_{yy} - I_{xx} \\
&- I_{zz})] + p[I_{xz}^2 + I_{xx}(I_{xx} - I_{yy})]\} \\
\dot{\phi} &= p + (q\sin\phi + r\cos\phi)\tan\theta \\
\dot{\theta} &= q\cos\phi - r\sin\phi \\
\dot{\psi} &= \frac{q\sin\phi + r\cos\phi}{\cos\theta}
\end{aligned}
\tag{3.7}
$$

The aircraft control inputs consist of the throttle δT, the elevator δE, the aileron δA, and rudder δR, i.e., $u_a = [\delta T\ \delta E\ \delta A\ \delta R]^T$. As shown in Ferrari and Stengel (2002), the UAV can be fully controlled by means of a reduced state vector $\mathbf{x}_{UAV} = [V\ \gamma\ q\ \theta\ r\ \beta\ p\ \mu]^T$, which is formulated in terms of the aircraft speed V, sideslip

angle β, and path angle γ, where

$$V = \sqrt{u^2 + v^2 + w^2} \tag{3.8}$$

$$\beta = \sin^{-1}(v/V) \tag{3.9}$$

$$\gamma = \sin^{-1}(-w/V) \tag{3.10}$$

and in terms of the bank angle μ, defined in Stengel (2004). All of these variables are specified individually in Appendix A.

3.4 IR Sensor Modeling

The field-of-view (FOV) of the onboard IR sensor is assumed to be a closed and bounded subset of a Euclidian space, $\mathcal{S} = [0,\ L_{IR}]^2 \subset \mathbb{R}^2$, with the square geometry illustrated by the grey area in Figure 3.1. It can be easily shown using planar geometry that the size of the FOV is a function of the aircraft altitude $H = -z_r$,

$$L_{IR} = H \sin \theta_{IR} \tag{3.11}$$

where z_r is defined positive downward by convention (Stengel, 2004), and θ_{IR} is the sensor's aperture angle. In this thesis, it is assumed that θ_{IR} is held constant, and that the orientation of the IR sensor is fixed with respect to the UAV flight direction, but that it always points perpendicularly towards the ground. It follows that the position and size of the FOV are a function of time, $\mathcal{S} = \mathcal{S}(t)$, and change based on the aircraft trajectory or path. For simplicity, it is assumed that the centroid of $\mathcal{S}(t)$ coincides with the UAV coordinates in inertial frame, $x_r(t)$ and $y_r(t)$ at any time t.

As illustrated in Figure 3.1, the position and geometry of the FOV determine which regions of the minefield can be measured by the airborne IR sensor. The IR sensor measurements are influenced by its height above the ground (H), and by

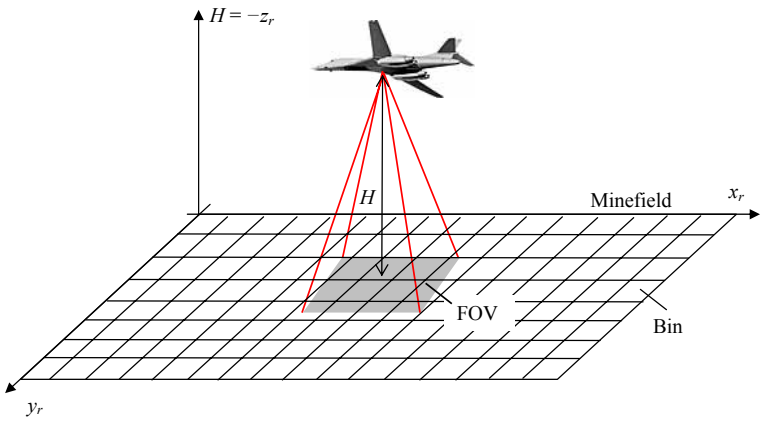


FIGURE 3.1: Problem description.

the environmental conditions in the minefield. A two-dimensional grid is superimposed on the minefield dividing it into unit-square cells. Soil characteristics, vegetation, and time-varying meteorological conditions, modeled according to MacDonald (2003); Dam (2003), are assigned to each cell, either at random or at user-specified positions. Buried targets are modeled as anti-tank mines (ATM), anti-personnel mines (APM), unexploded ordnance (UXO), and clutter objects (CLUT) that are sampled and reproduced using the Ordata Database (Explosive et al., 2006), which contains over 5,000 explosive items and 3,000 metallic and plastic objects that resemble anti-personnel mines. Each target i occupies one or more cells in the minefield depending on its size z_i, and is characterized by a depth d_i, and shape s_i (Table 3.1). The IR sensor mode, v_{IR}, is given by the UAV altitude (H) in km, which is discretized in a set of m possible values $\{a_1, \ldots, a_m\}$. At any given time, the space of admissible values of v_{IR} depends on the UAV speed, and is known from aircraft flight envelope. The aircraft flight envelope, denoted by $\mathcal{E}$, is the set of altitudes and velocities for which the aircraft can be trimmed (an example is shown in Figure 4.1). The envelope's boundary is designed by considering the stall speed, the thrust/power required and available, compressibility effects, and the maximum allowable dynamic pressure to prevent structural damage Stengel (2004).

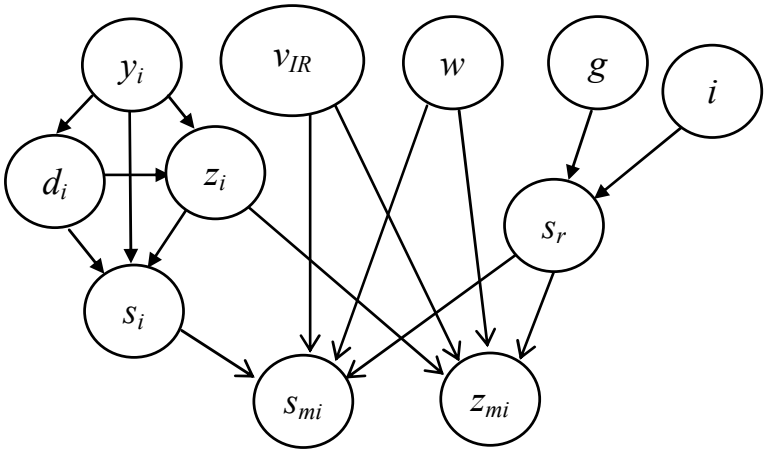


FIGURE 3.2: BN model of IR sensor (taken from Cai and Ferrari (2009)).

Table 3.1: IR Sensor Variables and Environmental Conditions

Symbol:	**Nodes:**	**Range:**
y_i	Target classification	{mine (1), not mine (0)}
v_{IR}	IR mode	$\{a_1, \ldots, a_m \mid a_h = h/0.2km\}$
E	Soil moisture (%): s_r	{dry [0, 10], wet (10, 40], saturated (> 40)}
	Soil composition: s_c	{very-sandy, sandy, high-clay, clay, silt}
	Soil uniformity: s_u	{yes, no}
	Vegetation: g	{no-vegetation, sparse, dense}
	Weather: w	{clear, overcast, raining}
	Illumination: i	{low (7-10 a.m. and 6-9 p.m.), medium (10-1 p.m.), high (1-6 p.m.)}
F_i	Depth (cm): d_i	{surface [0], shallow-buried (0, 12], buried (12, 60], deep-buried (> 60)}
	Size (cm): z_i	{small (2, 13], medium (13, 24], large (24, 40], extra-large (> 40)}
	Shape: s_i	{cylinder, box, sphere, long-slender, irregular}

An IR sensor detects anomalies in infrared radiation and, based on its height above the ground, builds an image of the FOV that includes measurements of shape and size for shallowly buried targets. Because they rely on temperature variations, their performance is highly influenced by illumination, weather, vegetation, and soil properties. As shown in Ferrari and Vaghi (2006), an IR sensor can be modeled by the Bayesian network (BN) in Figure 3.2, based on data and on the IR working

principles and detailed studies of Agema Thermovision 900 sensors (Dam, 2003). In particular, measurement errors are estimated by determining the spatial resolution of the sensor wth respect to the ground (based on the sensor's angular resolution and H). The ratio of resolution to object size is used to categorize the expected sensor performance into a small number of states, each of which has a prescribed conditional probability distribution for what the sensor will see (with respect to F_i), given the reality of what is on the ground. The category of sensor performance is then used to determine a probability distribution for the sensor measurements. In addition, limiting parameters of the sensor are incorporated into the model using cutoffs (e.g., a maximum depth below which nothing will be detected).

All BN nodes represent variables that influence the IR measurement process, and are defined as shown in Table 3.1. The IR BN model approximates the joint probability mass function (PMF) underlying the IR sensor measurements in terms of the recursive factorization

$$\begin{aligned} P(v_{IR}, E, M_i, F_i, y_i) &= P(M_i \mid v_{IR}, E, F_i) P(F_i \mid y_i) \\ &\times P(y_i) P(v_{IR}) P(E), \quad \forall i \end{aligned} \tag{3.12}$$

where $F_i = \{d_i, z_i, s_i\}$ is the set the features of the i^{th} target, $M_i = \{d_{m_i}, z_{m_i}, s_{m_i}\}$ are the *measured* target features extracted from sensor images, and y_i denotes the i^{th} target classification with the range $\mathcal{Y} = \{\text{mine, not mine}\}$. In this thesis it is assumed that the environmental conditions E_i are constant and uniform everywhere in $\mathcal{W}$, but are possibly unknown. The factors in (3.12) are conditional PMFs given by the BN conditional probability tables (CPTs) (see Jensen (2001) for a comprehensive review of BNs). By this approach, non-Gaussian sensor models can be obtained and used for sensor planning, as shown in Chapter 4.

As shown in previous research (Ferrari and Vaghi, 2006; Cai and Ferrari, 2009),

when a sensor is installed on a mobile platform, the measurement gathering process can be modeled as a Markov decision process (MDP), under proper assumptions. Although the MDP transition probability matrix of the UAV-IR system could potentially be obtained from the nonlinear dynamic equation (3.7), the BN sensor model, environmental maps, and weather forecasts, it would be computationally prohibitive to determine it for every minefield, weather, UAV, and IR sensor characteristics. Therefore, the goal of this thesis is to develop a Q-Learning technique that can learn the UAV-IR guidance policy from the sensing reward, without explicit knowledge of the transition probability matrix. By this approach, the same guidance algorithm can be applied to different airborne sensors and minefields, without redesigning the algorithm or modeling every system component.

4

Q-Learning Approach to the Treasure-Hunt Problem

4.1 Overview

The problem of determining the optimal sensor path for searching and classifying hidden targets, known as *treasure hunt problem*, was first formulated as an MDP in Ferrari and Cai (2009). An effective Q-learning technique for solving treasure hunt problems was presented in Cai and Ferrari (2008), and demonstrated through the benchmark problem of the game of CLUE®. In this thesis, Q-learning is applied to the new UAV-IR demining problem described in Chapter 3, which can be viewed as a new application example of treasure hunt. Since the sensor is installed onboard a UAV, the UAV path determines what cells can be intersected by the sensor's FOV, and measured by the IR sensor at any time.

Given the UAV dynamic model described in Chapter 3, it is possible to estimate a time-dependent, stochastic reward function for any feasible flight path. However, if mines are assumed to be randomly placed in the ROI according to a uniform probability density, then the reward per distance traveled along the ground will be

affected only by the altitude at which the UAV operates; low altitudes suffer from a small FOV, and high altitudes suffer from progressively poor detector and classifier performance. For simplicity, this thesis assumes that the aircraft maintains a straight flight path with respect to the ground, and that it flies at a constant velocity within the flight envelope shown in Figure 4.1. These simplifications make it possible to optimize the flight path with respect to one dimension: altitude.

4.2 Definition of Q-Learning State and Control Vectors

As a first step, the aircraft dynamics (3.7) are evaluated at N equally-spaced discrete points in time, $t_k = t_0 + k\Delta t,\ k = 0, \ldots, (N-1)$, over the interval $[t_0,\ t_f]$, where $\Delta t = (t_f - t_0)/N$ is the discretization interval. Between any two points in time, the control is assumed to be piecewise-constant and the UAV dynamics (3.7) are integrated by a 3^{rd} order Runge-Kutta integration routine (Stengel, 1986, pg. 77). In order to apply the Q-Learning technique in Chapter 2, the MDP state x_k must be observable, and may be defined as a subset of the full system state. Thus, based on the problem formulation in Chapter 3, the $x_k = [x_r(t_k)\ y_r(t_k)\ z_r(t_k)\ V(t_k)]^T$, since this subset of state variables determines the IR-sensor FOV's size, position, and orientation at t_k. The FOV's size and position, in turn, determine the hidden target characteristics in cell i, denoted by the set $\zeta_i = \{d_i, z_i, s_i, y_i\}$, and the hidden environmental conditions E_i, through the subset of cells that are intersected by the FOV, with index set I_k. The environmental conditions may or may not be known depending on the scenario. The set ζ_i of hidden variables can be estimated only after the FOV has intersected cell i.

The objective of the optimal greedy guidance policy, $u_k = \pi^*(x_k)$, is to compute the next UAV position, at t_{k+1}, such that the IR sensing performance over time is maximized. Therefore, the control vector is defined as the next waypoint, i.e., $u_k = [x_r(t_{k+1})\ y_r(t_{k+1})\ z_r(t_{k+1})]^T$, where $z_r(k+1)$ determines the next IR sensor mode

$v_{IR}(t_{k+1})$. In this approach, the Q-learning technique is said to provide an outer-loop algorithm, whose output can be followed by means of the inner-loop proportional-integral (PI) controller described in Ferrari and Stengel (2002).

4.3 Definition of Reward

After the IR measurements are obtained from all the cells inside the sensor's FOV, the IR BN model (3.12) is used to estimate (or infer) the target classification based on the *measured* target features (M_i) extracted from sensor images, the sensor mode v_{IR} and, possibly, known environmental conditions. In this thesis, BN inference is performed by a junction-tree algorithm available through the Matlab® BN-Toolbox commands *jtree_inf_engine*, *enter_evidence*, and *marginal_nodes* (Murphy, 2007). The inference algorithm provides the posterior PMF $P(y_i, d_i, z_i, s_i \mid v_{IR}, d_{i_m}, z_{i_m}, s_{i_m}, E_i)$, and the target classification is estimated by choosing the value of highest posterior probability, i.e.:

$$\hat{y}_i = \arg\max_{y_i^k \in \mathcal{Y}_i} P(y_i \mid v_{IR}, d_{i_m}, z_{i_m}, s_{i_m}, E_i) \tag{4.1}$$

The estimated target classification is then accompanied by the certainty level (CL), denoted by $c_i = P(\hat{y}_i \mid v_{IR}, d_{i_m}, z_{i_m}, s_{i_m}, E_i)$, which represents the confidence in the estimated value and, for a binary variable, is $0.5 < c_i \leqslant 1$.

Let y_i^* denote the actual classification of the target in cell i. Then, the classification error defined as,

$$e_i = |\hat{y}_i - y_i^*| \tag{4.2}$$

also is binary, and takes a value of 0 when the estimate is correct, and a value of 1 when the estimate is incorrect. If the estimate is correct, a higher CL is desirable, but if the estimate is incorrect, a lower CL is desirable because it indicates that

the estimate is uncertain. Thus, the IR sensor performance is reflected in the classification error (4.2) and in the CL. Additionally, the sensor performance depends on application-specific objectives for deploying the UAV-IR. For example, in some applications it may be of interest to minimize the number of false alarms, whereas in others it may be of interest to find cells without targets, in order to determine a safe path through $\mathcal{W}$. In this thesis, the application's objectives are characterized by a discrete risk function defined as,

$$\rho_i = \begin{cases} w_1 & \text{if} \quad \hat{y}_i = 1,\ y_i^* = 0 \text{ (false alarm)} \\ w_2 & \text{if} \quad \hat{y}_i = 0,\ y_i^* = 1 \text{ (misclassification)} \\ w_3 & \text{if} \quad \hat{y}_i = 1,\ y_i^* = 1 \text{ (mine detection)} \\ w_4 & \text{if} \quad \hat{y}_i = 0,\ y_i^* = 0 \text{ (void-cell detection)} \end{cases} \tag{4.3}$$

where $w_1, \ldots, w_4$ are user-defined positive constants that weigh the relative importance of the four cases listed in (4.3). If in a mission a false alarm poses a much greater risk than a misclassification, then $w_1 >> w_2$. If, in addition, it is of secondary importance to correctly classify mines, then $w_1 >> w_3 >> w_2, w_1$, and so on.

Then, the immediate reward from cell i can then be defined as a tradeoff between the measurement value and error,

$$r_i = W_v[(1 - e_i)]c_i\rho_i - W_e(e_i c_i \rho_i) \tag{4.4}$$

where W_v and W_e are user-defined positive constants that represent the desired tradeoff between the measurement value of obtaining correct classifications of mines or void cells, and the measurement error of incorrectly classifying mines or false alarms. At every time t_k, the IR sensor obtains measurements from a set of cells in its FOV, $\mathcal{S}(t_k)$ and, thus, the total value of the immediate reward is

$$r_k = R(x_k, u_k) = \sum_{i \in \mathcal{S}(t_k)} W_v[(1 - e_i)]c_i\rho_i - W_e(e_i c_i \rho_i) \tag{4.5}$$

and can be computed from the IR sensor measurements and the actual target classification y_i^*.

In this thesis, the Q function is approximated by a feedforward sigmoidal NN

$$Q(x_k, u_k) = W_2\Phi(W_1\ [x_k^T\ u_k^T]^T + b_1) + b_2 \tag{4.6}$$

by means of the update rule (2.14) as the UAV explores the state and control spaces. The s-dimensional operator Φ represents one hidden layer of s log-sigmoidal functions of the form $\sigma(n) \equiv 1/(1+e^{-n})$. The NN weights $W_1 \in \mathbb{R}^{s\times(n+m)}$, $W_2 \in \mathbb{R}^{1\times s}$, $b_1 \in \mathbb{R}^s$, and $b_2 \in \mathbb{R}^2$, are determined by the Bayesian regularization algorithm ('trainbr' (MacKay, 1992a,b)). A training set for (4.6) is formed according to the Q-learning approach. As a first step, the Cartesian product of the state and control spaces $\mathcal{X}\times\mathcal{A}$ is discretized. With the state and control definitions in Section 4.2, this is achieved by discretizing the flight envelope (e.g., see crosses in Figure 4.1). As a second step, the rule in (2.14) is applied iteratively over the discrete time t_k, while exploring $\mathcal{W}$ (already discretized into cells) by flying the UAV at every feasible pair of altitudes and velocities.

After the reward (4.5) is evaluated for every pair of state and control values explored by the UAV-IR, the data can be used to learn the Q function using (4.6). Then, the optimal policy, $u_k = \pi^*(x_k)$, is determined by maximizing the learned Q function using the greedy rule in (2.13). The effectiveness of this approach is demonstrated in the next chapter, using the system models described in Chapter 3.

4.4 Epistemology of the Field

Training the NN requires some form of evaluation of a given state/action pair. Section 4.3 describes a specific way to quantify this mathematically, but not how to know the input data needed to determine ρ_i for any i that passes into the FOV.

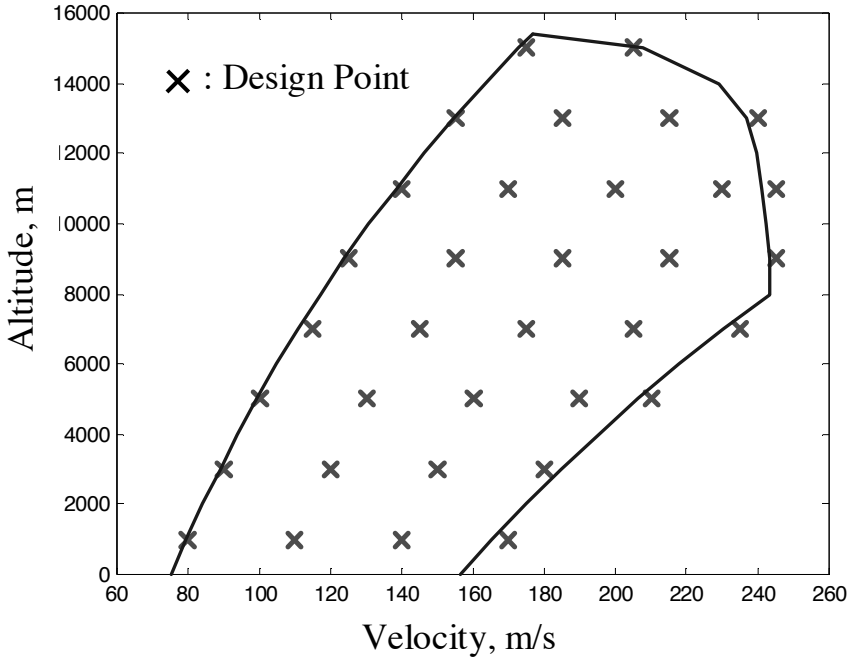


FIGURE 4.1: Aircraft Flight Envelope, taken from Ferrari and Stengel (2002).

This question is easy to answer if the NN that describes the Q-function is trained offline. In particular, the process would start by choosing a state-action pair and simulating it repeatedly within a given set of environmental conditions. Given the aircraft's observations over the training inputs, we can then assume that the ground truth is precisely known and score the state-action pair accordingly. This procedure can then be repeated for a number of state-action pairs.

For the special case of straight, level flight, this can be achieved by simulating a flight over a given field, with the altitude held constant as the ground-translation variables change. Given the aircraft's observations of the ROI, we can then assume that the ground truth is precisely known and score the state/action pair using Equation 4.5 and

$$r_f = \sum_{k \in \{0,1,\ldots,\frac{t_f - t_0}{\Delta t}\}} R(x_k, u_k) \tag{4.7}$$

It is worth noting here that, for the case of straight, level flight, ground speed is not considered as a factor. In the sensor simulation used for this thesis, ground speed does not have an effect on the performance of the sensor. However, speed can still have a (roughly) linear effect on a time-dependent scoring measure since higher speeds result in coverage of more new landmass per unit time. Using this model, then, one could optimize the time-dependent score for a given altitude by assuming that the aircraft flies at the highest speed possible for the given state-action pair. For the case of straight, level flight, this is determinable from Figure 4.1, and one can see that flying at higher altitude offers the advantage of a higher maximum airspeed. For the sake of this thesis, however, it is assumed that the aircraft flies at a uniform speed for any of the altitudes sampled. This is acceptable because there does exist a velocity interval of the flight envelope that is feasible for all the altitudes measured. Given a constant velocity, we then know that Equation 4.7 is related to a time-dependent scoring function through a simple proportionality constant that is invariable across all altitudes, which means that the relative evaluations of the different altitudes are described sufficiently.

Returning to the general case, the result of the training-set generation technique described above is a stochastic estimate of the relative performance of one state-action pair versus all the others. Using a procedure that is described further in Chapter 5, this can be transformed into a score $Q(x_k, u_k)$, which can in turn be used to batch-train the NN.

This offline-training policy gives us information about a control policy that we can expect to be near-optimal if we send the UAV to scout an unexplored but similar ROI. But what if we want to learn from experiences in the field? Offline learning can provide a strong initialization to guide an online learning algorithm in the early stages of exploration of the ROI, but, by definition, it will be blind to any dissimilarities between the new field and the field(s) used for training. Online training

requires a different training technique for the NN (incremental training instead of batch training), but, more critically, it requires a completely different feedback system.

The reason for this is because the offline-training system described above assumes that the ground truth for the entire ROI is determined with certainty and then used to score the performance of the sensor for various state-action pairs. But if the ground truth is available for the online-training case, then the UAV flight is pointless because its observations do not provide any new information about the ROI. However, if we can assume that some *subset* of the measurements we want to make is known, then state-action pairs can be scored using direct comparisons between the known information and the corresponding sensor measurements. In this case, we say that much of the dataset from the field is "unlabeled," and the exploration of the ROI becomes a Partially-Observable Markov Decision Process (POMDP).

Methods for evaluating POMDPs will not be covered in-depth in this thesis. However, to put things shortly, the context at hand would require us to develop some method of estimating the sensor's performance over the full set of observations rather than just the set whose ground truth is known. A simple model might assume that sensor will perform similarly over the unlabeled bins as it does over the labeled ones. A more sophisticated model would include some way of quantifying a PDF or PMF of the score, given the most recent performance measures.

4.5 Simplifying Assumptions

All of this forms a generalized framework that can be used to optimize a number of parameters of the flight path (within bounds of feasibility) by optimizing the aircraft control policy. For this thesis, a simplified version of the problem described in Chapter 3 and refined in Section 4.2 is used to demonstrate the efficacy of the Q-learning algorithm.

As mentioned in Section 4.1, one of the assumptions is that we are looking at straight, level flight. This gives us the luxury of optimizing the altitude of the aircraft by itself rather than over multiple parameters. Given the assumption that the flight path under consideration is being executed within the flight envelope (Figure 4.1), this also means that we can look directly at the impact that *being* at a given altitude has over the scoring function, rather than optimizing the elevator controls directly. Extension of this methodology to higher-dimensional optimization problems is left to future research.

Another simplifying assumption is that we are using offline training to determine an optimal policy for future flights. The alternative would be online learning of the optimal policy, which requires incremental training of the NN that describes the Q-function. It would also require some tinkering to find a promising value of the exploration coefficient, ϵ, as well as some type of immediate feedback, such as the partial ground measurements described in Section 4.4.

The next chapter describes the results obtained from the application of Q-learning to this simplified problem.

5

Results

5.1 Implementation and Simulation

The UAV is simulated by integrating the ODE in Equation 3.7, using the Matlab® program FLIGHT, developed in Stengel (2004). This program simulates the aircraft flight and determines the trim conditions for any nominal altitude and velocity that lie inside the aircraft flight envelope $\mathcal{E}$, illustrated in Figure 4.1. In this thesis, it is assumed that the UAV maintains steady-level longitudinal flight along a diagonal path, and that the Q-guidance algorithm must determine the optimal nominal altitude H^*, based on the UAV-IR sensing reward obtained by flying over a minefield $\mathcal{W}$ with an arbitrary but homogeneous set of geometric and environmental conditions, and known targets.

The UAV-IR sensor, and the minefield $\mathcal{W}$ are simulated numerically, using the mathematical models described in Chapter 3. In the simulation, as soon as the FOV of the IR sensor, $\mathcal{F}$, intersects a cell containing a target, measurements are reproduced and deteriorated based on the target features, the sensor's mode and working principles, and the environmental conditions in the cell (Ferrari and Vaghi,

2006). As an example, the set of cells measured by the UAV-IR sensor at a sample moment in time, and an altitude $a_8 = 1.6$ km is shown by purple bins in Figure 5.1. Red bins represent undetected targets (mines or clutter), green bins represent detected targets, and black bins represent obstacles.

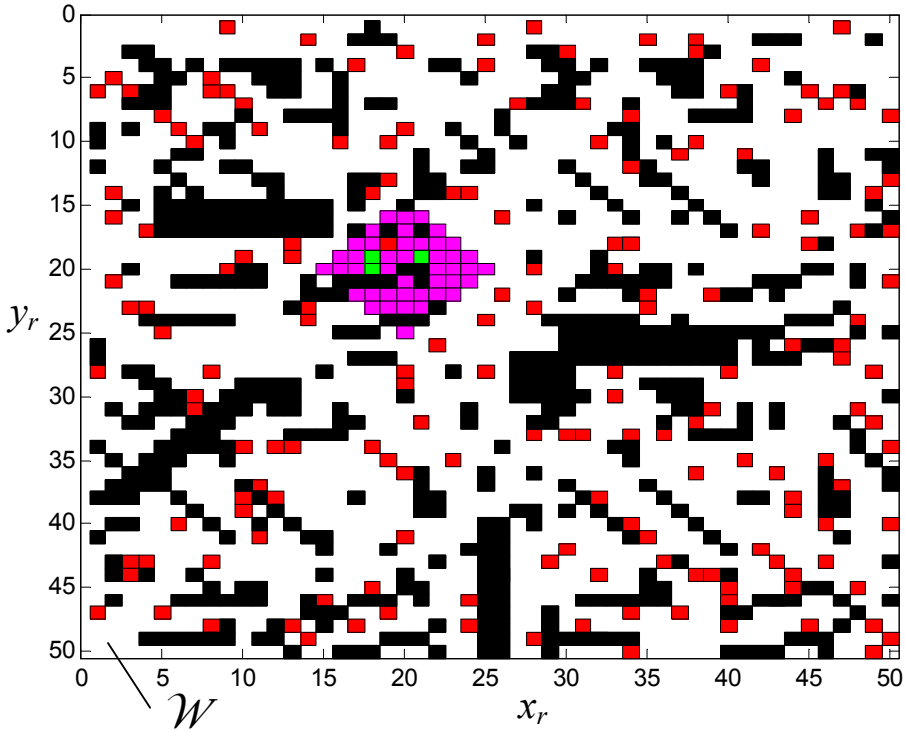


FIGURE 5.1: Instantaneous UAV-IR sensor's FOV at $a_8 = 1.6$ km.

In this thesis, the environmental conditions in Table 3.1 are assumed to be uniform in $\mathcal{W}$ and, therefore, the UAV-IR can learn the Q-function by flying over $\mathcal{W}$ at various altitudes and velocities in $\mathcal{E}$. At higher altitudes, the IR measurements are typically less accurate, but more cells are measured because the FOV is larger. At lower altitudes, the FOV is smaller, translating to fewer cells being measured by the sensor, but the IR measurements are more accurate. As an example, the cells measured by the UAV-IR sensor along a path at $a_8 = 1.6$ km, and during the time interval $[t_0,\ t_f]$, are shown by the purple bins in Figure 5.2, using the same color legend used in Figure 5.1. For comparison, the cells measured along the same path

at $a_{32} = 6.4$ km is shown in Figure 5.3.

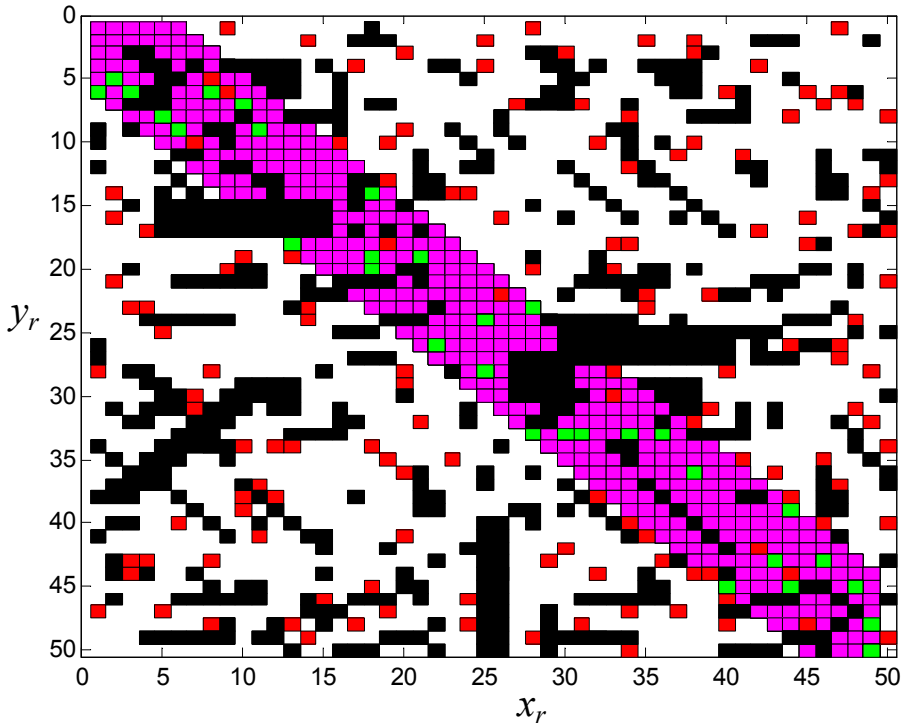


FIGURE 5.2: Cells measured by UAV-IR during $[t_0,\ t_f]$, at $a_8 = 1.6$ km.

As shown in Figure 5.7, the percentages of mines (red bins) and clutter-targets (yellow bins) that are correctly classified is higher at a_8, whereas the percentage of targets that were undetected (blue bins, labeled by "U") is higher at a_{32}. The CL (%) of the classifications, plotted on each target detected in Figure 5.7, also tends to be higher at lower altitudes. While the UAV-IR explores the state space, its measurements and the actual classification of the targets are used to compute the reward in (4.5), and to learn the Q-function from (2.14). In this thesis, the weights in the risk function (4.3) are chosen as $w_1 = 120$, $w_2 = 90$, $w_3 = 250$, and $w_4 = 1$ for false alarms, false negatives, correct mine classifications, and correct void-cell detections, respectively. This places the primary emphasis on correct mine detections, as might be the case for a mission focused on detecting mine clusters.

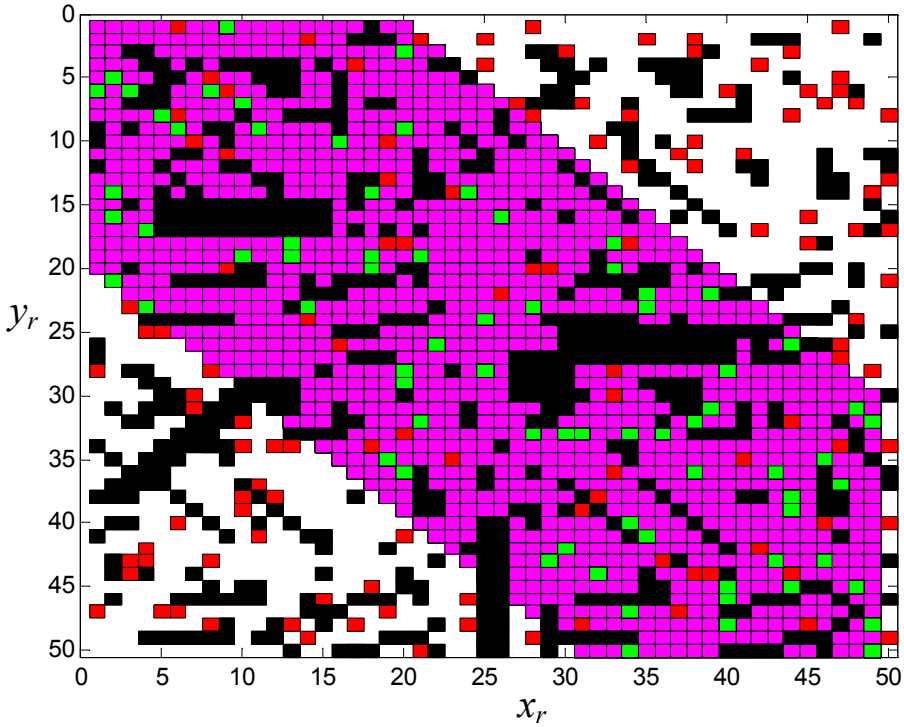


FIGURE 5.3: Cells measured by UAV-IR during $[t_0,\ t_f]$, at $a_{32} = 6.4$ km.

5.2 Stochasticity in the Sensor Model

When pushing the limits of sensor performance, one will often come across the problem of noisy signals; in the real world, sensor data may not be the result of a simple deterministic relationship between the ground truth and the known variables. Therefore, though the given ROI is identical with respect to environmental conditions and target placement for all flights performed for this thesis, almost no two flights over the ROI will be the same. This is because of the probabilistic nature of the sensor simulation, which can be summarized as follows.

Given the ground truth of a situation, there are several ways in which the detector can be expected not to see a target at all. For instance, if a target lies deeper than 10 m underground, there will be no detection. Alternatively, if it lies more than 2 m underground but the weather is rainy and the local vegetation is dense, there will be no detection.

Assuming an object (either a mine or a mine-like object) is detected, an image contrast will be assigned to it depending on the specific combination of environmental conditions. This designation is selected from the set of "poor," "low," "medium," "high," and a null value. In order to produce the data used in this thesis, a minefield was generated with homogeneous environmental conditions; those conditions were clear skies, high illumination, clay soil, dense vegetation, and 30 percent ground moisture (considered "wet," as opposed to"dry" or "saturated"). Given these conditions, the image contrast is labeled as "low" for all detectable targets on the field. The image contrast takes a null value for all empty bins and undetectable targets.

The simulation model is intended to mimic the behavior of an Agema Thermovision 900 infrared sensor. This sensor is known to have a spatial resolution of 1.5 milliradians; given this number, the altitude of the UAV, and a small-angle approximation, the sensor resolution ξ_{IR} is given by

$$\xi_{IR} = H * 0.0015 \tag{5.1}$$

Given the value of ξ_{IR}, noise is introduced into the measured size and shape variables (s_{mi} and z_{mi}, respectively). In the case of the shape measurement, this is done through a series of steps. First, a coefficient for the shape error is generated in accordance with Table 5.1. Then, for each target, a random number is generated from a uniform distribution on $[0, 1]$. This is a luck factor, intended to introduce stochasticity to the sensor measurements. If it happens to exceed the shape error coefficient, then the correct shape will be reported to the detector. If not, the detector will receive a signal based on the difference between the two and the actual grund truth, z_i. For instance, if the luck factor is lower than the shape error coefficient, but happens to be within 0.2 of it, then a cylindrical shape will be reported as a box shape.

Given that object size is taken from a continuous state space rather than a small set of enumerated shapes, it is possible to create noise more elegantly. For this thesis, a program was used in which the measured size, s_{mi}, is generated by adding the real size, s_i, using the following equation

$$s_{mi} = s_i + 0.01(B + \sigma * Random([0, 1])) \tag{5.2}$$

where B represents a deterministic measurement bias, and σ represents a scaling coefficient for a random variable. For a given target, values for B and σ are assigned based on image resolution and target size, in much the same way that the shape error coefficient was determined. The multiplier of 0.01 can be tweaked to simulate more or less noise, but it is not meant to depend on environmental variables.

Table 5.1: Relationship Between Actual Target Size and Shape Error Coefficient

	Poor	**Low**	**Medium**	**High**
$z_i \geqslant 8\xi_{IR}$	0.5	0.4	0.2	0.1
$8\xi_{IR} > z_i \geqslant 5\xi_{IR}$	0.6	0.5	0.3	0.15
$5\xi_{IR} > z_i \geqslant 3\xi_{IR}$	0.7	0.6	0.4	0.25
$3\xi_{IR} > z_i$	0.8	0.7	0.45	0.3

These noisy signals are plugged into the sensor BN described in Chapter 3, in order that a target classification y_i may be determined as the most likely explanation (MLE) of the inputs, based on information from the ORDATA database. Now that the framework has been developed for modeling the target environment and the measurements taken of it, the Q-learning stochastic optimization method will be described explicitly.

5.3 Score Estimation

For the special case of offline training treated here, we want to find a strong estimate of the expected reward for a wide variety of the altitudes contained in the flight enve-

lope, and then use these numbers to estimate the Q-value for all altitudes contained in the flight envelope. The phrase "expected reward" refers to a modified concept of the instantaneous reward r_k described by Equation 4.5. In particular, it is intended to provide an estimate of the expected of the reward returned per unit time.

More explicitly, for the given ROI consider the set of all measured cells for a given altitude Λ_H, and assume that the sample flight path of the UAV with respect to $\mathcal{W}$ is does not change from one trial flight to another. Then the scoring estimate r'_H is given by

$$r'_H = \sum_{i \in \Lambda_H} W_v[(1 - e_i)]c_i\rho_i - W_e(e_i c_i \rho_i) \tag{5.3}$$

Equation 5.3 is nearly identical to Equation 4.5 except that the domain of the sum includes the rewards for an entire flight (i.e., over the sets highlighted in Figures 5.2 and 5.3) rather than just the reward at a particular instant.

The benefit of this scoring function within the context of offline learning (as opposed to adding or averaging all the values of r_k for a given flight across the field) is that it avoids double-counting cell measurements. Given that the sensor model does not include a means to improve sensor performance for longer exposure times, Equation 5.3 provides a way to avoid granting high altitudes with unjustifiably high scores for getting a right answer counted repeatedly, or low scores for having missed detections and incorrect classifications counted more times than they would be at lower altitudes.

Given the set of measures $r'_H, H \in \{1, 2, ..., 50\}$, one can try to filter some of the measurement noise by repeating the measurement several times and averaging the measurements. For this thesis, the measurements were repeated ten times, and the average value of r'_H is designated by $R'(H)$. Given a constant flight velocity for each altitude tested, $R'(H)$ acts as an estimate of the expected reward per unit time for

a flight at a given altitude over a field, with the exception of a constant factor that does not vary with respect to H. This factor is equivalent to the amount of time needed to cross the field at the measurement velocity; since it is held constant for all of the measured altitudes, it does not affect the relative score of each altitude for the given environmental conditions and ROI.

For one set of these measurements (i.e., the estimates of expected reward $R'(H)$), the raw reward estimates are shown in Figure 5.4. An NN function approximation is also shown in that figure, but only for purposes of demonstrating the difficulty of filtering this noisy data.

5.4 Q-Learning Implementation

The Q-learning algorithm used for this thesis is based on the application of Equation 2.14 to the reward function described in Chapter 4. Specifically, the algorithm's actions followed these steps:

1. Simulate a series of flights at different altitudes above the ROI. Keep the flight path constant, and vary only the altitude between flights.

2. For each possible altitude, simulate ten passes along the same flight path and record all sensor outputs.

3. Score each flight (to produce r'_H) and average the scores for flights at the same altitude to produce $R'(H)$.

4. For each altitude mode h, initialize $V_{old}(h) = R'(h)$.

5. Update: For each altitude, set $V_{new}(h) = R'(h) + \gamma \max_{u \in \mathcal{A}} V_{old}(h+u)$, where $u \in \mathcal{A} = \{0, 1\}$ for the lowest altitude tested, $u \in \mathcal{A} = \{-1, 0\}$ for the highest altitude tested, and $u \in \mathcal{A} = \{-1, 0, 1\}$ for all altitudes in between.

6. Further update: $V_{old}(h) \leftarrow (1-\alpha)V_{old}(h) + \alpha V_{new}(h)$. Repeat updates a large number of times (e.g., 10^5).

7. By Equation 2.11, this will provide the value of Q for each altitude, maximized over the possible control actions (ascend, descend, remain level).

8. The Q values can be interpolated over the continuous state space described by the flight envelope (within the bounds of the trials made) using a 2-layer feedforward NN.

9. For an aircraft in any given state within the domain of the resulting curve, the optimal control policy is to approach the nearest peak as quickly as possible.

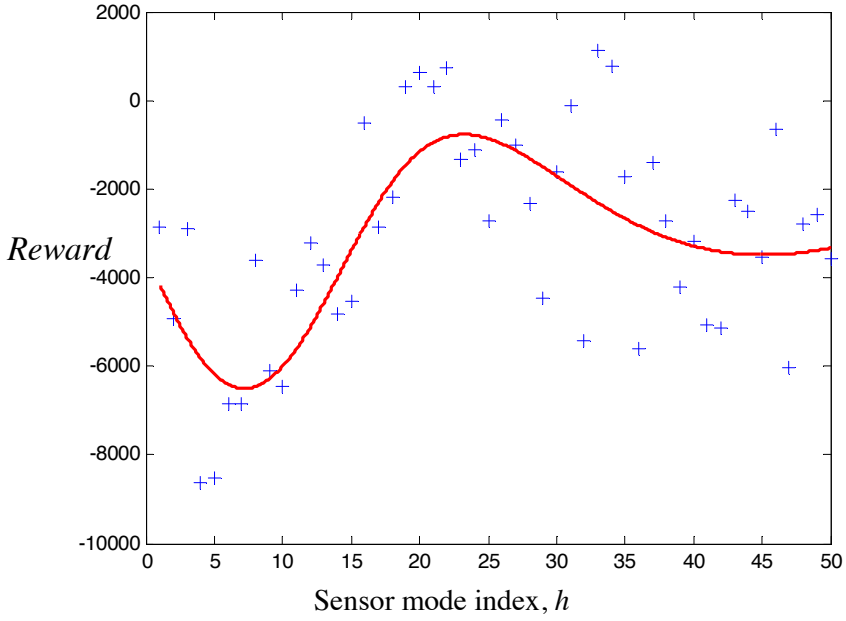


FIGURE 5.4: Raw score data (0 iterations performed, yielding reward scores, not Q-scores) as a function of altitude. An NN function approximation (10-neuron hidden layer with ≈300 epochs of Bayesian-regularized training) is able to detect some pattern, but that pattern is extremely noisy, and the reward function is myopic.

The two-fold function of value iteration is apparent when comparing Figure 5.4 with Figure 5.8. One such function is to filter out noise. The other is simple policy

determination; given a tradeoff coefficient γ that quantifies the value of future rewards relative to immediate gains, this value iteration procedure tells us when it is or is not beneficial to move to the altitude that globally maximizes the iterated value function. The reason is that the optimal policy is always to choose the action that maximizes $Q(x_k, u_k)$ for the given state x_k at timestep k. Here, the only choices are to ascend or descend by some minimal increment, or to remain level. Ergo, the Q-learning algorithm transforms the policy determination problem into a question of hill climbing.

The value of α is primarily important for purposes of online Q-learning. For this offline procedure, it is sufficient to set it equal to 1 in order to facilitate quick convergence. If it is instead set to another value > 0, then, for the case of offline training, the *shape* (i.e., all the information needed for determining the optimal policy) of the resulting curve will be practically indistinguishable from the case where α equals 1 for a sufficiently large number of iterations. Case in point, one may compare Figure 5.5 with Figure 5.9. Both are interpolated by similar NNs (Bayesian-regularized with a 10-neuron hidden layer, trained for a few hundred epochs). For the former case, the y-axis appears to be scaled differenly, but the two graphs have identical shapes.

5.5 Determining the Optimal Policy

Figure 5.8 shows the Q-value for each altitude. One might object that, according to the Bellman Equation (Equation 2.11), Q is a function of state and action, not the state alone. As noted in the figure caption, however, the value on the y-axis is actually given by $\max_{u_k \in \mathcal{A}} Q(h_k, u_k)$. The reason that it is fair to label this as the Q value for a given state is because the expectation of the instantaneous rewards, based on the methods used for this thesis, is not dependent on the current control value;

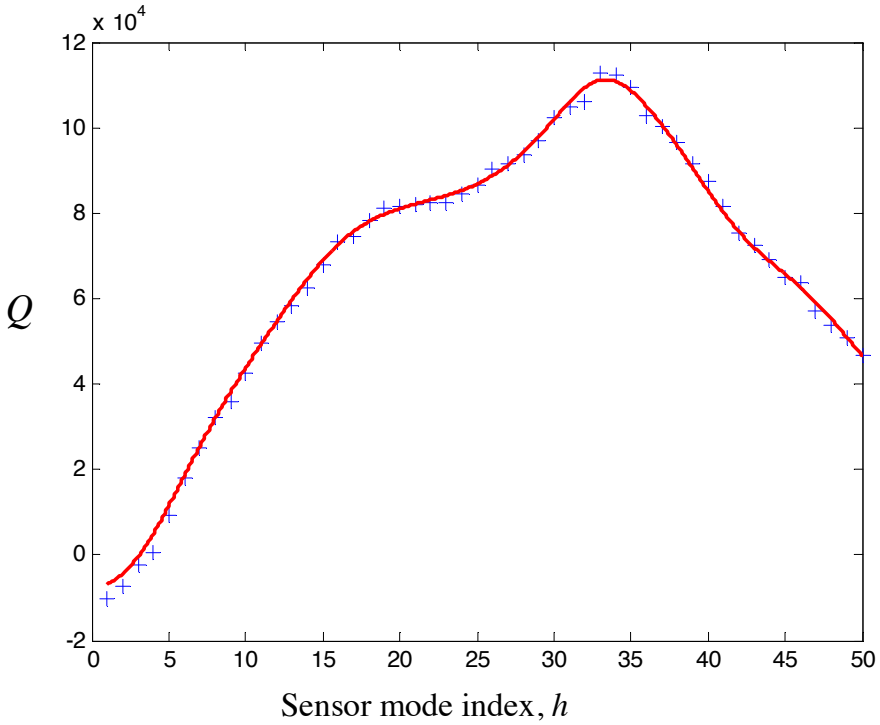


FIGURE 5.5: Plot of Q-values (given by $\max_{u_k \in \mathcal{A}} Q(h_k, u_k)$) for a reduced value of α (0.5 instead of 1), given $\gamma = 0.99$.

instead, the control value is relevant for purposes of determining $V(x_{k+1})$, but since $V(x_{k+1}) = \max_{u_{k+1} \in \mathcal{A}} Q(x_{k+1}, u_{k+1})$, it is acceptable in this context to measure Q as $\max_{u_k \in \mathcal{A}} Q(h_k, u_k)$.

Exactly how this information should be used depends on the exact context in which the UAV is operating. If the operators know *a priori* that the aircraft will fly over a field with environmental conditions very similar to the ones over which the IR sensor was tested to create this graph, then the operator should have the aircraft cruise over the ROI at the global optimum of the interpolated graph, which happens here close to altitude mode $h = 34$, or about 6.8 km above the ground.

This case is not terribly interesting. If, however, the situation is that a UAV has entered an ROI of the type used for this training, then the use of the figure changes.

We might ask the question of whether it is worth passing through altitude modes whose expected returns are relatively low in order to reach an altitude mode whose expected reward is higher than that of the current state. This is where the value of the discount factor γ makes a difference to the optimal policy.

Equation 2.13 describes how the optimal policy is determined from the Q-function. In the simplified, discretized state-action space (i.e., where $h_k \in \{1, 2, ..., 50\}$ and $u_k \in \{0, 1\}$ for $h_k = 1$, $u_k \in \{-1, 0, 1\}$ for $h_k \in \{2, 3, ..., 49\}$, $u_k \in \{-1, 0\}$ for $h_k = 50$), this tells us that, given an estimate of the Q-value of each of the altitude modes examined, the optimal policy requires us dictates that the UAV must ascend or descend towards the highest-valued neighbor available unless it already happens to be operating at a locally optimal altitude (i.e., an altitude for which $Q(h, -1) \leqslant Q(h, 0)$ and $Q(h, 1) \leqslant Q(h, 0)$).

In the continuous case, the answer is not as simple. The Q-function interpolation provided by the NN tells us that if the slope of the Q-function is positive for the current altitude mode, then the UAV should ascend. If the slope is negative, the UAV should descend. This means that the optimal policy for the UAV is to hone in on a local maximum of Q.

In the simplified action ascend/descend/maintain level flight action space, the graph suggests a simplistic notion of moving toward a local minimum with the assumption that each move will be made in some standard time period Δt. This corresponds to a model in which an aircraft quickly establishes a standard rate of climb, moves to the altitude with the locally optimal Q-value, and quickly levels off the aircraft. But the idea of considering a continuously valued state space begs questions about a continuously valued action space $\mathcal{A}$ as well as continuously valued time.

Consider, for instance, Figure 5.6. This figure was generated using a longitudinal simulation of a short-takeoff-and-landing (STOL) aircraft with an elevator whose

actions are determined by a state feedback controller, as described in Nelson (1998). The controller is built to maintain level flight against any unexpected displacements of the aircraft. The state being controlled is the displacement of the aircraft from that level flight. Initially, the model is set such that for an altitude of 3000 m, this displacement is zero. Then, at time t = 2 seconds, the displacement is artificially lowered to -100 m so that the new equilibrium altitude is 3100 m.

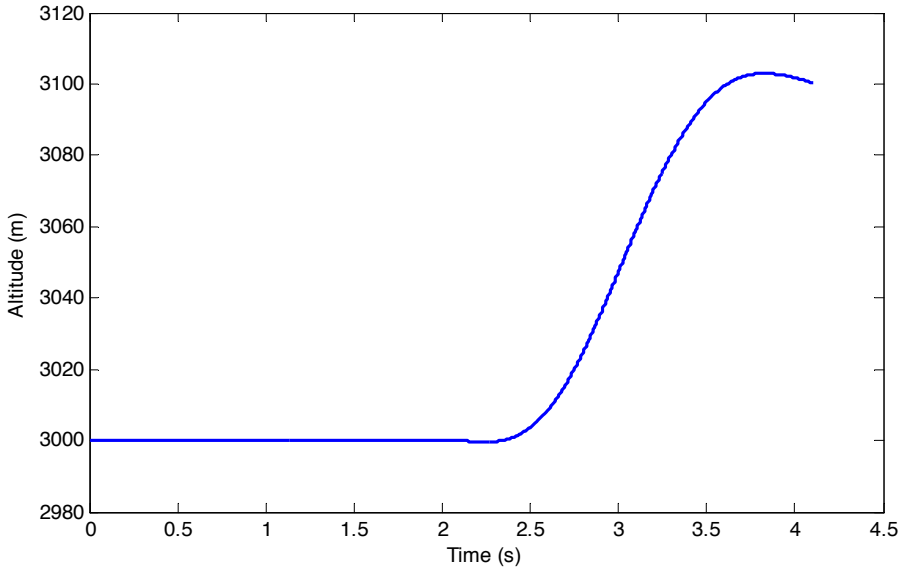


FIGURE 5.6: Altitude transition from 3000 m to 3100 m, based on simulation of an STOL vehicle with elevator controls determined by a state-feedback controller, as described in Nelson (1998).

The point of this graph is to demonstrate the type of altitude profile that may be expected if an aircraft controller is simply commanded to move the aircraft from one altitude to another, as is suggested above. Note the nonlinear ascent rates as the transition begins and ends. As the aircraft levels off, it even overshoots the target altitude by a small margin. These effects can be accounted for if the action space is mapped directly to a range of feasible elevator angles for the UAV and the model of the aircraft dynamics is used to determine at what altitude the aircraft will be flying

at time $t_0 + \delta t$, what its rate of climb or fall will be, the rate climb or fall, the rate of change of the aircraft pitch, etc.

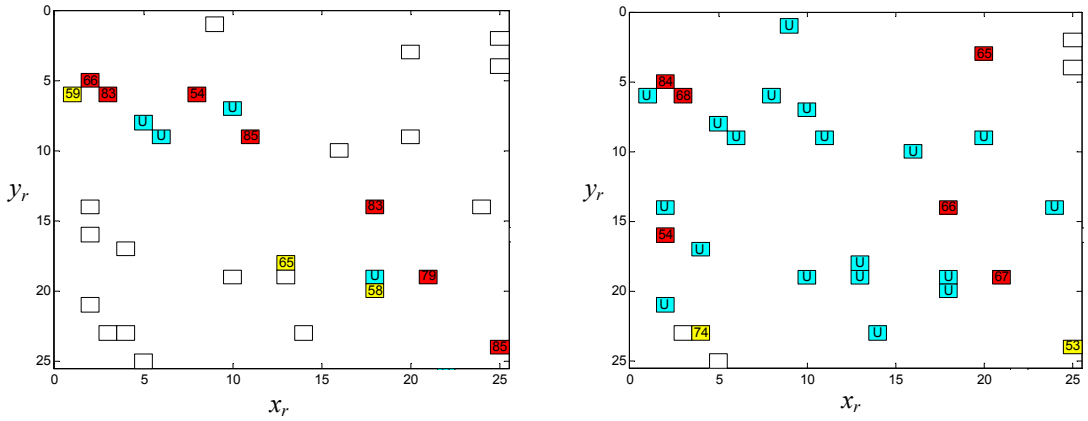
Note, however, that this approach increases the dimension of the state vector, thereby increasing computational burden. Note also that the graph shows a nearly straight line between the altitudes of 3020 m and 3080 m (i.e., the segment of the transition that is separated by the initial and final states by 20 m), and that the range of altitudes considered for this thesis is 9800 m wide. Because of this, it is assumed that the simplified transition model is sufficient to determine the optimal aircraft motions accurately, and that the precise elevator control sequence needed to make the transition can be determined using a state feedback controller like the one described in Nelson (1998).

5.6 Choosing γ

Figures 5.8 and 5.9 represent two different versions of the Q-function learned by UAV-IR, and approximated by the NN in Equation 4.6. Each is a nonlinear function of v_{IR}, derived from the same simulation dataset, with all the same weights in the risk function; the difference is that Figure 5.8 is based on a stepwise discount factor of $\gamma = 0.9$, whereas Figure 5.9 is based on a stepwise discount factor of $\gamma = 0.99$.

Once learning is completed, the greedy policy in (2.13) is computed by maximizing the learned Q-function. This policy provides the optimal sensor's altitude at time $t_{(k+1)}$ as a function of the present aircraft state (altitude and velocity) at t_k. In this study, the optimal altitude is found to be $H^* = a_{34} = 6.8$ km. Thus, as verified by the sensing performance comparison in Table 5.2, when the UAV-IR sensor flies at this altitude, it obtains the highest reward value, as specified by the risk function (Equation 4.3).

Choosing the discount factor γ depends on how much the aircraft operator values long-term results against possible short-term tradeoffs. Note that there is a local



(a) (b)

FIGURE 5.7: Classification results for the UAV-IR at $a_8 = 1.6$ (a), and at $a_{32} = 6.4$ km (b), during a sample time interval. The red boxes indicate a mine classification, and the yellow boxes are clutter classifications. The numbers contained in those boxes are the corresponding confidence levels. The blue boxes mark targets that the sensor failed to see entirely.

maximum of Q near H^* for Figure 5.8 as well as Figure 5.9. This means that, as long as the UAV is sufficiently close to the optimal altitude, the optimal greedy policy will always dictate that it move to that altitude. But how close "sufficiently close" is a function of γ; if γ is large enough (as in Figure 5.9), then the optimal policy will be to move to the global optimum, no matter what the aircraft's initial altitude is.

In general, one should choose high values of γ when the length of time that will be spent flying over the ROI is much greater than the time that will likely be needed to transition between the current altitude and one that is locally optimal with respect to Q. Conversely, one should choose low values of γ when flying over small fields, where the amount of time that will likely be needed to transition may be substantial compared to the length of time that will be spent flying over the field. However, further research is needed to develop a formal, mathematical framework for determining γ automatically.

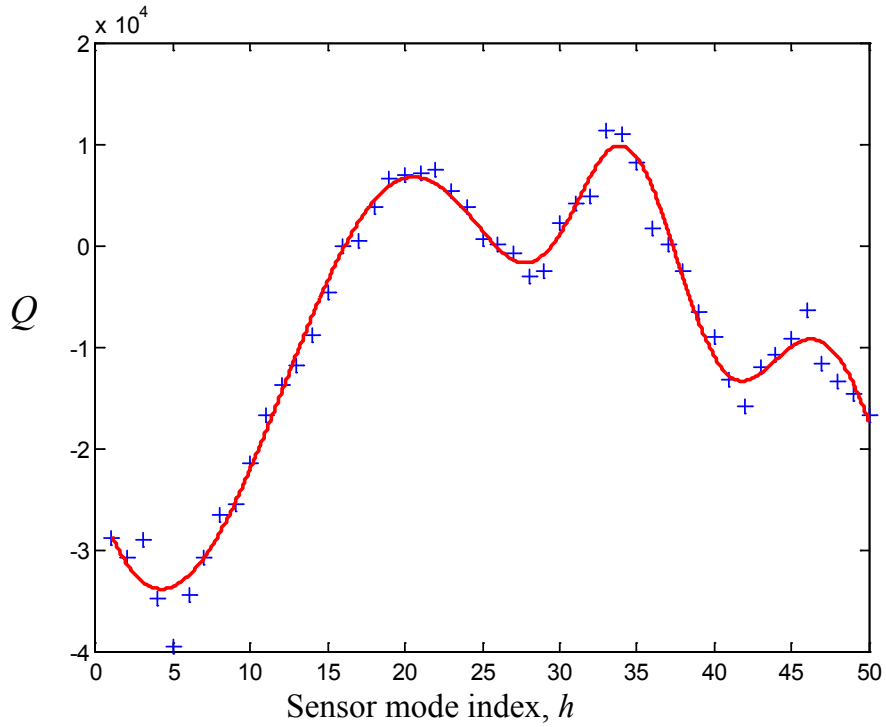


FIGURE 5.8: Q-values given by $\max_{u_k \in \mathcal{A}} Q(h_k, u_k)$, learned by UAV-IR system as a function of v_{IR}, with $\gamma = 0.9$.

Table 5.2: Performance Comparison

UAV-IR Mode	Total Bins	Mine Detections (CL)	Clutter Detections (CL)	Undetected Targets
a_8	515	13 (63%)	7 (67%)	17
a_{32}	1551	29 (68%)	23 (76%)	50
a_{34}	1610	32 (66%)	24 71%)	54

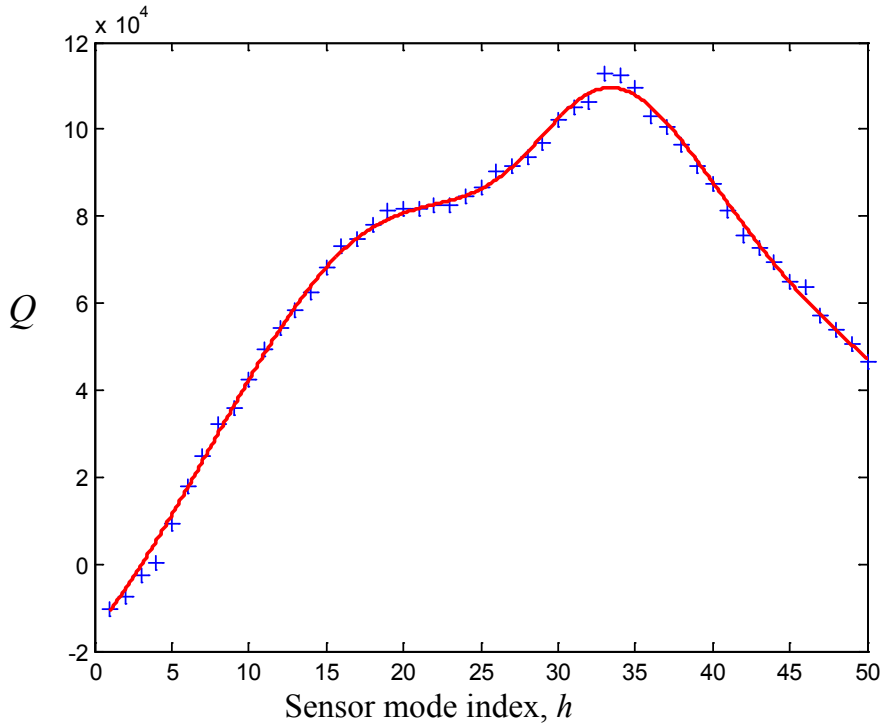


FIGURE 5.9: Q-values given by $\max_{u_k \in \mathcal{A}} Q(h_k, u_k)$, learned by UAV-IR system as a function of v_{IR}, with $\gamma = 0.99$.

6

Conclusions and Recommendations

6.1 Conclusions

This thesis develops a system whereby an aircraft can use offline, supervised learning to determine its optimal control policy as a function of state in a particular environment. It also provides a novel framework for applying Q-Learning to geometric path planning problems. The research demonstrates how a BN can be used to provide a computational model of sensor performance when its readings are influenced by large numbers of variables that cannot be fully measured or analytically understood. Additionally, it is worth noting that the BN classifier inevitably improves the value of the Q function by filtering out some of the mine-like objects in the field, and the Q-Learning algorithm provides a basis for evaluating the tradeoff between data accuracy and data collection rate. Ergo, the research demonstrates a novel method for combining different learning architectures.

The algorithm is flexible enough to lend itself to a number of different objectives and environmental expectations. If the primary purpose of the UAV mission is to find as many mines as possible, then the weight placed on correct classification can

be very high, as it was for the experiments described here. However, other goals (such as finding mine-free regions) can easily be emphasized by changing the risk parameters.

6.2 Future Research

There are a few ways to build on this research methodology. In particular, one could develop an online Q-Learning system such that the Q function would evolve with new observations using an actor-critic architecture and a greedy policy. Such a system could be initialized based on the supervised training techniques employed here and could use incremental NN training algorithms to morph the Q function. This would allow the system to train efficiently for various environments. At this point, employing an ϵ-greedy policy might be beneficial for carefully-chosen values of ϵ, on account of increased exploration of the state space. From there, possible extensions include semisupervised learning, where the ground truth is not fully known, but some subset of the observed environment is known or is assumed to be accurately determinable, potentially enabling the UAV to adjust its control policy while deployed in the field.

Appendix A

Flight Model Variables

In Chapter 3, some of the variables in Section 3.3 are described in groups for the sake of brevity. The variables are re-presented here with their individual descriptions.

h	Altitude mode
x_a	Twelve-dimensional aircraft state vector
u_a	Four-dimensional aircraft control vector
u	Velocity in the x-direction of the UAV body reference frame
v	Velocity in the y-direction of the UAV body reference frame
w	Velocity in the z-direction of the UAV body reference frame
x_r	North position in the terrestrial reference frame
y_r	East position in the terrestrial reference frame
z_r	Altitude indicator; $z_r = -h$
p	Roll rate in the UAV body reference frame
q	Pitch rate in the x-direction of the UAV body reference frame
r	Yaw rate in the x-direction of the UAV body reference frame
ϕ	Roll angle in the terrestrial reference frame

θ	Pitch angle in the x-direction in the terrestrial reference frame
ψ	Yaw angle in the x-direction in the terrestrial reference frame
X_b	Acceleration in the x-direction in the UAV body reference frame
Y_b	Acceleration in the y-direction in the UAV body reference frame
Z_b	Acceleration in the z-direction in the UAV body reference frame
L_b	Angular roll acceleration in the UAV body reference frame
M_b	Angular pitch acceleration in the x-direction of the UAV body reference frame
N_b	Angular yaw acceleration in the x-direction of the UAV body reference frame
$I_x x$	Moment of inertia about UAV body x-axis
$I_y y$	Moment of inertia about UAV body x-axis
$I_z z$	Moment of inertia about UAV body x-axis
$I_x z$	XZ product of inertia
V	Scalar aircraft speed
β	Sideslip angle
γ	Path angle; not to be confused with γ used in the Bellman equation
μ	Aircraft bank angle
δT	Throttle position
δE	Elevator angle
δA	Aileron angle
δR	Rudder angle

Appendix B

MATLAB Code

For reasons of explicitness, the program used to generate Figures 5.5, 5.8, and 5.9 is shown below.

```
clear

Gamma = 0.99; %Time discount coefficient
Alpha = 0.5;

load QMineMultiFlightResults;

%This file determines the Q function using iteration over the reward
%estimates.
CorrectReward = 10; %Specifying W_v
IncorrectPunishment = 10; %Specifying W_e

for index = 1:50
    AverageScore(index) = 0;
end

%In this section, individual flight records are checked and scored.  The
%original simulation records include a set of values rho_i for each bin
%measured, but this program is constructed so that those rho_i values can
%easily be revised from those of the original sensor simulation.
for Flight=1:10
    for ParcelNum = 1:50
        RewardRecord(ParcelNum) = 0;
```

```
        ComparisonMatrix = ComparisonSuperCell{Flight}{ParcelNum};
        for i =1:size(ComparisonMatrix,1)
            if ComparisonMatrix(i,7)==0.04 %Correct non-detection
                ComparisonMatrix(i,7)= 1;
            elseif ComparisonMatrix(i,7)== 1  %False positive
                ComparisonMatrix(i,7)= 120;
            elseif ComparisonMatrix(i,7)== 2  %False negative
                ComparisonMatrix(i,7)= 90;
            elseif ComparisonMatrix(i,7) == 3  %Correct detection
                ComparisonMatrix(i,7) = 250;
            end
            RewardRecord(ParcelNum)=RewardRecord(ParcelNum) + ...
                ((CorrectReward*(1-ComparisonMatrix(i,6))*...
                ComparisonMatrix(i,5)*ComparisonMatrix(i,7))-...
                (IncorrectPunishment*ComparisonMatrix(i,6)*...
                ComparisonMatrix(i,5)*ComparisonMatrix(i,7)));
        end
    end
    for index = 1:50
        AverageScore(index) = AverageScore(index)+RewardRecord(index)/10;
    end

end

% Initialization with reward values
for Altitude=1:50
    Q_old(Altitude) = AverageScore(Altitude);
end
for Altitude=1:50
    Q_new(Altitude) = AverageScore(Altitude);
end


% This is the value iteration process.  For sufficiently high numbers of
% iterations, a smooth-looking set of data points emerges.
for Iteration = 1:100000
    for Altitude=1:50
        if Altitude > 1 && Altitude < 50
            if Q_old(Altitude)>=Q_old(Altitude-1) && Q_old(Altitude)>=...
                    Q_old(Altitude+1)
                Q_new(Altitude) = AverageScore(Altitude) + ...
                    Gamma*Q_old(Altitude);
            elseif Q_old(Altitude-1)>=Q_old(Altitude) && ...
                    Q_old(Altitude-1)>=Q_old(Altitude+1)
                Q_new(Altitude) = AverageScore(Altitude) + ...
                    Gamma*Q_old(Altitude-1);
            else
                Q_new(Altitude) = AverageScore(Altitude) + ...
                    Gamma*Q_old(Altitude+1);
            end
        elseif Altitude == 1
```

```
            if Q_old(Altitude)>=Q_old(Altitude+1)
                Q_new(Altitude) = AverageScore(Altitude) + ...
                    Gamma*Q_old(Altitude);
            else
                Q_new(Altitude) = AverageScore(Altitude) + ...
                    Gamma*Q_old(Altitude+1);
            end
        else
            if Q_old(Altitude)>=Q_old(Altitude-1)
                Q_new(Altitude) = AverageScore(Altitude) + ...
                    Gamma*Q_old(Altitude);
            else
                Q_new(Altitude) = AverageScore(Altitude) + ...
                    Gamma*Q_old(Altitude-1);
            end
        end
    end
    for Altitude = 1:50
        Q_old(Altitude) = (1-Alpha)*Q_old(Altitude) + ...
            Alpha*Q_new(Altitude);
    end
end

% Given the Q-values of the 50 discrete altitude modes, it remains only to
% interpolate between them using a neural network.

% M-file to train and simulate a single-input, two-layer,
% feed-forward backpropagation neural network.
X = 1 : 1 : 50;
for i=1:50
    Y(i) = Q_new(i);
end

pr = [-16000 4000];
m1 = 10; m2 = 1;
%Initialize 2-layer feed-forward network:
% net    = newff (X,Y,5);
net = newff(X, Y, [m1 m2], {'logsig','purelin'}, 'trainbr');
net    = init (net); %Default Nguyen-Widrow initialization
%Training:
net.trainParam.goal = 10;
net.trainParam.max_fail=200;
net.trainParam.mu_inc=1.1;
net.trainParam.epochs = 300;
net = train(net, X, Y);
%Simulation:
X_sim = 1 : 0.001 : 50;
Y_nn = sim (net, X_sim);
figure
plot(X, Y, '+'); hold on
plot(X_sim, Y_nn, 'r-'); hold off
```

```
set(findobj(gca,'Type','line','Color','red'),'LineWidth',2);
```

Bibliography

Baumgartner, K. (2005), "Bayesian Network Modeling of Offender Bahavior for Criminal Profiling," Master's thesis, Duke University.

Bellman, R. (1957), "A Markovian Decision Process," *Indiana Univ. Math. J.*, 6, 679–684.

Bertsekas, D. P. (1995), *Dynamic Programming and Optimal Control, Vols. I and II*, Athena Scientific, Belmont, MA.

Bertsekas, D. P. and Tsitsiklis, J. N. (2002), *Introduction to Probability*, Athena Scientific, Belmont, MA.

Cai, C. and Ferrari, S. (2008), "A Q-Learning Approach to Developing an Automated Neural Computer Player for the Board Game of CLUE®," in *International Joint Conference on Neural Networks*, pp. 2347–2353, Hong Kong.

Cai, C. and Ferrari, S. (2009), "Information-Driven Sensor Path Planning by Approximate Cell Decomposition," *IEEE Transactions on Systems, Man, and Cybernetics - Part B*, 39.

Cooper, G. and Herkovits, E. (1992), "A Bayesian Method for the Induction of Probabilistic Networks from Data," *Machine Learning*, 9, 309–347.

Cybenko, G. (1989), "Approximation by Superpositions of a Sigmoidal Function," *Mathematics of Control, Signals, and Systems*, 2, 303–314.

Dam, R. V. (2003), "Soil Effects on Thermal Signatures of Buried Nonmetallic Landmines," *Detection and Remediation Technologies for Mines and Minelike Targets VIII, Proc. of the SPIE*, 5089, 1210–1218.

Demuth, H., Beale, M., and Hagan, M. (2010), *Neural Network Toolbox 6 User's Guide*, The MathWorks, Inc.

Explosive, Ordnance, Disposal, (EOD), and Technicians (2006), *ORDATA Online*, Avaliable: http://maic.jmu.edu/ordata/mission.asp.

Ferrari, S. (2000), "Tutorials in Robotics and Intelligent Systems, Part V: Introduction to MATLAB Neural Network Toolbox," .

Ferrari, S. and Cai, C. (2009), "Information-Driven Search Strategies in the Board Game of CLUE®," *IEEE Transactions on Systems, Man, and Cybernetics - Part B*, 39.

Ferrari, S. and Stengel, R. (2002), "Classical/Neural Synthesis of Nonlinear Control Systems," *Journal of Guidance, Control, and Dynamics*, 25, 442–448.

Ferrari, S. and Stengel, R. (2004), "Model-based Adaptive Critic Designs," in *Learning and Approximate Dynamic Programming*, eds. J. Si, A. Barto, and W. Powell, John Wiley and Sons.

Ferrari, S. and Vaghi, A. (2006), "Demining Sensor Modeling and Feature-level Fusion by Bayesian Networks," *IEEE Sensors*, 6, 471–483.

Heckerman, D. (1995), "A Bayesian approach to learning causal networks," in *In Uncertainty in AI: Proceedings of the Eleventh Conference*, pp. 285–295.

Jensen, F. (2001), *Bayesian Networks and Decision Graphs*, Springer-Verlag.

Jensensius, M. (2005), "Cnstrained Learning in Neural Network Systems," Master's thesis, Duke University.

MacDonald, J. (2003), *Alternatives for Landmine Detection*, Rand Publications.

MacKay, D. (1992a), "Bayesian Interpolation," *Neural Computation*, 4, 415–447.

MacKay, D. (1992b), "A Practical Bayesian Framework for Backpropagation Networks," *Neural Computation*, 4, 448–472.

Murphy, K. (2007), *Bayes Net Toolbox for Matlab*, Available: http://code.google.com/p/bnt/.

Nelson, R. C. (1998), *Flight Stability and Automatic Control, Second Edition*, WCB/McGraw-Hill.

Russell, S. and Norvig, P. (2003), *Artificial Intelligence: A Modern Approach*, Prentice Hall, NJ.

Scruggs, J. (2010), "Linear Systems Notes," CE 285, Duke University.

Si, J., Barto, A., and Powell, W. (1992), "Learning and Approximate Dynamic Programming," *Proceedings of the IEEE*, 80, 1384–1399.

Si, J., Barto, A. G., Powell, W. B., and II, D. W. (eds.) (2004), *Handbook of Learning and Approximate Dynamic Programming*, IEEE Press, Piscataway, NJ.

Stengel, R. F. (1986), *Optimal Control and Estimation*, Dover Publications, Inc.

Stengel, R. F. (2004), *Flight Dynamics*, Princeton University Press, Princeton, NJ.

Watkins, C. (1989), "Learning from Delayed Rewards," Ph.D. thesis, King's College.

Werbos, P. (2004), "ADP: Goals, Opportunities, and Principles," in *Learning and Approximate Dynamic Programming*, eds. J. Si, A. Barto, and W. Powell, John Wiley and Sons.

White, D. and Jordan, M. (1992), "Optimal Control: A Foundation for Intelligent Control," in *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches*, eds. D. White and D. Sofge, Van Nostrand Reinhold.