UNLV | UNIVERSITY LIBRARIES

UNLV Theses, Dissertations, Professional Papers, and Capstones

5-1-2014

Implementing Influence Diagram Concepts to Optimize Border Patrol Operations using Unmanned Aerial Vehicles

Ashish Karki
University of Nevada, Las Vegas

Follow this and additional works at: https://digitalscholarship.unlv.edu/thesesdissertations

 Part of the Computer Sciences Commons

Repository Citation

Karki, Ashish, "Implementing Influence Diagram Concepts to Optimize Border Patrol Operations using Unmanned Aerial Vehicles" (2014). *UNLV Theses, Dissertations, Professional Papers, and Capstones*. 2103.
http://dx.doi.org/10.34917/5836122

This Thesis is protected by copyright and/or related rights. It has been brought to you by Digital Scholarship@UNLV with permission from the rights-holder(s). You are free to use this Thesis in any way that is permitted by the copyright and related rights legislation that applies to your use. For other uses you need to obtain permission from the rights-holder(s) directly, unless additional rights are indicated by a Creative Commons license in the record and/or on the work itself.

This Thesis has been accepted for inclusion in UNLV Theses, Dissertations, Professional Papers, and Capstones by an authorized administrator of Digital Scholarship@UNLV. For more information, please contact digitalscholarship@unlv.edu.

IMPLEMENTING INFLUENCE DIAGRAM CONCEPTS TO OPTIMIZE BORDER PATROL OPERATIONS USING UNMANNED AERIAL VEHICLES

by

Ashish Karki

Bachelor's Degree in Computer Engineering
Pulchowk Campus, Institute of Engineering
Tribhuvan University, Nepal
2010

A thesis submitted in partial fulfillment of
the requirements for the

Master of Science in Computer Science

Department of Computer Science
Howard R. Hughes College of Engineering
The Graduate College

University of Nevada, Las Vegas
May 2014

© Ashish Karki, 2014
All Rights Reserved



THE GRADUATE COLLEGE

We recommend the thesis prepared under our supervision by

Ashish Karki

entitled

Implementing Influence Diagram Concepts to Optimize Border Patrol Operations using Unmanned Aerial Vehicles

is approved in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science

Department of Computer Science

Wolfgang Bein, Ph.D., Committee Chair

Kazem Taghva, Ph.D., Committee Member

Laxmi P. Gewali, Ph.D., Committee Member

Emma Regentova, Ph.D., Graduate College Representative

Kathryn Hausbeck Korgan, Ph.D., Interim Dean of the Graduate College

May 2014

ABSTRACT

Implementing Influence Diagram Concepts to Optimize Border Patrol Operations using Unmanned Aerial Vehicles

by

Ashish Karki

Dr. Wolfgang Bein, Examination Committee Chair

Professor, Department of Computer Science

University of Nevada, Las Vegas

The most common approach for border patrol operations is the use of human personnel and manned ground vehicles, which is expensive, at times inefficient and sometimes even hazardous to people involved. The length of the US border, mostly covering unpopulated areas, with harsh atmospheric conditions makes it more susceptible to illegal human activities. Automated border surveillance by unattended, fixed, ground sensors forming an electronic fence has proven expensive, inefficient and was prone to unacceptable rate of false alarms.

A better approach would be using Unmanned Aerial Vehicles (UAVs) in combination with such ground sensors. This would help improve the overall effectiveness of the surveillance system as a UAV could first scan the alert area before sending in personnel and vehicles, if deemed necessary.

In this thesis, we are proposing border surveillance using multiple Unmanned Aerial Vehicles (UAVs) in combination with alert stations consisting of Unattended Ground Sensors (UGSs) along the border line/fence. Upon detecting an event, an alert would be triggered by any UGS. We simulate this process by reading probability data for different timestamps from a text file. And, based on utility values of each stations, two UAVs decide on which alert stations to service.

ACKNOWLEDGEMENTS

"First and foremost, my humble gratitude to my committee chair Dr. Wolfgang Bein for accepting me to work under his guidance. His insights and comments have helped me immesely during this time. It has been an honor to have Dr. Laxmi P. Gewali, Dr. Kazem Taghva and Dr. Emma Regentova in this thesis committee and my sincerest thanks to all of them. I would also like to extend my thankfulness to Dr. Doina Bein for her guidance and time to bring the thesis to its current form.

I am deeply indebted to Dr. Ajoy K. Datta for his help and invaluable guidance during my masters study at UNLV. I would also like to express my appreciation to the Department of Computer Science, its faculty members and staff for their support as well.

Thanks is also due to all my friends and colleagues who have been a very important part of my personal and work life. Last but not the least, this thesis is dedicated to my parents, Rudra and Anjana Karki, for being the most loving and encouraging of all at every stage of my life. "

Ashish Karki

University of Nevada, Las Vegas
May 2014

TABLE OF CONTENTS

ABSTRACT iii

ACKNOWLEDGEMENTS iv

TABLE OF CONTENTS v

LIST OF FIGURES viii

CHAPTER 1 INTRODUCTION 1

1.1 Motivation . . . 2

1.2 Related Work . . . 2

1.3 Thesis Summary . . . 3

CHAPTER 2 LITERATURE REVIEW 5

2.1 Decision Making . . . 5

2.2 Decision Support Tools . . . 5

2.2.1 Decision Trees . . . 6

2.2.2 Influence Diagrams . . . 8

2.3 Important Concepts and Notations . . . 12

2.3.1 Probability Data . . . 12

2.3.2 Intelligence Data . . . 12

2.3.3 Probabilistic Decision Making with Intelligence Data . . . 12

2.3.4 Maximizing Expected Utility (MEU) Principle . . . 13

CHAPTER 3 MATHEMATICAL MODELING 14

3.1 Types of Events . . . 14

3.2 Types of Site Data . . . 14

3.2.1 Probability Data . . . 15

3.2.2 Intelligence Data . . . 16

3.2.3 Ground Truth . . . 18

3.3 UGS States . . . 19
3.4 UAV States . . . 20
3.5 UAV Direction (from graphical point of view) . . . 21
3.6 Value Functions . . . 21
3.6.1 Gain Function . . . 22
3.6.2 Cost Function . . . 22
3.6.3 Utility Function . . . 23
3.7 Influence Diagram Concepts in Decision Making . . . 23

CHAPTER 4 SOFTWARE IMPLEMENTATION 25

4.1 Programming Environment and Setup . . . 25
4.2 Code Structure . . . 25
4.3 Class Diagrams . . . 26
4.3.1 Class Diagram for Drawing Package . . . 27
4.3.2 Class Diagram for Main Package . . . 28
4.3.3 Class Diagram for Whole Project . . . 29
4.4 Implementation Phases . . . 30
4.4.1 Single UAV - Multiple Alert Stations . . . 30
4.4.2 Two UAVs - Multiple Alert Stations with Fixed Area of Surveillance . . . 30
4.4.3 Two UAVs - Multiple Alert Stations with Changing Area of Surveillance . . . 30
4.5 Implementation of Decision Making with Pseudo-codes . . . 31
4.5.1 Updating Probabilistic Distribution with Intelligence Data . . . 31
4.5.2 Calculating Site Gain Values . . . 32
4.5.3 Calculating Site Cost Values . . . 34
4.5.4 Calculating Site Utility Values . . . 35
4.5.5 Applying MEU Principle . . . 35
4.5.6 UAV locations per Timestamp . . . 36
4.5.7 Calculating Actual Gain and Cost . . . 36

CHAPTER 5 RESULTS AND USER DOCUMENTATION 38

5.1 Console Output . . . 38
5.1.1 Processing of Valid and Invalid Intelligence Data . . . 38
5.1.2 Processing of Invalid Probabilistic Distribution Data . . . 39
5.1.3 Processing of Valid Probabilistic Distribution Data . . . 39
5.2 Graphical Display . . . 41

5.3 User Documentation 43
5.3.1 Configuring and Running the Project 43
5.3.2 Input and Output files 44
5.3.3 Code Structure and Flow 44

CHAPTER 6 CONCLUSION AND FUTURE WORK **45**

APPENDIX A CODE SNIPPETS INVOLVING DECISION MAKING **46**
A.1 Calculating Estimated Site Gain 46
A.2 Calculating Site Utility and Applying MEU Principle 49
A.3 Calculating Next Location for UAVs 51

APPENDIX B CODE SNIPPET INVOLVING GRAPHICS DISPLAY **55**

BIBLIOGRAPHY **62**

VITA **64**

LIST OF FIGURES

2.1 Steps of decision making 6
2.2 A decision tree analysis to decide whether to buy a new car 7
2.3 The value of variable X is known when decision Y is made 10
2.4 The decision X is known when decision Y is made 10
2.5 The probabilities related with chance variable Y is dependent on the outcome of variable X 10
2.6 The probabilities related with chance variable Y is dependent on the decision X . . . 10
2.7 An influence diagram to decide which car to buy 11
2.8 An influence diagram to depict expected utility 13

3.1 A block of probability values for timestamp = 70 15
3.2 Two blocks of probability values to show invalid data 16
3.3 A block of intelligence data values for timestamp = 4 17
3.4 A block of intelligence values to show invalid data 17
3.5 Two blocks of ground truth values 18
3.6 Diagram depicting UGS states 19
3.7 Diagram depicting UAV states 20
3.8 Influence Diagram of our implementation 24

4.1 Class Diagram for Drawing Package 27
4.2 Class Diagram for Main Package 28
4.3 Class Diagram for Whole Project 29
4.4 Gain Matrix used in Decision Making 33

5.1 Graphical Display at timestamp = 0 41
5.2 Graphical Display at timestamp = 70 42
5.3 Graphical Display at timestamp = 85 42
5.4 Graphical Display at timestamp = 90 42

5.5 New Project Diaglog box in Eclipse 43

CHAPTER 1

INTRODUCTION

The border of a country defines its political boundaries and areas of its jurisdictions. A border region carries immense importance for a country in terms its national security. But, proper controlling of such regions is a major challenge for most of them. As mentioned before, these boundaries not only contain land areas but also water bodies and rough territories. Also, the length of the boundary line may sometimes be long and patrolling it puts a strain on the resources that any nation can afford. As an example the total length of border between the U.S. and Mexico is approximately 2000 miles and that between the U.S. and Canada is approximately 5500 miles [1].

Most countries opt for patrolling their borders using guards stationed at posts. These personnel are often times backed up by ground vehicles and aerial support like helicopters [2]. This approach is not without drawbacks:

1. Equipment like vehicles and helicopters are expensive to build and maintain.
2. It could be inefficient as unnecessary resources could be spent on false alerts or insignificant events like the intrusion of a small animal.
3. As mentioned before, the boundaries sometimes contain rough terrain which can be difficult to secure, besides being life-threatening to the security personnel involved.

Sometimes security may be improved by fencing a part of the border line. But, fencing alone is not enough. These fences need to be patrolled. Ground sensors can sometimes be a part of such fencing or could be used separately. Such sensors trigger alarms at possible intrusion. False alarms require unnecessary human intervention which in turn is expensive and even dangerous.

1.1 Motivation

To compliment the current measures used in border patrol operations, a need for aerial surveillance using unmanned vehicles has emerged. Countries like the United States are actively testing and deploying Unmanned Aerial Vehicles (UAVs) [1] [3]. A UAV can significantly extend the range of security missions. UAVs are equipped with various sensors that can detect potential violators inside thick woods and even in rugged areas or mountain regions. A remote controlled UAV would be safer than a manually operated vehicle or aircraft in certain conditions.

For our purpose, we assume the border to be a terrestrial area which is more or less like a line. In other words, it is considered long and thin (relative to actual border size). There are n number of alert stations consisting of Unattended Ground Sensors (UGSs). Any UGS upon detecting an intrusion of any kind, sets off an alert. A UAV is then sent to this particular site or station to further investigate the alert. In subsequent sections, we describe the use of multiple UAVs to investigate alerts based on priority calculated using probabilistic data.

It is worth mentioning that the idea of using UGSs and UAVs for border patrolling could be extended to Perimeter Patrol Operations as well. The application of such operations could be securing a nuclear facility or a military installation and monitoring a wildlife reserve or an oil field [2].

1.2 Related Work

A perimeter patrol problem consisting of multiple UAVs equipped with cameras and controlled remotely by an operator is addressed in [4]. After the trigger of an alert, a UAV flies to the site to investigate the alert, that is to check if the alert is a true and if yes, what kind of intrusion is underway. The decision problem solved is the determination of optimal loitering or dwelling time of a UAV at an alert site. That is the optimal amount of time spent at a site so that maximum amount of information can be gathered but at the same time the servicing delay to other sites is kept as minimal as possible. The problem is formulated as a Markov decision process and a solution is obtained using dynamic programming. Simulations are run for cases like using one or two UAVs and UAVs with or without turnaround capabilities. The quality of service is improved when using two UAVs with turnaround capabilities. Similar discussion has been done in [5] [6].

A system to coordinate the working of multiple UAVs to be utilized in border and perimeter patrolling jobs is presented in [7]. The motion control and navigation among the UAVs is divided into

a hierarchical architecture. The lower hierarchy consists of controllers that perform positioning and tracking jobs. The higher hierarchy systems are used for maneuvering operations. A team of UAVs is assigned to patrol a particular region of a border. A border region is defined to be a geographical region which is, in general, like a line: thin and long. A particular border region is split into sectors which are roughly the same in shape and size. Each aircraft is assigned a sector. If any UAV detects an event or target, it alerts its operator. If the operator deems the target is important, the UAV is commanded to follow it. The sectors are then dynamically reassigned among remaining UAVs.

A variation of the UAV patrol problem would be the Dynamic Traveling Repairman Problem (DTRP) [8]. In the latter case, the paths traveled may not be in a closed path or a line. The objective in this case is to minimize the service delay time rather than the traveling time between various points or sites. An example requiring the minimization of wait time is ambulance services. Few properties associated with DTRP are: 1) objective is to minimize waiting time rather than travel time; 2) the arrival of service demands is random; 3) and, the demands are dynamic with respect to time.

1.3 Thesis Summary

These days a lot of focus is being given to the use of UAVs for border patrolling. UAVs also called drones or remotely piloted vehicles (RPVs) [9] are used in co-ordination with sensors, fences and video cameras. More particularly, we are concerned with sensors contained in posts or alert stations called UGS.

We have implemented an algorithm by which two UAVs monitor a set of alert sites. For our simulation, values expressed in terms of percentage for different types of events (detected by a UGS) are extracted from a text file. The percentages represent the probability of occurrence of each type of event for a particular site. In addition, there is an intelligence data file containing coefficient values for the same types of events. The intelligence data indicates the general pattern of events detected by a station over a longer period of time. In brief, the following steps are then performed:

1. The probability data is updated using the intelligence data.
2. For each site (for a particular timestamp), two values are computed: gain and cost.
3. For each site(for a particular timestamp), a utility value is computed where utility = gain - cost.
4. We apply the principle of Maximizing Expected Utility (MEU) to choose two sites, marked as active, which would be serviced by the two UAVs.

The working of UAVs is programmed such that they "greedily" choose an active alert station closest to their starting location or alert site. Additionally, to display the motion of UAVs and flagging or setting off of alerts, a graphical animation panel has been implemented.

CHAPTER 2

LITERATURE REVIEW

This chapter contains the concepts and models used in our thesis. Because we primarily deal with a decision problem, an analysis of relevant decision making tools is done in this chapter. First we define and present the steps of decision making. Next, we discuss about two decision tools namely decision trees and influence diagrams. We examine their semantics and simple use cases.

2.1 Decision Making

Decision making can be defined as a process in which a selection is made from a set of alternatives or options [10]. The selection is made in such a way so as to minimize or maximize an objective.

Decision Making forms an important part of daily activity for people like administrators or scientists. An example of decision making could be that of a school administrator deciding whether to hire a new teaching staff or increase the load of the current staff.

As a part of decision making, it is required to first identify all possible and feasible choices. Next, these choices need to be analyzed or evaluated based on some criteria. And, finally a resulting choice or selection is made that satisfies certain needs [11].

The Rational model of decision making is one in which all aspects of the process such as the set of choices, results and decision criteria are known beforehand [12]. As per this model, decision making can be listed in six steps shown in Fig. 2.1:

2.2 Decision Support Tools

In the current section, a discussion about tools or theories that aid in the process of decision making is presented. Such tools help in properly defining a problem, analyzing it and reaching a conclusion. These tools, especially diagrammatic ones, can help convey information to relevant people.

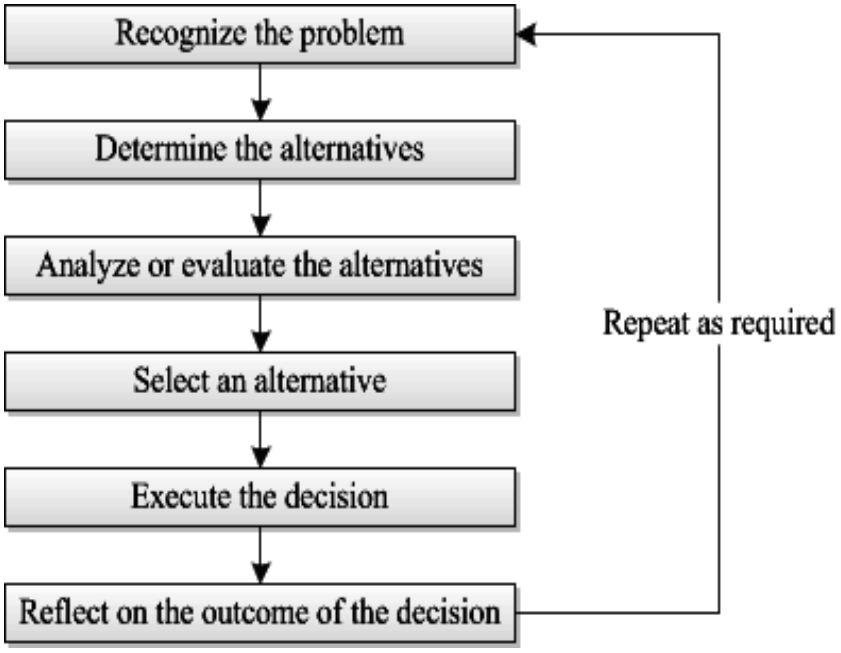


Figure 2.1: Steps of decision making

2.2.1 Decision Trees

A decision tree is a decision analysis tool in which a problem is represented as a directed acyclic graph with nodes and arcs [13]. A node is an element in which either a decision is evaluated or an uncertainty is calculated. The alternative or branch which gives the best overall value is the best path within the tree. A decision tree, if arranged from left to right, means that the events on the left have occurred earlier than the one on its right and so on.

Decision Tree Structure

Within a decision tree, a node and all its descendants is called a *branch* of that node. The bottom-most nodes are called *leaves* [14].

A decision tree has mainly two types of nodes:

1. Decision node: it is drawn as a square and represents the alternatives or choices a decision maker has in his control.

2. Chance node: it is drawn as a circle and represents uncertain factors during decision making.

Both decision and chance nodes will have arcs leading away from them to depict the possible outcome of such nodes.

A simple example of utilizing decision trees for decision making is shown in Fig. 2.2.

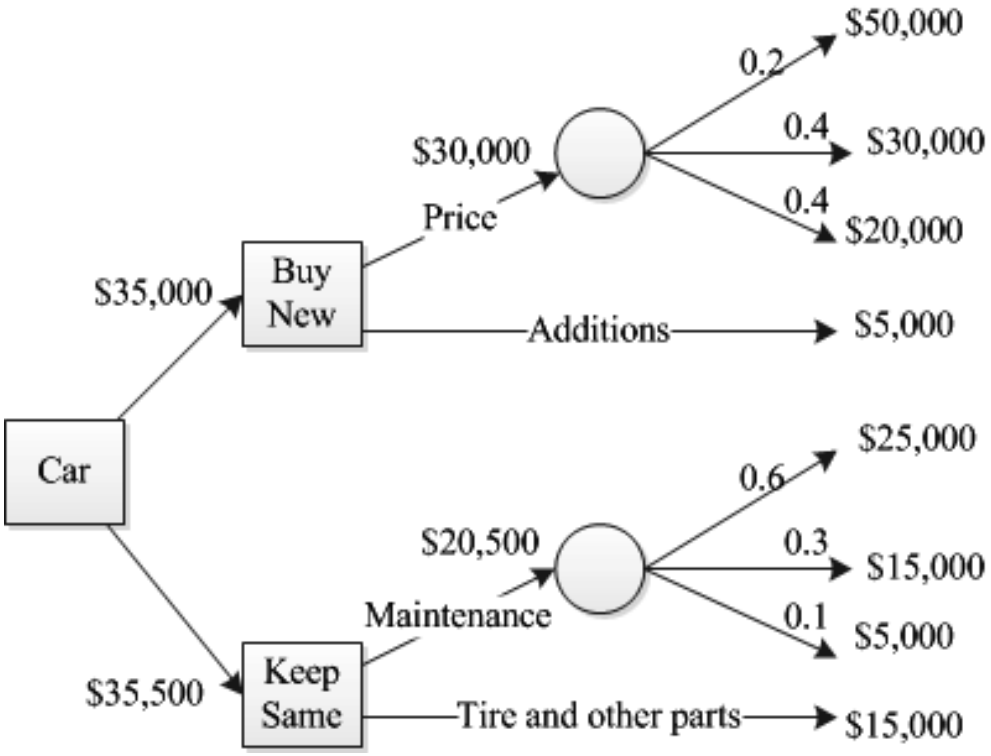


Figure 2.2: A decision tree analysis to decide whether to buy a new car

Evaluating a Decision Tree

The evaluation begins at the far right end of a given tree. For every chance node, the expected outcome is calculated by summing the product of measured value of outcome times its probability for each arc leading away from that node. These expected outcomes become input for other chance and decision nodes. At a decision node, the path that has the maximum expected outcome value is selected from a set of alternatives.

Keeping these concepts in mind, from Fig. 2.2 we have following inferences:

- The values printed over the arcs or leading arrows from a chance node are the probabilities of occurrence of an outcome. For example, on the Buy New car branch the probability that we would buy a car of priced at $50,000 is 0.2.
- Values at chance nodes are aggregated and printed over the arcs leading into the chance nodes.
- Values at decision nodes is a summation of dollar amount in the arcs leading away from respective nodes.
- In this example, it turns out that the total cost of using the same car is more than that of buying a new car. Hence, the best financial choice is to buy a new car.

Relationship with Influence Diagrams

A decision tree is very useful in graphically displaying the decision alternatives of a problem. But in cases where there are too many variables: decision alternatives and uncertainties, the graph can quickly become large and complex. In such cases, we can represent the same problem with influence diagrams because they focus on relationship among various elements and less on statistical values. Influence diagrams are discussed in Section 2.2.2.

2.2.2 Influence Diagrams

An influence diagram (ID) is a graphical representation of a decision situation. The graph consists of nodes representing various values. It also contains arcs that depict relationships between the nodes [15]. IDs were described by [16] as modeling tools to visualize relationships among random variables and resulting decisions.

Influence Diagram Structure

An ID is a directed acyclic graph [17]. It describes a decision problem in three levels: relational, functional and numerical [18]. A brief discussion of these levels will be presented in items below. This discussion follows from the ones presented in [19] [17] [20].

1. **Level of relation**: This level is the actual graphical representation of a problem using nodes and arcs that connect one node to another. The arcs define relationships or dependencies among different nodes. There are three type of nodes (and a sub-node type) and three types of arcs used in this level.

   (a) **Types of Nodes**

      i. Decision Node: it represents a decision to be made and is drawn as a rectangle or square. The variables that represent decision nodes are under the control of a decision maker.

      ii. Chance or Uncertainty Node: it represents uncertainty in a problem and is drawn as an oval or a circle. Such nodes represent random variables.

         - Deterministic Node: it is a special case in which the outcome is deterministic whenever the outcomes of other uncertainties are known. It is drawn as a double oval or a circle.

      iii. Value or Utility Node: it represents the final choice or expected reward of a decision and is drawn as a diamond or an octagon. Such nodes are represented by a utility function.

   (a) **Types of Arcs**

      i. Conditional Arc: it is directed towards a chance node and indicates that the probability of this node is conditioned on the value of the input node.

      ii. Functional Arc: it is directed towards a value node and indicates that the utility function of this node is itself a function of the input node.

      iii. Informational Arc: it is directed towards a decision node and indicates that the choice(s) made in this node are conditional upon an input node. This node can be an uncertainty or another decision.

2. Level of function: This level specifies the actual function that describes the dependencies indicated by the graphical structure at the relational level.

3. Level of number: This level specifies the numerical values related to probability and utility functions.

We outline the dependencies between different types of arcs and nodes in Fig. 2.3, Fig. 2.4, Fig. 2.5 and Fig. 2.6.

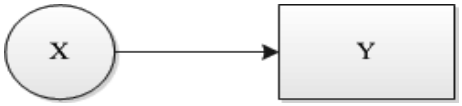


Figure 2.3: The value of variable X is known when decision Y is made

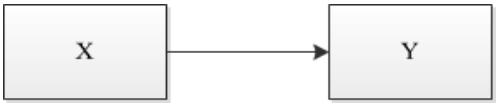


Figure 2.4: The decision X is known when decision Y is made

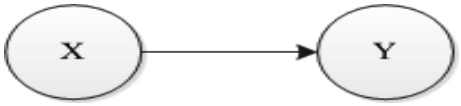


Figure 2.5: The probabilities related with chance variable Y is dependent on the outcome of variable X



Figure 2.6: The probabilities related with chance variable Y is dependent on the decision X

Next, we depict a simple example of using influence diagrams in Fig. 2.7.

In Fig. 2.7, note the following:

- The car design and features variable is deterministic because we are assuming that the features are fixed for a given price.

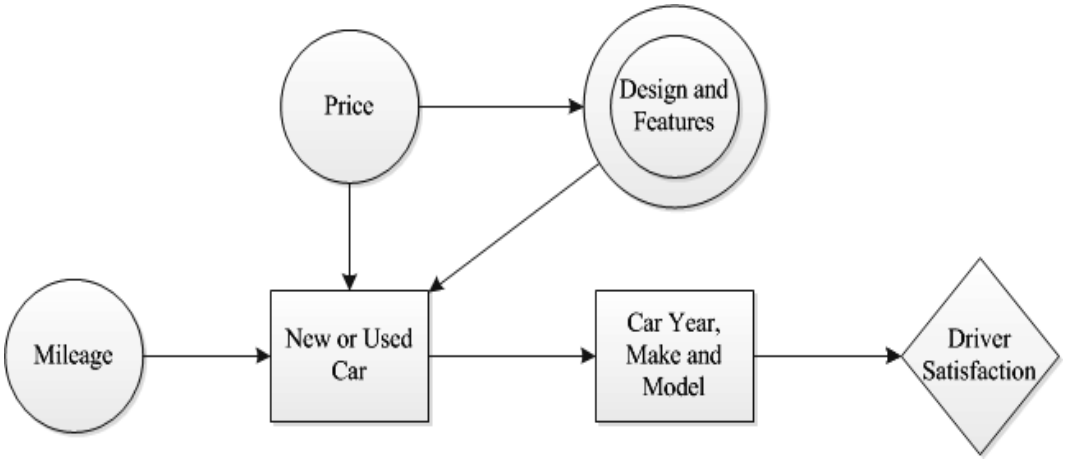


Figure 2.7: An influence diagram to decide which car to buy

- The decision of getting a new car or a used car depends on the values of price, design and features (both interior and exterior) and car mileage (miles per gallon). Consider a scenario in which a buyer gets a used car for a decent price with good mileage and features, he might buy one. In another scenario, a buyer may purchase a new car if his budget allows for it and the features are great (mileage may not be very important here).
- The car make and model as well as the year it was manufactured (i.e. how old or new the car is) determine the driver satisfaction value.
- There is no arc between the price node and the mileage node of the car which means that a more expensive car may not necessarily get a better mileage and vice-versa.
- The arcs ending in the Design and Features node are conditional arcs.
- The informational arc ending in the New or Used Car node indicates that the decision is made depending upon the price, features designs and fuel economy of a car.
- The informational arc ending in Car Year, Make and Model node indicates that the buyer already knows if he is buying a new or a used car before making this decision.
- Finally, the functional arc ending in Driver Satisfaction node indicates that the satisfaction depend upon the value of a car's manufacture year and its brand and model.

2.3 Important Concepts and Notations

2.3.1 Probability Data

An Unattended Ground Sensor (UGS) is a group of sensors that contains, but is not limited to, thermal and seismic sensors. Each UGS detects events which are an aggregation from all its sensors. The events that are detected are classified into an event space (as discussed in Section 3.1) based on a *perception* of threat posed by each detection. As an example for a border patrolling operation, we can say that events categorized as being human are less potent as threats than those categorized as being vehicles. The final result of classification is a probabilistic distribution, expressed in percentage, over the event space. An example of probability distribution for an alert site, where N = no event, S = small animal, L = large animal, H = human and V = vehicle from the event space, can be: (5% N, 10% S, 20% L, 60% H, 5% V) . For each alert station, one such distribution is generated for a timestamp. Therefore, there would be more than one block of data (representing all alert stations) for each timestamp value.

2.3.2 Intelligence Data

Intelligence Data is a set of coefficients over the same event space that compounds the probability distribution to either strengthen or weaken some values. In general, our assumption is that the intelligence data changes very slowly compared to its probabilistic counterpart. Hence, intelligence is what is known to be a pattern over a longer period of time. As an analogy, we can compare probability values to weather and intelligence values to climate of any geographical region.

2.3.3 Probabilistic Decision Making with Intelligence Data

In our decision making process, we update the probabilistic distribution values with intelligence coefficients, if they are available. Then, decision making is performed on the updated probability values. If no intelligence is known, the probabilistic distribution remains unchanged. Otherwise, intelligence changes the probability. In other words, if something is more probable based on past experience, its probabilistic distribution is increased. For example: if an alert site is known for wild animal crossings, then the values for small and large animals are increased.

Basic Mathematical Example

The process of updating is performed by multiplying the values (of the event space) obtained from the probability distribution with the corresponding values in the intelligence coefficients. The multiplied values are normalized to obtain final values. Consider the aforementioned probability distribution

for a site given by (5, 10, 20, 60, 5). If our intelligence coefficients are $(\frac{1}{2}, 2, 2, 1, \frac{1}{2})$, the probability of N and V is halved, that of S and L are doubled and that of H remains the same. The calculations are performed as shown:

$(5, 10, 20, 60, 5) \times (\frac{1}{2}, 2, 2, 1, \frac{1}{2}) = (\frac{5}{2}, 20, 40, 60, \frac{5}{2})$

These values are normalized as:

$(\frac{5}{2} \times \frac{100}{125}, 40 \times \frac{100}{125}, 60 \times \frac{100}{125}, \frac{5}{2} \times \frac{100}{125})$ where $125 = \frac{5}{2} + 20 + 40 + 60 + \frac{5}{2}$

$= (2, 16, 32, 48, 2)$ are the final updated distribution values for a site

2.3.4 Maximizing Expected Utility (MEU) Principle

A basic influence diagram for decision making can be drawn as shown in Fig. 2.8 [19].

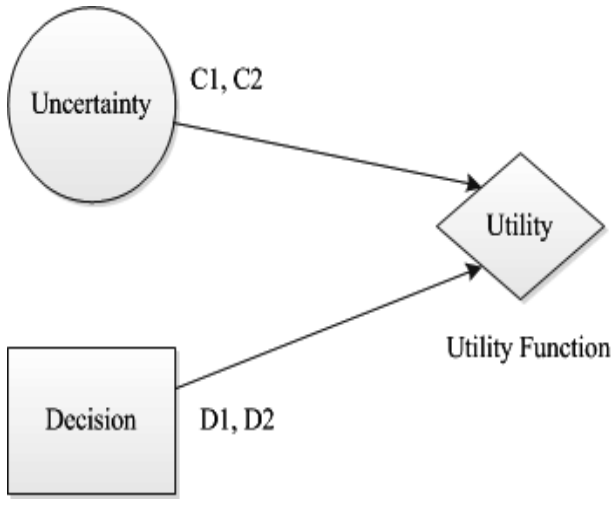


Figure 2.8: An influence diagram to depict expected utility

Let us assume that the outcomes of Uncertainty are C1 and C2 and the probabilities associated with them are P(C1) and P(C2) respectively. Let the alternatives for Decision be represented as D1 and D2. As per the MEU principle, we make our decision in order to maximize the value of Utility [21] [22]. The utility can be represented by a function shown in 2.1.

$$U(D_{\mathrm{i}}, C_{\mathrm{j}}) : P \rightarrow \mathbb{R} \tag{2.1}$$

CHAPTER 3

MATHEMATICAL MODELING

In this chapter we present topics related to real-life objects and actions such as the types of events sensed by a UGS, the states of a UAV and a UGS and the way in which influence diagram semantics is used in our decision making.

3.1 Types of Events

We have defined five types of events that can be sensed by a UGS. These event types also referred to as *event space* are No Event (N), Small Animal (S), Large Animal (L), Human (H) and Vehicle (V). To each of these events we have assigned an importance value or weight which are totally ordered:

$$\mathrm{N} < \mathrm{S} < \mathrm{L} < \mathrm{H} < \mathrm{V} \tag{3.1}$$

So, if our event space (E) is the set {N, S, L, H, V}, then we can define the weight space (W) as follows:

$$W : E \rightarrow \mathbb{R}^{+\,|E|} \tag{3.2}$$

Based on (3.2), we have:

$$W(N) = 1, W(S) = 2, W(L) = 3, W(H) = 4, W(V) = 5 \tag{3.3}$$

3.2 Types of Site Data

The input to the decision making algorithm implemented in our thesis comes from two text files. One of these files contains probability distribution over the event space described in section 3.1 and the other is a set of coefficients. The algorithm outputs a file that contains the ground truth which is the actual event occurring in a given alert station for a specific timestamp. We are concerned with three data files whose format and contents are described in the subsections that follow.

3.2.1 Probability Data

This data is contained in a text file named *SiteProbabilityDistribution*. The file contains blocks of data for separate, increasing timestamps. An example of a block of data is as shown in Fig. 3.1.

```
# Time stamp = 70
# comment: everything is fine here
1 5 10 20 60 5
2 5 5 25 15 50
3 10 10 50 15 15
4 60 15 12.5 9.5 3
5 20 19.7 20.3 30 10
6 7 35 33 20 5
7 1 3.5 5 20 70.5
8 1 1 1 1 96
9 1 90 2 2 5
```

Figure 3.1: A block of probability values for timestamp = 70

In Fig. 3.1, note the following:

- In each block, the first number in every row represents the site ID for which the probability data in listed.
- The probabilistic event data is placed after the site ID in the order of N S L H V events.
- Any row that starts with a # symbol is considered a comment.
- Each block always starts with a comment that mentions the timestamp.

Probability Data Error Checking

The probability data for a given timestamp must adhere to the following two rules, otherwise it is not processed by the algorithm:

- In every row, the values corresponding to all the events must be ≥ 0.0.
- The sum of the values for a site must be equals to 100. Each row must then have a sum equals to 100%.

If any one or both of the above rules are violated during error checking, the whole timestamp data is ignored and the processing moves forward to the next available timestmap. Fig. 3.2 gives two examples of invalid or erroneous data sets.

```
# Time stamp = 10
# comment: sum is 102% for
site 2, this time stamp is
discarded
1 5 10 20 60 5
2 5 5 25 50 17
3 3 10 27.5 55 9.5
4 10 10 15 50 15
5 20 19.7 20.3 30 10
6 7 35 33 20 5
7 1 8.5 10 20 60.5
8 1 1 1 1 96
9 1 90 2 2 5

# Time stamp = 30
# comment: one of the values
< 0 for site 4, this time
stamp is discarded
1 5 10 20 60 5
2 5 5 25 50 15
3 3 10 27.5 50 9.5
4 10 10 -15 50 30
5 20 19.7 20.3 30 10
6 7 35 33 20 5
7 1 8.5 10 20 60.5
8 1 1 1 1 96
9 1 90 2 2 5
```

Figure 3.2: Two blocks of probability values to show invalid data

3.2.2 Intelligence Data

In our implementation, this data is placed in a text file named *SiteIntelligenceData*. This file too contains blocks of data for separate, increasing timestamps. An example of a block of data is as shown in Fig. 3.3.

In Fig. 3.3, the following points are noteworthy:

- Like the probability data, in each block, the first number in every row represents the site ID.
- The intelligence data is placed after the site ID in the order of N S L H V events.
- Any row that starts with a # symbol is considered comment. The start of each block contains a comment that mentions the timestamp of the current data set.
- Because we assume that the intelligence data is almost constant with respect to probability data, all the timestamps in its file would be less than or equals to that in the probability file.

```
# Time stamp = 4
# comment: everything is
fine here
1 0.5 2 2 1 0.5
2 1 1 1 1 1
3 1 1 2 2 1
4 0.5 1 3 2 0.5
5 0.1 0.5 2 1 0.5
6 1 1 1 1 1
7 2 1 2 2 0.5
8 1 1 1 1 1
9 1 3 2 2 1
```

Figure 3.3: A block of intelligence data values for timestamp = 4

Intelligence Data Error Checking

The intelligence data for a given timestamp must abide by one rule mentioned below, otherwise it is ignored by the algorithm:

- In every row, each value that corresponds to a event type must be ≥ 0.0.

Fig. 3.4 is an example of invalid intelligence data set.

```
# Time stamp = 2
# comment: value is negative
for site 3, this time stamp
is discarded
1 0.5 2 2 1 0.5
2 1 1 1 1 1
3 -1 2 2 2 1
4 0.5 1 3 2 0.5
5 0.1 0.5 2 1 0.5
6 1 1 1 1 1
7 2 1 2 2 0.5
8 1 1 1 1 1
9 1 3 2 2 1
```

Figure 3.4: A block of intelligence values to show invalid data

This invalid data set is kept in the file only as a means to display the error checking mechanism of the algorithm in the coding implementation. If no valid intelligence data set is found, then the probability data is not updated with intelligence values.

3.2.3 Ground Truth

Ground Truth is the actual event that occurred during a specific timestamp. Because the alert stations in our case are unsupervised, they do not have access to the ground truth. Therefore, we consider the ground truth to be the most probable value for each station (for a specific timestamp) breaking any ties by order (whichever comes last). All our inferred truth values are put in a text file named *SiteGroundTruth*. We follow the following steps to generate the ground truth file:

1. Read each valid timestamp data block from the probability distribution file.
2. In each row, for each alert site/station, read values for each of the five event type.
3. The event corresponding to the maximum value from the previous step is the ground truth.
4. Write this event to the ground truth file.

Fig. 3.5 gives an idea of what the values look like.

```
# Time stamp = 70
1 H
2 V
3 L
4 L
5 L
6 S
7 H
8 V
9 S

# Time stamp = 75
1 L
2 S
3 L
4 L
5 H
6 L
7 L
8 L
9 S
```

Figure 3.5: Two blocks of ground truth values

In Fig. 3.5, note the following:

- For this file too, a # symbol at the start means a line of comment. For every block, its corresponding timestamp is written to the ground truth file.

- Consider the valid timestamp : 70, its distribution values for site 1 are 5 10 20 60 5. In this set of values, the maximum is 60 which corresponds to the event type human (H).

3.3 UGS States

Each UGS has four different states in which it transitions. The states are as follows:

1. Idle: this state is denoted by I and indicates that the UGS is inactive or idle. A UGS starts in this state.
2. Alert: this state is denoted by A and indicates that the UGS has triggered an alert.
3. Service: this state is denoted by S and indicates that the UGS that triggered an alert is being serviced by a UAV, i.e. the UAV is dwelling over or around the UGS location.
4. Undecided: this state is denoted by U and indicates that a UGS is currently inactive.

These states and related transitions are shown in Fig. 3.6.

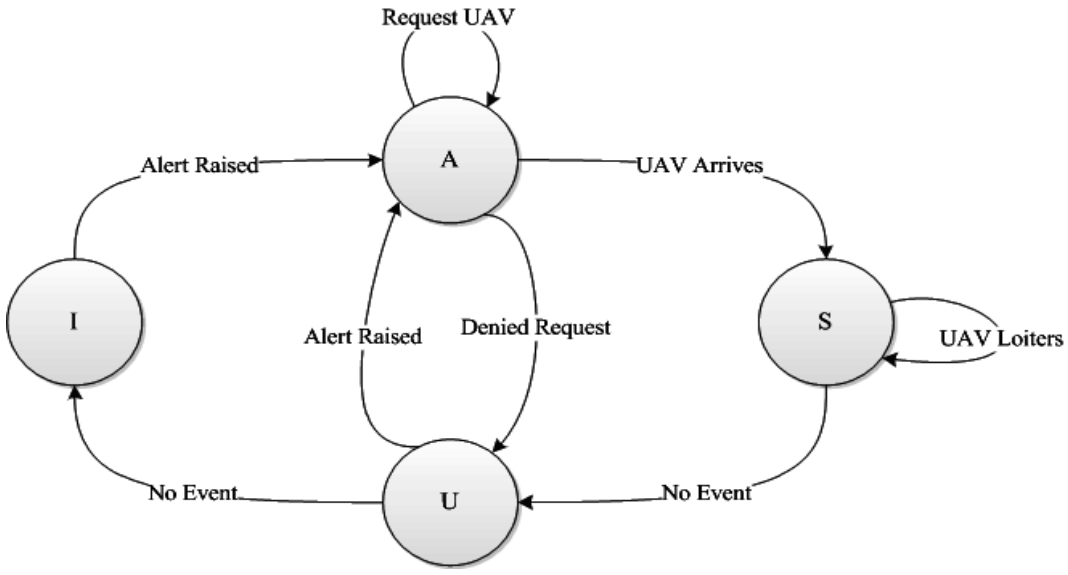


Figure 3.6: Diagram depicting UGS states

From Fig. 3.6 it is obvious that once an alert is triggered, a request for a UAV is continuously generated until either the site is serviced (by a UAV) or the request is denied (for example by a

control system or algorithm because the request has timed out). Additionally, once a site is serviced and there are no further events, the UGS transitions to the undecided state. From this state, it can either move to either the idle state or the alert state whichever occurs first.

3.4 UAV States

Each UAV has four states among which its transition occurs. The states are presented below:

1. Idle: this state is denoted by I and indicates that the UAV is simply patrolling.
2. Service: this state is denoted by S and indicates that the UGS that triggered an alert is being serviced by a UAV, i.e., a UAV is dwelling over or around the UGS location.
3. Continue Direction: this state is denoted by CD and indicates that a specific UAV will continue in its current direction to move to a new station/site.
4. Reverse Direction: this state is denoted by RD and indicates that a specific UAV will reverse in its current direction to move to a new station/site.

These states and related transitions are shown in Fig 3.7.

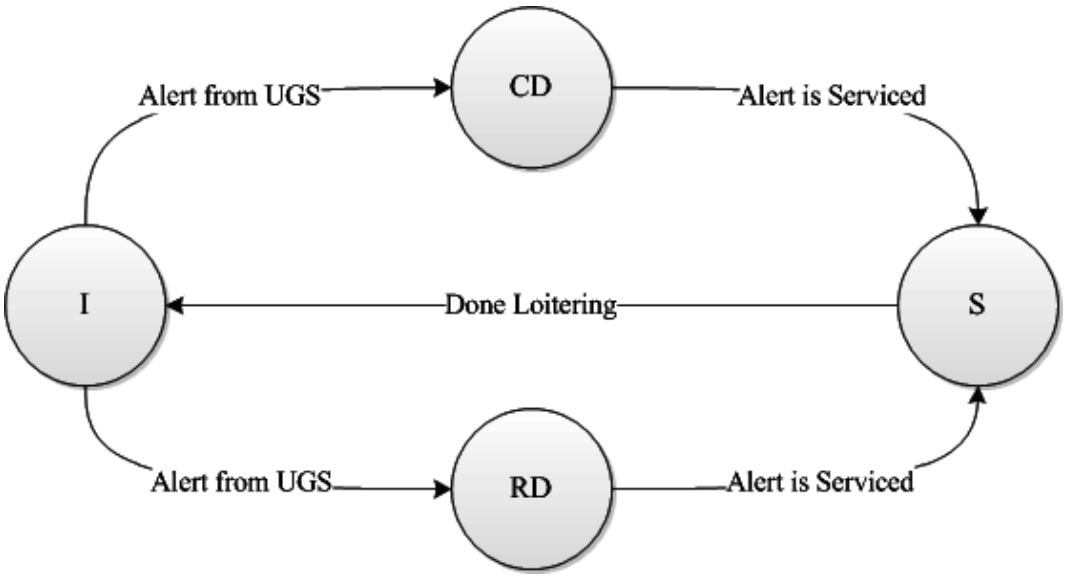


Figure 3.7: Diagram depicting UAV states

We assume that each of our two UAVs initially loiter over the two extreme edges, i.e. the first and the last alert station. Once a UAV is done servicing an alert, it would then move back to the idle state which could indicate that it is either loitering in the same station or it had moved back to its initial location/position or it is in a brief transition state before it moves to a new location for servicing.

We have also assumed the border to be a long straight line. That is why a specific UAV would be traveling in a straight line and would either continue or reverse its current direction to change its alert station location. We provide details about a UAV's direction and the way it changes in the Section 3.5.

3.5 UAV Direction (from graphical point of view)

Each UAV has to move a fix number of units, i.e., x-coordinates in the graphical display panel (discussed in Chapter 4). The direction of the UAV, either positive or negative or zero, plays an important role in calculating these number of units. The basic steps for calculating the next direction for a UAV is listed below:

Site IDs: unique consecutive numerical values starting from 1 assigned to each alert site/station (the leftmost station's ID is 1),

Let p = previous Site ID of a UAV, c = current Site ID of the UAV and d = next direction value.

The value of d is, if required, is calculated as

$$d = (p - c) \,/\, abs(p - c) \text{ where } \textit{abs} \text{ represents the absolute value} \tag{3.4}$$

Hence, the set of values d takes is {-1, 0, 1}.

1. if p - c $>$ 0 then d is positive and its value is calculated from Equation 3.4
2. if p - c $<$ 0 then d is negative and its value is calculated from Equation 3.4
3. if p = c then d is zero

Based on the values of d, each UAV could either continue its direction or reverse its direction or loiter in the same location.

3.6 Value Functions

Value functions are the functions that are computed during the decision making process by our algorithm. Let us define a few terms in order to explain the functions.

1. Probability of Events (p) = $(p_j)_i$ where j is the number of event types as described in Section 3.2 and i is the total number of alert stations. It can be defined as:

$$p : E \rightarrow [0, 1] \tag{3.5}$$

It follows from 3.5 that the probability distribution is every event type mapped to a value between 0 and 100%.

2. Intelligence Data (I) can be defined as:

$$I : E \rightarrow \mathbb{R}^+ \tag{3.6}$$

It follows from 3.6 that the intelligence data is every event type mapped to a positive real value ≥ 0.

3. Weight of Events (W) also described in Section 3.2. It can be defined as given in Function 3.2.

4. Gain Matrix (GM) is a 5×5 matrix that represents the comparison of observed events (as reported by a UGS) against the ground truth (the actual event type). If $n = |E|$ is the number of events, then GM is defined as:

$$GM : N \times N \rightarrow \mathbb{R}^{+\ n \times n} \tag{3.7}$$

Now, we define three value functions: gain, cost and utility in succeeding subsections.

3.6.1 Gain Function

Gain Function (GF) of sensed data is a measure of how much a system gains by being at that specific location. Gain is based on the weight of the events and the gain matrix combined with the probability distribution and if available the intelligence data. Hence, the gain function can be defined as:

$$GF = (p, I, W, GM) \tag{3.8}$$

3.6.2 Cost Function

Cost Function (CF) is the cost of not using a UAV in other alert stations when it is loitering over a given station. It is the maximum value of gain at all other stations except the current station. Hence, cost function can be defined as:

$$CF = \max(\text{gain of all sites other than the current one}) \tag{3.9}$$

3.6.3 Utility Function

Utility is simply the numerical value of benefit obtained by engaging a UAV at a specific alert site. Utility for a site is the calculation:

$$Utility = \text{(gain - cost) for that site} \tag{3.10}$$

3.7 Influence Diagram Concepts in Decision Making

The influence diagram of the decision algorithm that we have implemented is shown in Fig. 3.8.

Fig. 3.8 presents the following significant points:

- The whole decision making is performed in two parts: the first part is evaluating function values at the individual alert stations and the second part is comparing these values in a centralized manner.
- The central system can be a server which operates on input information and sends commands to each UAV about their next location.
- Although the utility values are calculated at each node/alert station and is distributed, the actual decision making is centralized.

Part 1: Processing at Alert Station i

1. In this part, the data read from the sensors is probabilistically categorized into five event types. Once categorized, this data forms the probability of events and is updated using the intelligence coefficients. The details of how the update is performed in discussed in Section 4.5. A utility value, for every timestamp that has been recorded, is calculated following a series of steps from the probabilistic distribution data.
2. The above step is carried out in all alert stations to obtain a collection of utility values.

Part 2: Processing at Central System

1. The central system takes all the utility values from all stations and applies MEU principle to choose where to send each of the two UAVs. The decisions that are made are to send UAV1 to site x, to send UAV2 to site y and not to send the UAVs to the remaining sites.
2. The final value is the presence of the two UAVs at specific alert stations.

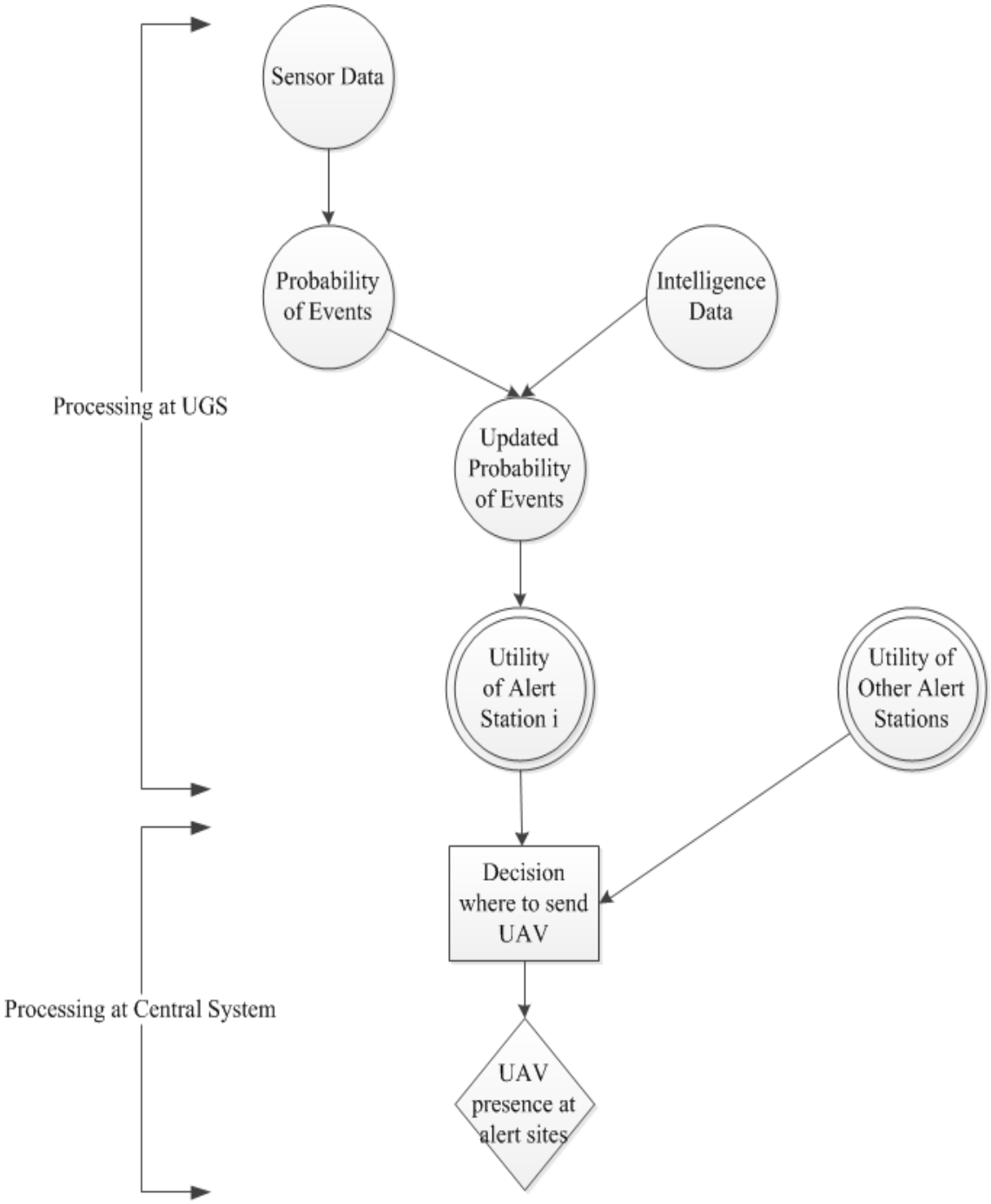


Figure 3.8: Influence Diagram of our implementation

CHAPTER 4

SOFTWARE IMPLEMENTATION

In this chapter we present topics related to the actual coding level implementation of our thesis project.

4.1 Programming Environment and Setup

The project code is implemented in Java programming language. Java is a high level programming language much similar to C++ but is strictly object oriented. A .java file is compiled into byte-codes stored in .class file. This file is then executed by the Java Virtual Machine (JVM) installed in any operating system. The java runtime environment version used in our project is 1.7.0.

The Integrated Development Environment (IDE) employed for project development is Eclipse (version: Kepler Service Release 1). The code has been vesion controlled using TortoiseSVN tool integrated to the IDE itself.

4.2 Code Structure

The source code for the project is divided into two categories or from a IDE point of view into two source packages:

1. mainPackage: this package contains all the core logic classes including the MainClass class to start running the project.
2. drawingPackage: this package contains all the component classes that are concerned with displaying the graphical panel of the project.

Additionally, we have two text files used as input to process the probabilistic distribution and intelligence coefficients. When the project is running, a ground truth file is also created.

4.3 Class Diagrams

A class diagram displays relationships among classes and interfaces within a project or package. In our case, we are only concerned with java class as the project does not define any interface. The diagram represents a java class using a rectangular box. Such boxes are partitioned into sections. The first section contains class name and its package information. The second section contains the attributes or simply called the variables of this class. The third section contains all the methods declared within the class. Three different class diagrams have been created for our project:

1. Class-Diagram-drawingPackage: this diagram shows the java classes, their attributes and methods and their relationship with each other within the drawingPackage package or folder.

2. Class-Diagram-mainPackage: this diagram shows the java classes, their attributes and methods and their relationship with each other within the mainPackage package or folder.

3. Class-Diagram-wholeProject: this diagram is minimized to show the overall project structure and only contains the first section of all available java classes and their relationship with each other.

In a class diagram, the actual relationships between classes is displayed using arrows or arcs. The most common relationships displayed in our diagrams are association and dependency. These relationships are summarized below:

1. Association relationship: it is illustrated by the use of a solid straight line with the association value and its multiplicity printed as a label of the line. An association value is the instances by which any two classes are related and multiplicity is a count of how many such instances of a class (pointed to by the arrow head) are created and used in the other class.

2. Dependency relationship: it is illustrated by the use of a dotted straight line and implies a weaker relationship between the classes involved. In such relationship a class A (the one pointed to by the arrow head) is dependent on another class B by the virtue of using some variables or instances of class B as method parameters and also that a change in the structure of class B affects class A.

Next, we present the aforesaid class diagrams in three separate subsections.

4.3.1 Class Diagram for Drawing Package

In addition to other relations shown, we have an inner class called UAVTimerTask inside the class DrawingPanel represented by a *+ symbol inside a circle at the parent class* DrawingPanel's end.

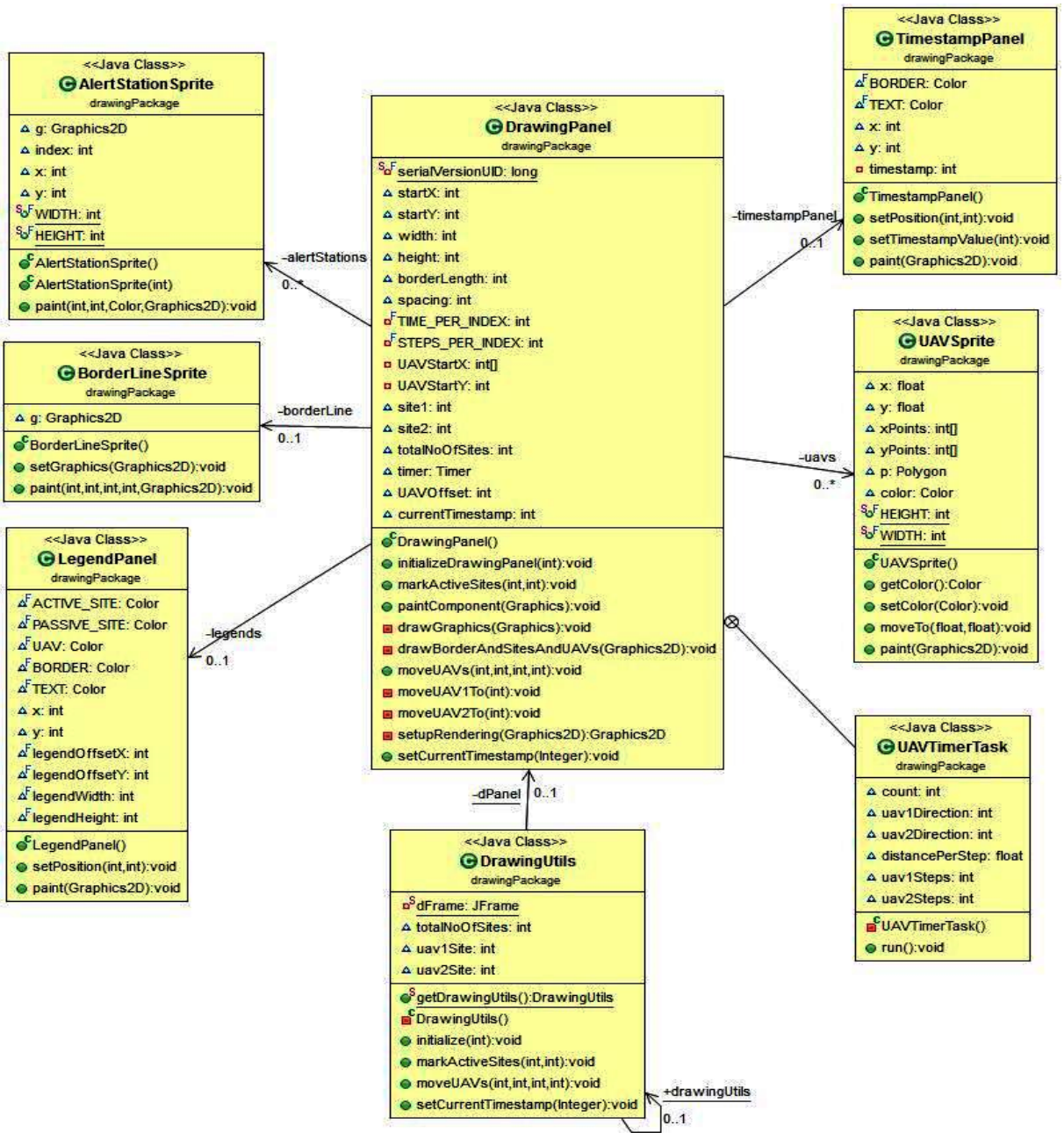


Figure 4.1: Class Diagram for Drawing Package

4.3.2 Class Diagram for Main Package

In the Main package, the UtilityValueClass contains all the core logic and steps and hence builds relations with all the other classes in the package.

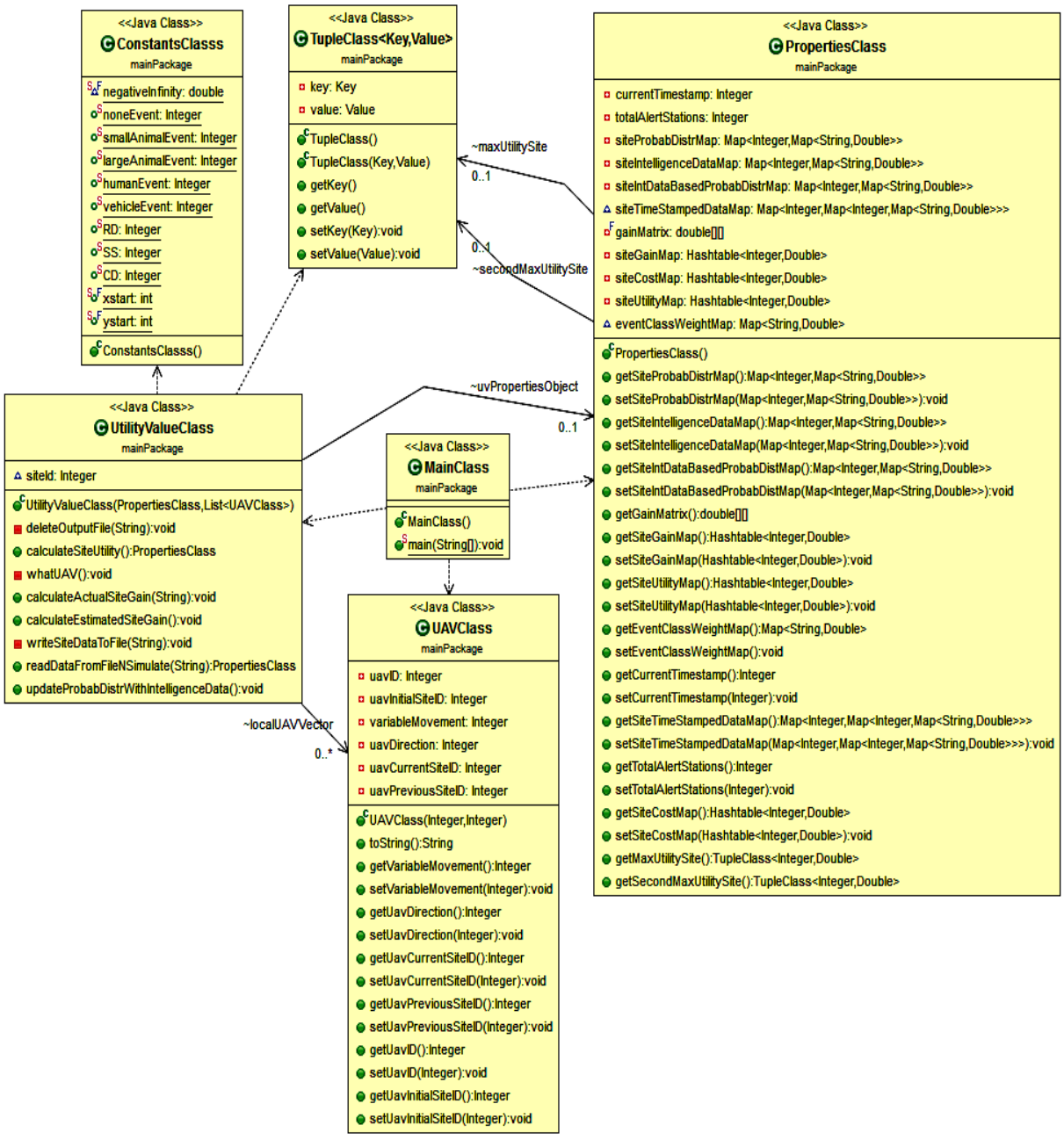


Figure 4.2: Class Diagram for Main Package

4.3.3 Class Diagram for Whole Project

From this diagram, it can be inferred that for graphics related operations the UtilityValue class in the main package calls the methods contained in the DrawingUtils class which then delegates the logic functionality to the DrawingPanel class.

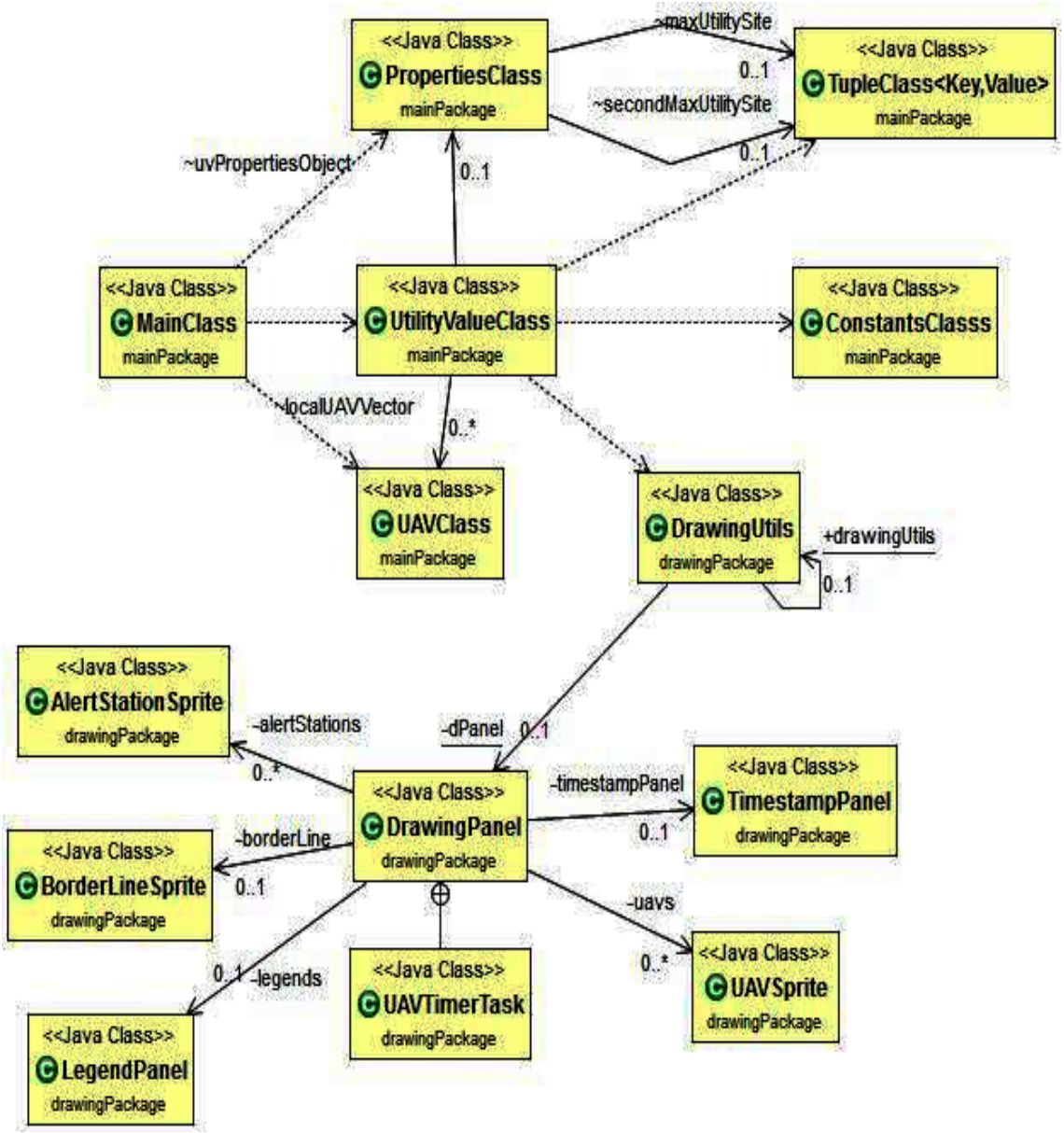


Figure 4.3: Class Diagram for Whole Project

4.4 Implementation Phases

We have used an incremental approach to code our project. The project has been implemented in successive phases with each phase adding to or upgrading capabilities or flexibilities in solving the problem more effectively. There are three such phases or stages, each of which is discussed in the sections that follow.

4.4.1 Single UAV - Multiple Alert Stations

The implementation started by using a single UAV to patrol all the alert stations. In this phase, we calculate a utility value for all stations but choose only one value at the end: the maximum utility. The UAV would be sent for patrolling to the corresponding site.

4.4.2 Two UAVs - Multiple Alert Stations with Fixed Area of Surveillance

The implementation in the first phase is extended in this phase by adding another UAV to the patrolling operations. This would again involve calculating the utility values. One station with the best utility is selected for patrolling and one of the UAVs is sent to the location. A restriction of this algorithm was that each UAV would be assigned only half or approximately half the stations for monitoring: for UAV1 the number of sites = integer(total number of sites / 2) and the rest of the sites to UAV2. There would be a boundary between patrolling areas of each UAV. Each UAV would cater to an alert only if it falls within its monitoring range.

4.4.3 Two UAVs - Multiple Alert Stations with Changing Area of Surveillance

The fixed boundary limitation in the second phase is removed and replaced with a moveable boundary that allows for each UAV to cater to alerts in almost any station. The only restriction is that while the UAVs can move from one active alert station to another, they do not cross-over. Each UAV has to choose the active alert station that is closest to its initial position (which is the first station for UAV1 and last station for UAV2). Hence, in every timestamp, two stations with two highest utility values are chosen. Currently, the order of choice is that UAV1 chooses its next station for patrolling which is followed by a choice by UAV2. Although, the reverse can be done without affecting the final result. Because each UAV "greedily" chooses an active alert site closest to its starting position, the process is analogous to stretching an elastic material from a fixed point.

4.5 Implementation of Decision Making with Pseudo-codes

This section presents as step-wise description of the way our algorithm processes the data and uses it to decide two best sites for patrolling (for a specific timestamp).

Before we start with these steps, it is important to know that the program or algorithm reads the probabilistic distribution file and intelligence coefficients file. If any data block (again for a specific timestamp) does not confirm to the rules of validity listed in Section 3.2.1 and Section 3.2.2 for probability and intelligence data respectively, then an error message is printed out in the eclipse IDE console and the processing moves on to the next available data block.

4.5.1 Updating Probabilistic Distribution with Intelligence Data

If intelligence data is available and is valid, an update of probability values is performed based on the former values. The theoretical details and a mathematical example about these are provided in Section 2.3.3. Within the program, an extra check is done that ensures that the number of alert stations, i.e. the number of rows for corresponding timestamp in both the files are the same. In cases where the numbers do not match, the update process is not followed through and simply the probability data block is used for further processing.

As mentioned in Section 2.3.3, in each timestamp, for each row of data, an update is the process of multiplying corresponding values for each event type in both the probabilistic and intelligence data files. These data are stored in a Java LinkedHashMap which is of the format Map<Integer, Map<String, Double>>. In this map, the data that is stored is <siteID, <eventType, probability/intelligence value>> for each site.

Once the values are updated, they are normalized to obtain final values. A pseudo-code of this specific process is shown below:

```
// for a specific timestamp, perform these operations
for (loop through all the sites) {
siteId = getSiteID();
probabilityDataMap = getProbabilityMap();
intelligenceDataMap = getIntelligenceCoffiecients();
```

```
// updating via multiplication of corresponding event types
for (loop through each row of a data block of the probability file) {
updatedProbabDistrValue = probabilityDataMap.getValue() * intelligenceDataMap.getValue();
// sum is used during normalization
sumOfupdatedProbabDistrValue += updatedProbabDistrValue;
// Store the value temporarily before normalization
updatedProbabDataMap.put(siteId, updatedProbabDistrValue);
}

// Normalize the augmented value
for (loop through the updated Probability Data Map) {
normalizedValue = updatedProbabDataMap.getValue() * (100 / sumOfupdatedProbabDistrValue);
// these are final values for this siteId/row of the data block
newUpdatedProbabDataMap.put(siteId, normalizedValue);
}

// Store the final values for this row/site in another Java Map
siteIntDataBasedProbabDistrMap.put(siteId, newUpdatedProbabDataMap);
}
```

4.5.2 Calculating Site Gain Values

Each site has a gain value associated with it. The calculation of site gain is based on probability values, intelligence coefficients, weight of the events (as given by 3.3) and gain matrix.

To re-emphasize, a gain matrix is a 5×5 matrix with the values in the rows representing observed events and that on the columns representing ground truth. Each value represents a gain made any site for the claimed or observed event versus the actual ground truth. The values across the main diagonal (going from top-left to bottom-right) is the weight of events. All the other values are relative to these weight values and represents how much we would lose by wrongly categorizing an event. For example: in the third row if L (Large Animal) is correctly categorized, then the gain is 3. But, if we wrongly categorize it by saying it is a S (Small Animal), then from the second row and third column we have the gain value to be only 2.5. Fig. 4.4 shows the gain matrix that we have used in our calculations:

Ground Truth

Observed Event	N	S	L	H	V
N	1	1.5	2.5	2	1
S	1	2	2.5	3	2
L	1	1.5	3	3.5	3
H	1	1.5	2	4	4.5
V	1	1.5	2	3.5	5

Figure 4.4: Gain Matrix used in Decision Making

To calculate the gain values for each alert station, we multiply the corresponding event types from an updated probabilistic distribution and each row of the gain matrix and sum the multiplied results. The same process is repeated for each row of the matrix in order to obtain partial sums. These sums are representative of gains made if the observed event is any one of the types from the event space. All the partial sums are added and divided by the total number of events to get the final result. This calculation can be summarized by the following expression:

$$(\sum_{i=1}^{N}(\sum_{j=1}^{N} E_{site_j} \times E_{gain-matrix_j}))/N \tag{4.1}$$

where N is the total number of events in the event space,

j is the columns (of a row) within the gain matrix,

i is the rows within the gain matrix,

E_{site_j} is the value of an event from the probabilistic distribution of a site and

$E_{gain-matrix_j}$ is the value of the same event from the gain matrix.

A pseudo-code of this specific process is shown below:

```
// perform this operation on the updated probability map
for (loop through the updated site probability map) {
siteId = getSiteID();
probabilityDataMap = getUpdatedProbabilityMap();

// For each site and each row of gain matrix:
// (i) multiply event values and then sum those values > partialSum
// (ii) add all partialSum and divide them by the no of events
```

```
for (loop through each row > each column of the Gain Matrix) {
for (loop through each column/event type of a row in the probability map) {
// express probability in percentage: divide by 100
partialGainValue += (probabilityDataMap.getValue() / 100) * gainMatrix[firstDim][secondDim];
secondDim += 1;
}
firstDim += 1;
// reset secondDim for the next row
secondDim = 0;
}

// Average the partialGainValue by dividing by the total number of events
siteGainMap.put(siteId, partialGainValue / totalNoOfEvents);
// Reset first dim for a new site
firstDim = 0;
// Reset partial gain for a new site
partialGainValue = 0.0;
}
// Store the gain for this site in a Java Map
setSiteGainMap(siteGainMap);
```

4.5.3 Calculating Site Cost Values

As described in Section 3.6.2, the cost of a site is the loss of all other sites not to be serviced by a UAV. If a UAV services as site, it cannot service the remaining sites. So, the cost of any site i is the maximum loss of not servicing a site other than i. In other words, the cost of site i is the maximum gain of all the sites other than i. The pseudo-code of this process can be shown as:

```
for (loop through the site gain map, original entry) {
// cost: maximum among all gains (except this one)
for (duplicate entry/loop for site gain map) {
// ignore the site currently being processed
if (originalSiteEntry.getSiteID() == duplicateSiteEntry.getSiteID()) continue;
// update with a new maximum value if needed
if (cost < duplicateSiteEntry.getValue()) { cost = duplicateSiteEntry.getValue();}
}
```

```
// store the cost in a map per siteID
siteCostMap().put(originalSiteEntry.getSiteID(), cost); }
```

4.5.4 Calculating Site Utility Values

As described in Section 3.6.3, utility is the target value obtained after all our calculations. It is expressed as difference between the gain and cost values of a site and its pseudo-code can be shown as:

```
for (loop through the site gain map) {
// cost for current site is calculated here

// Utility (of a site) = gain (of that site) - cost(of the same site)
utility = getGainValue() - cost;

// Store utility in another Java map
SiteUtilityMap().put(siteEntry.getKey(),utility);
}
```

4.5.5 Applying MEU Principle

As the Maximizing Utility (MEU) principle demands (refer Section 2.3.4), we select the utility that maximizes our benefit. Since we have two UAVs, we therefore select the two highest values from a set of utilities (for a given timestamp). These are the sites to which the two UAVs are commanded to move for monitoring in the current timestamp. The pseudo-code of this process is shown below:

```
printout("Applying MEU principle for time stamp: " + uvPropertiesObject.getCurrentTimestamp());
// determine 1st and 2nd maximum utility values
// selectedSiteA: site with maximum utility, selectedSiteB: site with second maximum utility
for (loop through the utility values map) {
if (maxUtility < utilityMap.getValue()) {
// Just swap values
selectedSiteB = selectedSiteA;
secondMaxUtility = maxUtility;

selectedSiteA = utilityMap.getSiteID(); maxUtility = utilityEntry.getValue();
} else if (secondMaxUtility < utilityMap.getValue()) {
```

```
selectedSiteB = utilityMap.getSiteID(); secondMaxUtility = utilityEntry.getValue();
}
}
```

4.5.6 UAV locations per Timestamp

Once the two best sites for monitoring are decided, the next step to be performed is to determine which UAV goes to which of the two active alert sites. The theoretical aspect of this is discussed in Section 4.4.3 and its pseudo-code is presented below:

```
// find which of the active alert site is closer to uav1
// uav1ToAlertActiveSite1: distance in terms of site ID between UAV1 & active alert site 1
// uav1ToAlertActiveSite2: distance in terms of site ID between UAV1 & active alert site 2
if (uav1ToAlertActiveSite1 < uav1ToAlertActiveSite2) {
uav1.setUavCurrentSiteID(activeAlertSite1.getSiteID());
uav2.setUavCurrentSiteID(activeAlertSite2.getSiteID());
} else {
uav1.setUavCurrentSiteID(activeAlertSite2.getSiteID());
uav2.setUavCurrentSiteID(activeAlertSite1.getSiteID());
}
```

4.5.7 Calculating Actual Gain and Cost

While the UAVs are being positioned over the two alert stations with the highest utility values, we also calculate other gain and cost values by reading from the ground truth file. The values are the actual gain and the actual cost of moving one UAV to site I and the other to site II. These values are solely for the purpose of comparison with the estimated gain and cost values and do not currently form a feedback into the algorithm. The pseudo-code of this calculation is as follows:

```
while (read each line from ground truth file with variable strLine) {
// within the file find the data block for current timestamp
if(timestampReadFromFile != currentTimestamp)
continue; // if no match above, do not process lines below this

// the split results in two elements with siteID at index 0 and actual event at index 1
grdTruthArray = strLine.split(" ");
siteActualMap.put(siteID, mostProbableEventWt);
```

```
}

// set the values for actual gains
selectedSiteIActualGain = siteActualMap.get(uvPropertiesObject
.getMaxUtilitySite().getKey());
selectedSiteIIActualGain = siteActualMap.get(uvPropertiesObject
.getSecondMaxUtilitySite().getKey());

for (loop through siteActualMap with variable actualEntry) {
if (this is not the site with max utility) {
if (selectedSiteIActualCost < actualEntry.getValue()) {
selectedSiteIActualCost = actualEntry.getValue();
}
}
if (this is not the site with second max utility) {
if (selectedSiteIIActualCost < actualEntry.getValue()) {
selectedSiteIIActualCost = actualEntry.getValue();
}
}
}
```

CHAPTER 5

RESULTS AND USER DOCUMENTATION

This chapter presents the results of processing displayed in the eclipse IDE console. We also present the graphical panel that displays the alert generated in stations and the resulting motion of UAVs. In addition to these, we have assembled a brief documentation about configuring and running the project.

5.1 Console Output

Error checking for the validity of data is an important part of our algorithm. In the following sections, we look at error checking as well as valid data processing of both probabilistic distribution and intelligence coefficient files.

5.1.1 Processing of Valid and Invalid Intelligence Data

The output shown below corresponds to the processing of data shown in Fig. 3.4 and then in Fig. 3.3. Because the data at timestamp 2 is invalid, only the next data block for timestamp 4 is considered. No further intelligence data blocks added as its values are considered almost constant during the processing of probabilistic distribution data.

```
Time stamp:2
ERROR: Invalid data format for current time stamp in the intelligence data file.
Moving to next available time stamp
**********************************************************
Time stamp:4
intelligence data map for time stamp: 4 is:
{1={N=0.5, S=2.0, L=2.0, H=1.0, V=0.5}, 2={N=1.0, S=1.0, L=1.0, H=1.0, V=1.0},
3={N=1.0, S=1.0, L=2.0, H=2.0, V=1.0}, 4={N=0.5, S=1.0, L=3.0, H=2.0, V=0.5},
5={N=0.1, S=0.5, L=2.0, H=1.0, V=0.5}, 6={N=1.0, S=1.0, L=1.0, H=1.0, V=1.0},
7={N=2.0, S=1.0, L=2.0, H=2.0, V=0.5}, 8={N=1.0, S=1.0, L=1.0, H=1.0, V=1.0},
```

```
9={N=1.0, S=3.0, L=2.0, H=2.0, V=1.0}}
```

5.1.2 Processing of Invalid Probabilistic Distribution Data

The output shown below corresponds to the processing of data shown in Fig. 3.2.

```
Time stamp:10
ERROR: Invalid data format for current time stamp in the probability file.
Moving to next available time stamp
**********************************************************
Time stamp:30
ERROR: Invalid data format for current time stamp in the probability file.
Moving to next available time stamp
**********************************************************
```

5.1.3 Processing of Valid Probabilistic Distribution Data

The output presented below corresponds to the processing of data shown in Fig. 3.1.

```
Time stamp:70
probability distr map for time stamp: 70 is:
{1={N=5.0, S=10.0, L=20.0, H=60.0, V=5.0}, 2={N=5.0, S=5.0, L=25.0, H=15.0, V=50.0},
3={N=10.0, S=10.0, L=50.0, H=15.0, V=15.0}, 4={N=60.0, S=15.0, L=12.5, H=9.5, V=3.0},
5={N=20.0, S=19.7, L=20.3, H=30.0, V=10.0}, 6={N=7.0, S=35.0, L=33.0, H=20.0, V=5.0},
7={N=1.0, S=3.5, L=5.0, H=20.0, V=70.5}, 8={N=1.0, S=1.0, L=1.0, H=1.0, V=96.0},
9={N=1.0, S=90.0, L=2.0, H=2.0, V=5.0}}
**********************************************************
int data based updated probab distr for time stamp: 70 is:
{1={N=2.0, S=16.0, L=32.0, H=48.0, V=2.0}, 2={N=5.0, S=5.0, L=25.0, H=15.0, V=50.0},
3={N=6.0606060606060606, S=6.0606060606060606, L=60.60606060606061, H=18.181818181818183,
V=9.090909090909092},
4={N=29.12621359223301, S=14.563106796116505, L=36.407766990291265, H=18.446601941747574,
V=1.4563106796116505},
5={N=2.287021154945683, S=11.263579188107489, L=46.42652944539737, H=34.305317324185246,
V=5.717552887364208},
6={N=7.0, S=35.0, L=33.0, H=20.0, V=5.0}, 7={N=2.203856749311295, S=3.8567493112947657,
L=11.019283746556475, H=44.0771349862259, V=38.84297520661157},
```

```
8={N=1.0, S=1.0, L=1.0, H=1.0, V=96.0},
9={N=0.352112676056338, S=95.07042253521126, L=1.408450704225352, H=1.408450704225352,
V=1.76056338028169}}
************************************
Data set added to SiteGroundTruth.txt for time stamp: 70
************************************************************
estimated sitegain for time stamp: 70 is:
{1=2.6419999999999995, 2=2.7600000000000002, 3=2.4757575757575756, 4=2.0334951456310684,
5=2.5923384791309316, 6=2.217, 7=2.962809917355372, 8=3.058, 9=1.658098591549296}
************************************
Utility values for time stamp:70 is
{1=-0.41600000000000037, 2=-0.2979999999999996, 3=-0.5822424242424242, 4=-1.0245048543689315,
5=-0.4656615208690682, 6=-0.8409999999999997, 7=-0.0951900826446277, 8=0.0951900826446277,
9=-1.3999014084507038}
************************************
Applying MEU principle for time stamp: 70 ::::::::::::::::::::::::
The sites with active alerts are site: 8 with utility value: 0.0951900826446277 and
site: 7 with utility value: -0.0951900826446277
********************************************************************
UAV ID: 1, current site: 7, previous site: 1 goes to Site: 7 for monitoring!!!
Estimated GAIN of moving to Site: 7 for time stamp: 70 is: 2.962809917355372,
Estimated COST of moving to Site: 7 for time stamp: 70 is: 3.058
********************************************************************
UAV ID: 2, current site: 8, previous site: 9 goes to Site: 8 for monitoring!!!
Estimated GAIN of moving to Site: 8 for time stamp: 70 is: 3.058,

Estimated COST of moving to Site: 8 for time stamp: 70 is: 2.962809917355372
************************************************************************************
Actual GAIN of moving UAV2 to Site: 8 for time stamp: 70 is: 5.0,
Actual COST of moving UAV2 to Site: 8 for time stamp: 70 is: 5.0
Actual GAIN of moving UAV1 to Site: 7 for time stamp: 70 is: 4.0,
Actual COST of moving UAV1 to Site: 7 for time stamp: 70 is: 5.0
************************************************************
FINISHED CALCULATION FOR TIME STAMP: 70
```

****
****

5.2 Graphical Display

We have created a simple graphical panel using Java 2D to display the concept of the project and its algorithm implementation more clearly. As we have mentioned before, our two UAVs start at the extreme edges of the border line where two alert stations are kept. The timestamp at the start is 0 (zero) and there are no alerts yet. This situation is shown in Fig. 5.1.

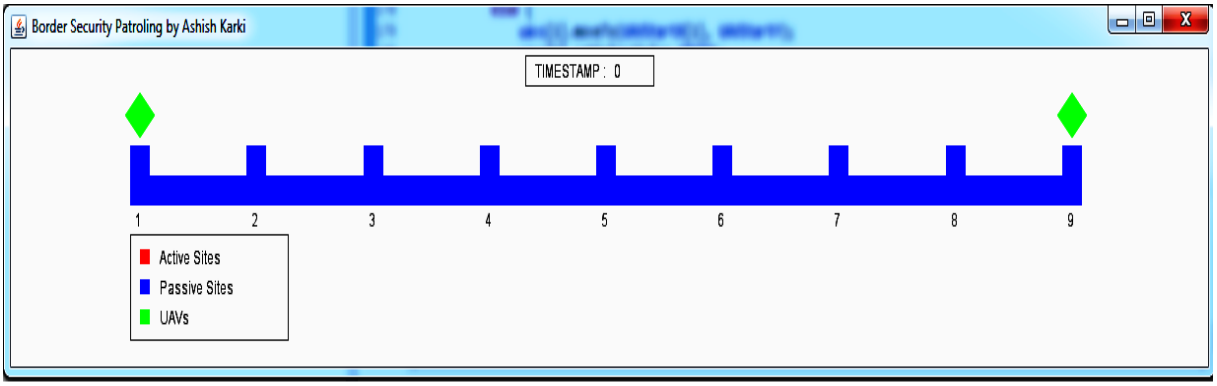


Figure 5.1: Graphical Display at timestamp = 0

If we consider our current probabilistic distribution file, the data for timestamps 10 and 30 are invalid and are ignored. The next valid data is for timestamp 70. The active alert sites in this case are sites 7 and 8. These sites are marked by a red rectangle (as opposed to a blue rectangle used to mark inactive or passive sites). The UAVs are commanded to move to these sites. This situation is shown in Fig. 5.2.

Now, consider the scenario shown in Fig. 5.3 for timestamp 85 where the UAVs are dwelling over the stations 7 and 8 again.

In the next valid timestamp which happens to be 90, the active sites/stations are 2 and 9. UAV1 is to travel to site 2 (as it is closer to its initial location which is site 1) and UAV2 is to travel to site 9 (as it is closer to its initial location which is site 9 itself). Fig. 5.4 displays the process of UAVs travelling to each these locations.

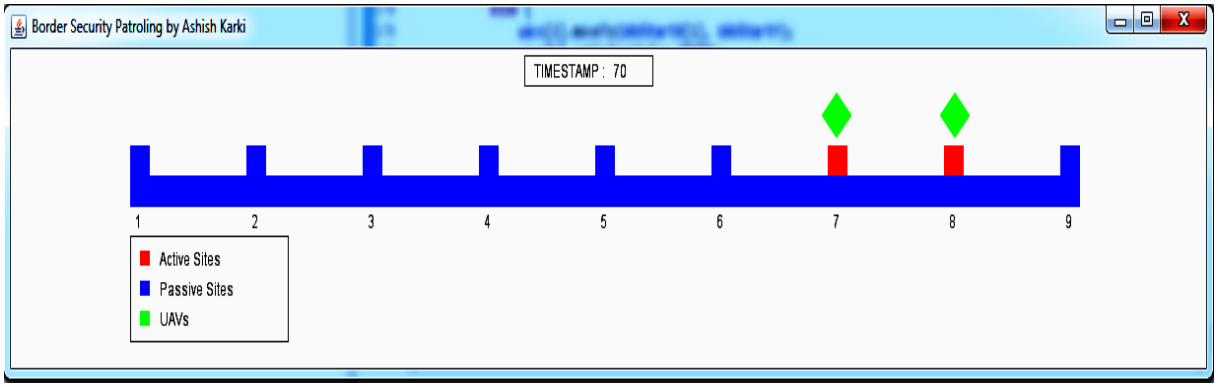


Figure 5.2: Graphical Display at timestamp = 70

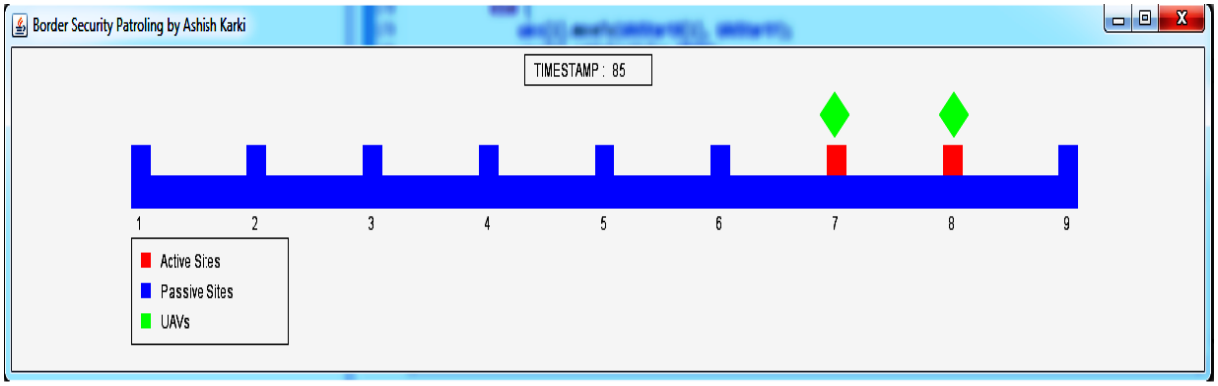


Figure 5.3: Graphical Display at timestamp = 85

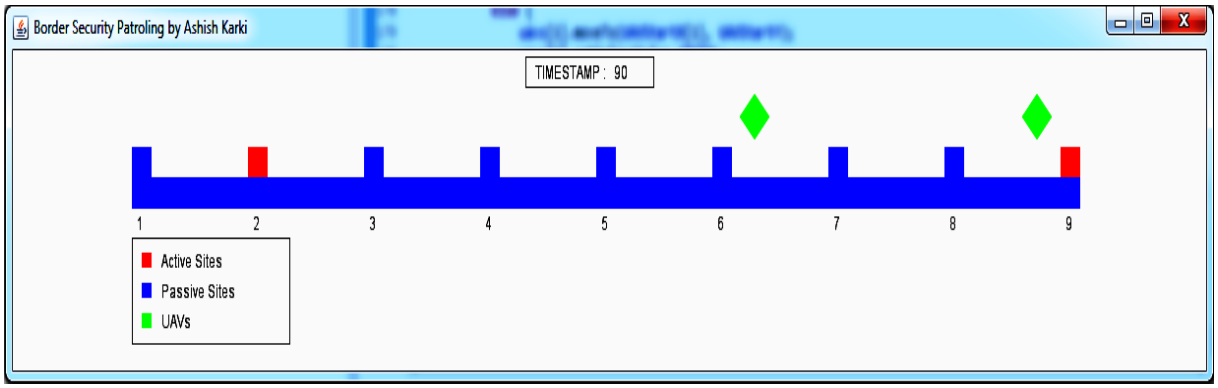


Figure 5.4: Graphical Display at timestamp = 90

5.3 User Documentation

This section contains a brief introduction to configuring the project, running it and its input and output file requirements.

5.3.1 Configuring and Running the Project

The project name is *BorderPatrolWithGraphics* which can be changed as required. It is written in Java (SE for core logic and 2D for graphics). Eclipse IDE has been used as a development platform. To open the project using eclipse, click the following menus in the sequence: File > New > Java Project. This opens the *New Java Project* dialog box. Uncheck the *Use Default Location* checkbox and browse for the location of the project. Once this is done, click on the Finish button to open the project. See Fig. 5.5 for hints.

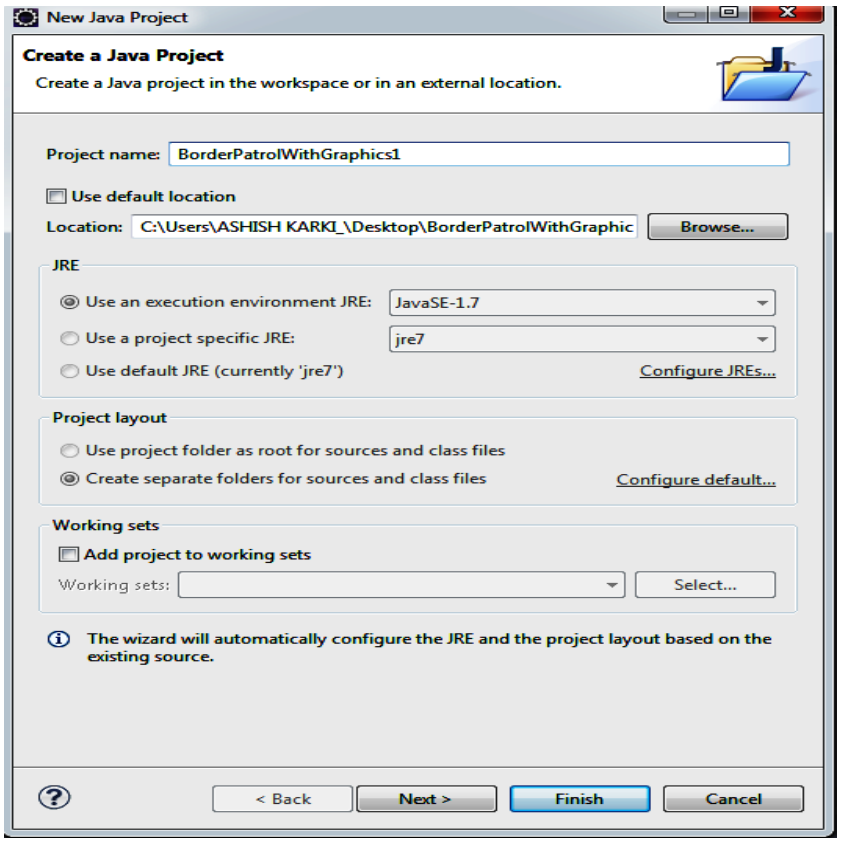


Figure 5.5: New Project Diaglog box in Eclipse

To run the project, one should select the Run option from the Run Menu or press the run button in the Shortcuts Menu (below the Main Menu bar). On a windows machine, pressing Ctrl + F11 gives the same result.

When the project is running, a graphics panel is displayed which shows the movement of UAVs and the alerts for each timestamp. The console of the eclipse IDE displays the details of calculations.

5.3.2 Input and Output files

The project requires at least one text file as an input containing the probabilistic distribution values in the format specified in Section 3.2.1. To augment this, another text file containing the intelligence coefficients, formatted as specified in Section 3.2.2, can be provided as an additional input. All the input files can be set for reading in the MainClass java file. Finally, a ground truth file currently named *SiteGroundTruth.txt* is created once the probabilistic distribution file is done reading. One should check the writeSiteDataToFile() method of the UtilityClass java file for details about configuring the name and contents of this file.

5.3.3 Code Structure and Flow

The execution of the project starts with the main() method inside the MainClass java class. The main logic is placed in the UtilityValueClass file and the flow from the main() method enters its readDataFromFileNSimulate() method. After all utility values for the current timestamp is calculated, a graphical panel is displayed with the flow going through the DrawingUtils class into the DrawingPanel class. To learn further about the packaging structure of the project and all the classes contained within it and their relationships, please refer to the class diagrams in Section 4.3 and the project code itself.

CHAPTER 6

CONCLUSION AND FUTURE WORK

In this thesis, we utilize the concepts from influence diagrams to build an algorithm which solves a decision problem. The problem is deciding where to send each UAV to dwell. This decision is based on determining two highest utility values from a set of values. And these values, in turn, are calculated for each alert stations/alert sites/UGSs from the updated probabilistic distributions. This process of calculation and decision is repeated for each valid timestamp data block that is read from the probabilistic distribution file. It is important to remember that the calculation of utility is performed at each UGS. A central system consisting of a computer server backed by one or more human operators collects all utility values and decides, using some criterion, the two sites chosen for monitoring during a specific timestamp.

Thus, utilizing an algorithm or computer program in border patrolling using UAVs and UGSs makes the overall process independent of human decision. Consequently, the decision making becomes less prone to human error and hence more efficient in terms of human resources.

The most important enhancement that can be done in the project is to make the scenario more realistic by considering the border as a region of overlapping areas. For example: when a UAV is servicing area i, it can also service area i-1 and i+1 simultaneously. Also, more than two UAVs can be integrated into the algorithm especially when a larger area or border line is under consideration. Within the algorithm's implementation, a restriction placed on UAVs that they cannot cross-over can be removed. This will increase the flexibility during operation and probably result in better response time to alerts.

APPENDIX A

CODE SNIPPETS INVOLVING DECISION MAKING

A.1 Calculating Estimated Site Gain

```
/**
 * @Description Estimate the gain of each site for a given time stamp
 *              (recorded in the original probability distribution file)
 */
public void calculateEstimatedSiteGain() {
/*
 * First, read and store intelligence data of each site. Note, that
 * there would be only one valid "intelligence data" group as this data
 * is considered to be almost constant throughout the process.
 */
if (uvPropertiesObject.getSiteProbabDistrMap().size() != uvPropertiesObject
.getSiteIntelligenceDataMap().size()) {
System.out
.println("ERROR: The length of probability distribution site entries "
" and intelligence data site entries do not match!!!!");
System.out
.println("No updating of probability with intelligence data will be done.");
System.out.println("************************************");

uvPropertiesObject
.setSiteIntDataBasedProbabDistMap(uvPropertiesObject
.getSiteProbabDistrMap());
} else {
// Update the probability distribution with intelligence data
```

```
updateProbabDistrWithIntelligenceData();
}

uvPropertiesObject.getSiteTimeStampedDataMap().put(
uvPropertiesObject.getCurrentTimestamp(),
uvPropertiesObject.getSiteIntDataBasedProbabDistMap());

// WRITE TO FILE HERE
writeSiteDataToFile("SiteGroundTruth.txt");

/*
 * Now, calculate the Gain for each site for all the sites. Perform
 * following operations for each site.
 */
// probability * intelligence data map with event class as key
Map<String, Double> updatedProbabDataMap = new LinkedHashMap<>();
// Map that temporarily stores the site gains
Hashtable<Integer, Double> siteGainMap = new Hashtable<>();
// First and second dimension values for the gain matrix array
int firstDim = 0; // rows within the array
int secondDim = 0; // columns within each row of the array
// partial gain value that gets added up to total gain for each site
double partialGainValue = 0.0;

for (Entry<Integer, Map<String, Double>> entry : uvPropertiesObject
.getSiteIntDataBasedProbabDistMap().entrySet()) {
siteId = entry.getKey();
updatedProbabDataMap = entry.getValue();

/*
 * For each site, go through each row of the gain matrix array and
 * multiply each column/event class in the row with the updated
 * probability distribution for the event class.
 */
```

```
for (@SuppressWarnings("unused")
double[] val : uvPropertiesObject.getGainMatrix()) {
for (Entry<String, Double> updatedProbabEntry : updatedProbabDataMap
.entrySet()) {
// express probability in percentage: divide by 100
partialGainValue += (updatedProbabEntry.getValue() / 100)
* uvPropertiesObject.getGainMatrix()[firstDim][secondDim];
secondDim += 1;
}

firstDim += 1;

// reset secondDim for the next row
secondDim = 0;
}

/*
 * We average the partialGainValue by dividing it by the number of
 * rows in the gain-matrix. The number of rows = number of events in
 * the eventClass i.e. {N,S,L,H,V} for this case.
 */
siteGainMap.put(siteId, partialGainValue
/ uvPropertiesObject.getEventClassWeightMap().size());
// Reset first dim for a new site
firstDim = 0;
// Reset partial gain for a new site
partialGainValue = 0.0;
}
uvPropertiesObject.setSiteGainMap(siteGainMap);
System.out.println("estimated sitegain for time stamp: "
+ uvPropertiesObject.getCurrentTimestamp() + " is:\n"
+ new TreeMap<>(uvPropertiesObject.getSiteGainMap()));
}
```

A.2 Calculating Site Utility and Applying MEU Principle

```
/**
 *
 * @Description Estimate the utility for a given time stamp (recorded in the
 *              original probability distribution file)
 * @return PropertiesClass object with the maps for site gain and site
 *         utility set to values
 */
public PropertiesClass calculateSiteUtility() {
// First calculate Gain for each site
calculateEstimatedSiteGain();

/*
 * Next, calculate the utility for each site.....................
 * Cost for a site = the maximum value of the gains of all sites other
 * than this site
 */
Double cost = 0.0;
Double utility = 0.0;
for (Entry<Integer, Double> siteEntry : uvPropertiesObject
.getSiteGainMap().entrySet()) {
/*
 * Following loop determines the maximum of the gains of all sites
 * other than the current site i.e determines the cost for current
 * site
 */
for (Entry<Integer, Double> duplicateSiteEntry : uvPropertiesObject
.getSiteGainMap().entrySet()) {
if (siteEntry.getKey() == duplicateSiteEntry.getKey())
continue;
// cost += duplicateSiteEntry.getValue();
if (cost < duplicateSiteEntry.getValue()) {
cost = duplicateSiteEntry.getValue();
}
```

```
}
uvPropertiesObject.getSiteCostMap().put(siteEntry.getKey(), cost);

// Utility (of a site) = gain (of that site) - cost(of the same site)
utility = siteEntry.getValue() - cost;

uvPropertiesObject.getSiteUtilityMap().put(siteEntry.getKey(),
utility);
// Reset cost for next loop
cost = 0.0;
}

System.out.println("Utility values for time stamp:"
+ uvPropertiesObject.getCurrentTimestamp() + " is \n"
+ new TreeMap<>(uvPropertiesObject.getSiteUtilityMap()));

// Using MEU (Maximizing Expected Utility) principle select the site
// with maximum utility
Integer selectedSiteA = -1;
double maxUtility = ConstantsClasss.negativeInfinity;

Integer selectedSiteB = -1;
double secondMaxUtility = ConstantsClasss.negativeInfinity;

System.out.println("Applying MEU principle for time stamp: "
+ uvPropertiesObject.getCurrentTimestamp());
// Following is an algorithm to determine 1st and 2nd maximum utility values.
for (Entry<Integer, Double> utilityEntry : uvPropertiesObject
.getSiteUtilityMap().entrySet()) {
if (maxUtility < utilityEntry.getValue()) {
// Just swap values
selectedSiteB = selectedSiteA;
secondMaxUtility = maxUtility;
```

```
selectedSiteA = utilityEntry.getKey();
maxUtility = utilityEntry.getValue();
} else if (secondMaxUtility < utilityEntry.getValue()) {
selectedSiteB = utilityEntry.getKey();
secondMaxUtility = utilityEntry.getValue();
}
}

// set first maximum and second maximum sites
uvPropertiesObject.getMaxUtilitySite().setKey(selectedSiteA);
uvPropertiesObject.getMaxUtilitySite().setValue(maxUtility);

uvPropertiesObject.getSecondMaxUtilitySite().setKey(selectedSiteB);
uvPropertiesObject.getSecondMaxUtilitySite().setValue(secondMaxUtility);

System.out.println("The sites with active alerts are site: "
+ selectedSiteA + " with utility value: " + maxUtility
+ " and site: " + selectedSiteB + " with utility value: " + secondMaxUtility);

// We have used dynamic code to check for which UAV here
whatUAV();

return uvPropertiesObject;
}
```

A.3 Calculating Next Location for UAVs

```
/**
 * @Description determine which UAV goes to which of the two active alert
 *              sites during the current time stamp. Also, update the
 *              properties of all UAVs.
 * @Notes (i): For each UAV, out of all active alert sites, the one which is
 *        closer to a particular UAV's initial position/Site is chosen for
 *        loitering/servicing
 * @Notes (ii): (i) ensures no UAV crosses-over another UAV. The property is
```

```
 *        somewhat similar to suspending an "elastic" object from a fixed
 *        point.
 * @Notes (iii): The order of choices {UAV1 chooses new site before UAV2 or
 *        vice-versa} does not make a difference to the final results.
 */
private void whatUAV() {
/*
 * Distance in terms of difference between site IDs of UAV1's current
 * site and one of the site, say Site A, which has an active alert.
 */
Integer uav1ToAlertActiveSite1 = -1;
/*
 * Distance in terms of difference between site IDs of UAV1's current
 * site and other of the site, say Site B, which has an active alert.
 */
Integer uav1ToAlertActiveSite2 = -1;

/*
 * Find the site of alert, one which would be closer to the initial
 * location of UAV1. Similar logic can be implemented for UAV2 with no
 * difference in result.
 */
UAVClass uav1 = this.localUAVVector.get(0);
UAVClass uav2 = this.localUAVVector.get(1);

uav1ToAlertActiveSite1 = Math.abs(uav1.getUavInitialSiteID()
- uvPropertiesObject.getMaxUtilitySite().getKey());
uav1ToAlertActiveSite2 = Math.abs(uav1.getUavInitialSiteID()
- uvPropertiesObject.getSecondMaxUtilitySite().getKey());

// Set previous sites before
uav1.setUavPreviousSiteID(uav1.getUavCurrentSiteID());
uav2.setUavPreviousSiteID(uav2.getUavCurrentSiteID());
```

```
// find which of the active alert site is closer to uav1
if (uav1ToAlertActiveSite1 < uav1ToAlertActiveSite2) {
uav1.setUavCurrentSiteID(uvPropertiesObject.getMaxUtilitySite()
.getKey());

uav2.setUavCurrentSiteID(uvPropertiesObject
.getSecondMaxUtilitySite().getKey());

} else {
uav1.setUavCurrentSiteID(uvPropertiesObject
.getSecondMaxUtilitySite().getKey());

uav2.setUavCurrentSiteID(uvPropertiesObject.getMaxUtilitySite()
.getKey());
}

System.out
.println("**********************************************************************");

// Graphics code
// How many stations does UAV1 transcend in this Time-stamp
int uav1DeltaIndex = uav1.getUavCurrentSiteID() - uav1.getUavPreviousSiteID();

// How many stations does UAV2 transcend in this Time-stamp
int uav2DeltaIndex = uav2.getUavCurrentSiteID() - uav2.getUavPreviousSiteID();

DrawingUtils drawing = DrawingUtils.getDrawingUtils();
drawing.setCurrentTimestamp(uvPropertiesObject.getCurrentTimestamp());

/*
 * In graphics, the sites are indexed starting from 0 rather than 1
 * (used in mainPackage). Hence, the subtraction by 1.
 */
drawing.markActiveSites(uav1.getUavCurrentSiteID() - 1,
```

```
uav2.getUavCurrentSiteID() - 1);
drawing.moveUAVs(uav1.getUavCurrentSiteID() - 1, uav1DeltaIndex,
uav2.getUavCurrentSiteID() - 1, uav2DeltaIndex);
// EO Graphics code ///////////

System.out.println(uav1.toString() + " goes to Site: "
+ uav1.getUavCurrentSiteID() + " for monitoring!!!");

System.out.print("Estimated GAIN of moving to Site: " + uav1.getUavCurrentSiteID()
+ " for time stamp: "+ uvPropertiesObject.getCurrentTimestamp() + " is: "
+ uvPropertiesObject.getSiteGainMap().get(uav1.getUavCurrentSiteID()));

System.out.println(", Estimated COST of moving to Site: " + uav1.getUavCurrentSiteID()
+ " for time stamp: " + uvPropertiesObject.getCurrentTimestamp() + " is: "
+ uvPropertiesObject.getSiteCostMap().get(uav1.getUavCurrentSiteID()));

System.out.println(uav2.toString() + " goes to Site: "
+ uav2.getUavCurrentSiteID() + " for monitoring!!!");

System.out.print("Estimated GAIN of moving to Site: "
+ uav2.getUavCurrentSiteID() + " for time stamp: "
+ uvPropertiesObject.getCurrentTimestamp() + " is: "
+ uvPropertiesObject.getSiteGainMap().get(uav2.getUavCurrentSiteID()));

System.out.println(", Estimated COST of moving to Site: "
+ uav2.getUavCurrentSiteID() + " for time stamp: "
+ uvPropertiesObject.getCurrentTimestamp() + " is: "
+ uvPropertiesObject.getSiteCostMap().get(uav2.getUavCurrentSiteID()));

}
```

APPENDIX B

CODE SNIPPET INVOLVING GRAPHICS DISPLAY

```
/**
* @Description initialize the panel with border lines and UAVs
*/
public void initializeDrawingPanel(int noOfSites) {
this.totalNoOfSites = noOfSites;

width = getWidth(); // width (length for our purpose) of the JFrame
height = getHeight(); // height of the JFrame

// left-most points of the borderLine/border
startX = (int) (width * 0.1);
startY = (int) (height * 0.4);

// border length is 80% of JFrame length
borderLength = (int) (width * 0.8);

// spacing between two stations
spacing = (int) ((borderLength - AlertStationSprite.WIDTH * noOfSites)
/ (noOfSites - 1));

// so that the ending of last site is the ending of the border too
borderLength = (int) ((spacing + AlertStationSprite.WIDTH)
* noOfSites - spacing);

borderLine = new BorderLineSprite();
alertStations = new AlertStationSprite[noOfSites];
```

```
legends = new LegendPanel();
legends.setPosition(startX, startY + 40);

timestampPanel = new TimestampPanel();
timestampPanel.setPosition(startX + 400, startY - 80);
timestampPanel.setTimestampValue(this.currentTimestamp);

// Initialize the Alert stations
for (int i = 0; i < noOfSites; i++) {
alertStations[i] = new AlertStationSprite(i + 1);
}

UAVOffset = (int) (AlertStationSprite.WIDTH / 2 - UAVSprite.WIDTH / 2);
// Starting x-coordinate position for UAV1
UAVStartX[0] = startX + UAVOffset;
// Starting x-coordinate position for UAV2
UAVStartX[1] = startX + borderLength - UAVOffset - UAVSprite.WIDTH;
// Starting y-coordinates position for both UAVs is same
UAVStartY = startY - AlertStationSprite.HEIGHT - UAVSprite.HEIGHT;

// Initialize the UAV elements
for (int i = 0; i < uavs.length; i++) {
uavs[i] = new UAVSprite();
// UAV1
if (i % 2 == 0) {
uavs[i].moveTo(UAVStartX[0], UAVStartY);
uavs[i].setColor(Color.GREEN);
}
// UAV2
else {
uavs[i].moveTo(UAVStartX[1], UAVStartY);
uavs[i].setColor(Color.GREEN);
}
```

```
}

try {
Thread.sleep(3000);
} catch (InterruptedException e) {
e.printStackTrace();
}
}

/**
 * @param site1, site2
 * @Description: mark the active sites for this timestamp
 */
public void markActiveSites(int site1, int site2) {
this.site1 = site1;
this.site2 = site2;
}

@Override
public void paintComponent(Graphics g) {
super.paintComponent(g);
drawGraphics(g);
}

/**
 * @param g
 * @Description delegates drawing of sites, border and UAVs to other method
 */
private void drawGraphics(Graphics g) {
Graphics2D g2 = (Graphics2D) g;
g2 = setupRendering(g2);

if (totalNoOfSites > 0) {
drawBorderAndSitesAndUAVs(g2);
```

```
}
}

/**
 * @param g2
 * @Description Draw the border line, alert sites, legends and UAVs on the
 *              Graphics panel
 */
private void drawBorderAndSitesAndUAVs(Graphics2D g2) {
// Border Line
borderLine
.paint(startX, startY, borderLength, (int) (height * 0.1), g2);
legends.paint(g2);

// Time stamp Panel
timestampPanel.setTimestampValue(this.currentTimestamp);
timestampPanel.paint(g2);

// Alert stations. Set the color of Alert stations. RED=ACTIVE, BLUE=PASSIVE
Color color;
for (int i = 0; i < totalNoOfSites; i++) {
if (i == site1 || i == site2) {
color = Color.RED;
} else {
color = Color.BLUE;
}
alertStations[i].paint(startX
+ (spacing + AlertStationSprite.WIDTH) * i, startY
- AlertStationSprite.HEIGHT, color, g2);
}

// Draw the UAVs
for (int i = 0; i < uavs.length; i++) {
uavs[i].paint(g2);
```

```
}
}

/**
 * @param uav1Index: new position/site of uav1
 * @param uav2Index: new position/site of uav2
 * @param uav1DeltaIndex: by how much does uav1 move
 * @param uav2DeltaIndex: by how much does uav2 move
 */
public void moveUAVs(int uav1Index, int uav1DeltaIndex,
int uav2Index, int uav2DeltaIndex) {
// distance to travel for each "graphical" step
final float distancePerStep = (float) (spacing + AlertStationSprite.WIDTH)
/ (float) STEPS_PER_INDEX;

// time in milliseconds to take for each "graphical" step
int timePerStep = TIME_PER_INDEX / STEPS_PER_INDEX;
// the number of "graphical" steps for uav1
final int uav1Steps = STEPS_PER_INDEX * uav1DeltaIndex;
// the number of "graphical" steps for uav2
final int uav2Steps = STEPS_PER_INDEX * uav2DeltaIndex;

timer = new Timer();

UAVTimerTask uavTimer = new UAVTimerTask();
uavTimer.distancePerStep = distancePerStep;
uavTimer.uav1Steps = Math.abs(uav1Steps);
uavTimer.uav2Steps = Math.abs(uav2Steps);

// if UAV1 changes location/position
if (0 != uav1Steps)
uavTimer.uav1Direction = uav1Steps / Math.abs(uav1Steps);
else
uavTimer.uav1Direction = 0;
```

```
// if UAV2 changes location/position
if (0 != uav2Steps)
uavTimer.uav2Direction = uav2Steps / Math.abs(uav2Steps);
else
uavTimer.uav2Direction = 0;

timer.scheduleAtFixedRate(uavTimer, 0, timePerStep);
}

/**
 * @param x
 * @Description Move UAV1 to given coordinates
 */
private void moveUAV1To(int x) {
uavs[0].moveTo(x, UAVStartY);
}

/**
 * @param x
 * @Description Move UAV2 to given coordinates
 */
private void moveUAV2To(int x) {
uavs[1].moveTo(x, UAVStartY);
}

/**
 * @Description Update the coordinates of uav1 and uav2 and paint new
 *              positions of the Graphics Panel
 */
private class UAVTimerTask extends TimerTask {
int count = 0;
int uav1Direction, uav2Direction;
float distancePerStep;
```

```
int uav1Steps, uav2Steps;

@Override
public void run() {
int x1, x2;
if (count < uav1Steps) {
x1 = (int) (distancePerStep * count * uav1Direction);
} else {
x1 = (int) (distancePerStep * uav1Steps * uav1Direction);
}

if (count < uav2Steps) {
x2 = (int) (distancePerStep * count * uav2Direction);
} else {
x2 = (int) (distancePerStep * uav2Steps * uav2Direction);
}

// move/update the coordinates of uav1 and uav2 to new positions
moveUAV1To(UAVStartX[0] + x1);
moveUAV2To(UAVStartX[1] + x2);

// The number of required steps for both uav1 and uav2 is done
if (count > uav1Steps && count > uav2Steps) {
UAVStartX[0] += x1;
UAVStartX[1] += x2;

timer.cancel();
}

// while the timer is running, repaint the Graphics panel
repaint();
count++;
}
}
```

BIBLIOGRAPHY

[1] K. J. Ordonez, *Modeling the U.S. Border Patrol Tucson Sector for the Deployment and Operations of Border Security Forces.* PhD thesis, Monterey, California. Naval Postgraduate School, 2006.

[2] R. W. Beard, T. W. McLain, D. B. Nelson, D. Kingston, and D. Johanson, "Decentralized Cooperative Aerial Surveillance Using Fixed-Wing Miniature UAVs," *Proceedings of the IEEE*, vol. 94, no. 7, pp. 1306–1324, 2006.

[3] C. Bolkcom, "Homeland Security: Unmanned Aerial Vehicles and Border Surveillance," DTIC Document, 2004.

[4] K. Kalyanam, P. Chandler, M. Pachter, and S. Darbha, "Optimization of Perimeter Patrol Operations Using Unmanned Aerial Vehicles," *Journal of Guidance, Control, and Dynamics*, vol. 35, no. 2, pp. 434–441, 2012.

[5] K. Krishnamoorthy, M. Pachter, P. Chandler, D. Casbeer, and S. Darbha, "UAV Perimeter Patrol Operations Optimization using Efficient Dynamic Programming," in *American Control Conference (ACC), 2011*, pp. 462–467, IEEE, 2011.

[6] K. Krishnamoorthy, M. Park, S. Darbha, M. Pachter, and P. Chandler, "Approximate Dynamic Programming Applied to UAV Perimeter Patrol," in *Recent Advances in Research on Unmanned Aerial Vehicles*, pp. 119–146, Springer, 2013.

[7] A. R. Girard, A. S. Howell, and J. K. Hedrick, "Border Patrol and Surveillance Missions using Multiple Unmanned Air Vehicles," in *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, vol. 1, pp. 620–625, IEEE, 2004.

[8] D. Bertsimas, G. Van Ryzin, *et al.*, "The Dynamic Traveling Repairman Problem," 1989.

[9] J. Blazakie, "Border Security and Unmanned Aerial Vehicles," DTIC Document, 2004.

[10] F. Eisenführ, M. Weber, and T. Langer, *Rational Decision Making.* Springer, 2010.

[11] P. Lunenburg, "The Decision Making Process," in *National Forum of Educational Administration and Supervision Journal*, vol. 27, pp. 1–12, 2010.

[12] M. Towler and S. Keast, *Rational Decision Making for Managers: An Introduction.* Wiley, 2009.

[13] S. Eriksen and L. R. Keller, "Decision Trees,"

[14] B. De Ville and P. Neville, *Decision Trees for Analytics Using SAS Enterprise Miner.* SAS Institute, 2013.

[15] R. D. Shachter, "Probabilistic Inference and Influence Diagrams," *Operations Research*, vol. 36, no. 4, pp. 589–604, 1988.

[16] A. C. Miller III, M. W. Merkhofer, R. A. Howard, J. E. Matheson, and T. R. Rice, "Development of Automated Aids for Decision Analysis," tech. rep., DTIC Document, 1976.

[17] A. Detwarasiti and R. D. Shachter, "Influence Diagrams for Team Decision Analysis," *Decision Analysis*, vol. 2, no. 4, pp. 207–228, 2005.

[18] J. E. Smith, S. Holtzman, and J. E. Matheson, "Structuring Conditional Relationships in Influence Diagrams," *Operations research*, vol. 41, no. 2, pp. 280–297, 1993.

[19] F. Er and S. Lezki, "The Usage of Influence Diagram for Decision Making in Textiles," *Asian Social Science*, vol. 8, no. 11, p. p163, 2012.

[20] S. Tamimi, J. Dickman, and R. Bauman, "Risk analysis Using Influence Diagrams," in *Uncertainty Modeling and Analysis, 1990. Proceedings., First International Symposium on*, pp. 348–353, IEEE, 1990.

[21] J. Levin, "Choice under Uncertainty," 2006.

[22] E. Horvitz, "Uncertainty, Utility, and Understanding," in *Intelligent Tutoring Systems*, pp. 18–18, Springer, 2000.

VITA

Graduate College
University of Nevada, Las Vegas

Ashish Karki

Degrees:

Bachelor's Degree in Computer Engineering, Pulchowk Campus, Institute of Engineering, Tribhuvan University, Nepal, 2010

Thesis Title: Implementing Influence Diagram Concepts to Optimize Border Patrol Operations using Unmanned Aerial Vehicles

Thesis Examination Committee:

Chairperson, Dr. Wolfgang Bein, Ph.D.
Committee Member, Dr. Laxmi P. Gewali, Ph.D.
Committee Member, Dr. Kazem Taghva, Ph.D.
Graduate Faculty Representative, Dr. Emma Regentova, Ph.D.