NAVAL
POSTGRADUATE
SCHOOL

MONTEREY, CALIFORNIA

THESIS

AUTONOMOUS UNDERWATER VEHICLE PLANNING
FOR INFORMATION EXPLOITATION

by

Adam Wiseman

March 2012

Thesis Co-Advisors:	Douglas Horner
	Oleg Yakimenko

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
4–4–2012	Master's Thesis	2010-04-01—2012-03-30

4. TITLE AND SUBTITLE

Autonomous Underwater Vehicle Planning for Information Exploitation

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Adam Wiseman

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Postgraduate School
Monterey, CA 93943

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Department of the Navy

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited

13. SUPPLEMENTARY NOTES

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. **IRB Protocol Number: N/A**

14. ABSTRACT

The ability of an Autonomous Underwater Vehicle (AUV) to dynamically plan safe routes and maneuvers in dangerous environments is directly relevant for the future of the use of AUVs in the exploration and exploitation of the underwater environment, specifically the littorals and inland waters. This thesis builds upon the existing body of knowledge of the REMUS AUV dynamics and kinematics and develops a control scheme for a real-time optimized vehicle trajectory that will permit continuous and autonomous collection and exploitation of external sensor data, which will facilitate full 360-degree, 2-dimensional mapping of the underwater environment surrounding the vehicle while preventing the vehicle from coming into contact with mapped objects in the water. The developed control schema will seek to generate a trajectory in real-time that optimizes a key parameter of interest, the Information Gain, while minimizing a specified cost function of constraints, such as kinematic limits and obstacle avoidance criteria.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
Unclassified	Unclassified	Unclassified	UU	117	19b. TELEPHONE NUMBER *(include area code)*

THIS PAGE INTENTIONALLY LEFT BLANK

AUTONOMOUS UNDERWATER VEHICLE PLANNING FOR INFORMATION EXPLOITATION

Adam Wiseman
Lieutenant, United States Navy
B.S., Engineering Management, University of Arizona, 2005

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
March 2012

Author: Adam Wiseman

Approved by: Douglas Horner
                           Thesis Co-Advisor

                           Oleg Yakimenko
                           Thesis Co-Advisor

                           Knox Millsaps
                           Chair, Department of Mechanical and Aerospace Engineering

iii

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The ability of an Autonomous Underwater Vehicle (AUV) to dynamically plan safe routes and maneuvers in dangerous environments is directly relevant for the future of the use of AUVs in the exploration and exploitation of the underwater environment, specifically the littorals and inland waters. This thesis builds upon the existing body of knowledge of the REMUS AUV dynamics and kinematics and develops a control scheme for a real-time optimized vehicle trajectory that will permit continuous and autonomous collection and exploitation of external sensor data, which will facilitate full 360-degree, 2-dimensional mapping of the underwater environment surrounding the vehicle while preventing the vehicle from coming into contact with mapped objects in the water. The developed control schema will seek to generate a trajectory in real-time that optimizes a key parameter of interest, the Information Gain, while minimizing a specified cost function of constraints, such as kinematic limits and obstacle avoidance criteria.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

List of Figures

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

Acknowledgements

First and foremost, I thank God for getting me through this with my sanity mostly intact.

Words cannot express my gratitude towards my wife and my rock, Selena. Her patient tolerance for my stress-induced moods, unflagging support of me in my many frustrations, and exceptional ability to take care of the home front during the many, many hours I spent at school toiling away on this thesis were an absolutely essential ingredient to my success in this endeavor. I simply cannot fathom how I could have done this without her.

I would like to thank Prof. Doug Horner for his patience with my shortcomings, his understanding during my misunderstandings, and his willingness to push me forward, out of my comfort zone, into uncharted territory despite my consternation and apprehension.

Finally, I thank Prof. Oleg Yakimenko for his abundant help. His willingness to spend many hours helping me through the toughest times, even on the weekends, debugging routines, checking code, and fixing my mistakes were critical to my success.

So, from the bottom of my heart, thank you Prof. Yakimenko, thank you Prof. Horner, and most of all, THANK YOU SELENA.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
Introduction

The ocean is a vast and ever-changing environment. It presents significant challenges for those that seek to explore and exploit it. Much of it is unknown and unexplored, and even much of the known areas can be inhospitable and downright hostile to human intrusion. It has been said that we know more about outer space than our own oceans [1]; yet it remains one of mankind's most vital and heavily exploited environments [2]. Ships have plied the seas for exploration, merchant trade, and military operations for millennia, but until the last century, what lie below the surface of the sea remained in the realm of myth and fantasy. While merchant mariners remain largely content to safely ply the navigable waters on the surface with little concern as to what lies below, the undersea environment is of great interest to scientists, engineers, and military planners and operators. To that end, great strides have been made in exploring this undersea world, from the deepest abysses to the shallow littorals and inland waterways connected therewith. As evidenced by recent funding, research and development priorities, the Navy has significant interest in exploring these shallow water environments [3].

1.1 Motivation

Dominance of the shallow waters of the littorals and inland waters is a vital concern for Maritime Component Commanders. Amphibious operations, Naval Surface Gunfire Support missions, Special Operations, submarine strike packages, near-shore and inshore reconnaissance, mine countermeasures, riverine combat, and waterway defense are just a few of the Naval operations that exploit this waterspace. To enable such dominance, the Commander must have the tools at his disposal to safely and completely explore unknown areas, investigate and map underwater features, and locate and counter submerged mines. Remotely Operated Vehicles (ROV's) and Autonomous Underwater Vehicles (AUVs) are being used extensively in fields of underwater cartography, exploration, salvage, and oceanographic research [4]. In recent years, the Navy has recognized their extreme utility in meeting the aforementioned needs in the context of military planning and operations. Just as Unmanned Aerial Vehicles have seen widespread use in similar roles by Air and Ground Commanders, AUV systems are rapidly becoming the go-to tool for Maritime Commanders to meet their unique needs in the littorals. Some of the AUV's attributes include:

Human Safety –Since AUVs are unmanned, the risk to human operators is reduced. Additionally, they can be used in roles traditionally performed by human divers, thus further eliminating the risk to the Navy's valuable human capital.

Flexible –Modern AUVs are designed and constructed with modularity in mind, allowing them to be outfitted with sensors, actuators, and tools to accomplish a wide variety of missions.

Mobile –Many AUV systems (vehicle and associated support equipment) are small enough to fit into the bed of a pickup truck or onto a helicopter, and are man-portable, allowing them to be quickly delivered just about anywhere they are needed. Additionally, their small size allows them to maneuver in spaces or areas inaccessible by manned underwater vehicles.

Autonomous –By definition, AUVs are autonomous, meaning that once set upon it's mission, no human control or intervention is required to accomplish the mission, resulting in lower manpower requirements, almost zero risk of danger to human operators, and much higher cost savings.

Low Cost –Cost savings over traditional means for accomplishing the stated objectives are reached by use of Commercial Off-The-Shelf (COTS) technologies, reduction of manpower requirements, size, and modularity. Further, the minimal amount of support equipment required for many AUVs is far less than that of a comparably capable ROV, further reducing costs.

1.2   Problem Statement

The Navy is interested in making AUVs more autonomous. In general, AUV autonomy is defined by the ability of the vehicle to interpret it's environment and plan trajectories that best adhere to the mission objectives. It is characterized by four distinct aspects:

Internal Representation of the Environment - the spatial mapping, or "Environmental Map", of objects and obstacles in the environment in which the vehicle is operating

Path Planning - creating a path to follow; two types of path planning: deliberative, where the path is planned based on known information stored in the Environmental Map; and reactive, where the vehicle detects objects or obstacles not already existing in the Environmental Map and plans a path to avoid them

Path Following - Path planning results in a trajectory that the vehicle is trying to follow. This provides direction to the controls, which then drives the vehicle along the path.

Sensory processing -As new sensory information is obtained, such as position and other raw sensory data, it is fed back into the loop to localize the vehicle and update the World map.

Reactive obstacle detection is an example of a behavior exhibiting greater autonomy. The external representation of the AUV environment could be a map with the water depths and known features and obstacles. The path planning module uses this information to plan a path. This path must be calculated by considering the limitations in the AUVs maneuverability. Finally, sensors such as a forward looking sonar provide imagery regarding the environment forward of the AUV. Computer vision techniques are used on the images and pertinent information is extracted and entered into the map. This feedback loop continues through the entire deployment cycle and must be accomplished in near real-time. [5]

For this thesis, I am interested persistent AUV autonomy. This is ability of the AUV to remain deployed for longer periods of time. The specific concept of operations is to put the vehicle into a suspended sleep state. All primary systems are shut down to conserve battery life with the exception of a communications or sensing device and the computer to run them. Before the AUV goes into this sleep mode, it surveys the area. During the survey it is looking for objects readily detected by the forward looking sonar.

The AUV is eventually awakened via communications or sensory detection. The first objective is to localize the position of the AUV. Due to limitations in positive ballast and oceanic conditions, the AUV may have drifted along the ocean floor. For tactical and safety reasons, it is preferable not to surface the AUV for a GPS fix. The goal is for the AUV to efficiently search the area for the navigational fixes that were surveyed prior to the sleep mode. This thesis addresses the path planning necessary for such a behavior. The goal is to develop an information theoretic search plan for the AUV to detect the survey sites taking into consideration the uncertainty associated with vehicle position due to drift.

This thesis describes an optimal two-dimensional exploratory path-planning schema for the REMUS AUV[1] that utilizes well known and understood concepts of probabilistic analysis (Bayesian inference) and information theoretics (entropic information, divergence measures, etc.) for analyzing and quantifying the information gain achieved by an onboard organic sensor. It is this latter subject of information analysis that is of key importance to this thesis. Rigorous analysis will be given to determining the information gain achieved by the autonomous system in the course of following various candidate trajectories. Information gain, in the context of the

---

[1]Although the methodologies explored by this thesis have been developed for use on the REMUS AUV, they are obviously not limited to use on this vehicle, and may certainly be more generally applied, with minimal effort or modification, to use in path planning for practically any autonomous agent operating under similar conditions and constraints.

present problem being addressed, represents a quantitative and qualitative measure of the situational awareness and utility of the data models being collected and developed within the system. This information gain will be utilized as the primary optimized parameter in the exploratory path planning and optimization routine. For this, the author will employ a contemporary approach to trajectory generation and optimization known as Direct Method of the Inverse Dynamics in the Virtual Domain (hereafter referred to as DM-IDVD), that not only ensures complete exploratory coverage of the unknown underwater space in which the vehicle operates, but allows real-time, near-optimal trajectory generation within the performance constraints of off the shelf computing and control systems widely used on autonomous vehicles.

1.3   Literature Review

Path planning is one of the most rigorously studied problems in the field of robotics. Numerous techniques, methodologies, and algorithms exist to process sensor information and plan a trajectory to direct an agent (e.g. vehicle, robotic device, etc.) from an initial state to a final state subject to dynamic, kinematic, environmental, configurational, or other constraints. The algorithms and guidance laws produced by such methodologies address such issues as localization, track following, cross-track error control, obstacle avoidance, and trajectory optimization. Of key importance to the efficacy of almost all of these methodologies, as applied to an autonomous vehicle, is the means by which the information gathered by the agent's onboard sensor(s) is collected, measured, quantified, analyzed, and applied.

1.3.1   Trajectory Generation

In the context of the REMUS AUV, many path planning methods have been explored. Most earlier works focused on simplified path planning problems such as contour following and obstacle avoidance. Van Reet explored contour following techniques utilizing logic-based gradient methods [6]. Heminger [7] and Healey [8] developed reactive obstacle avoidance techniques which used Gaussian Potential Functions for obstacle avoidance. Furukawa worked on an obstacle avoidance routine that combined a spline addition planner with a look-ahead pitch controller [9]. While sufficient for simplified mission tasks, these approaches have intrinsic limitations and drawbacks when applied to more complex scenarios, including sub-optimality, computational intensity, ambiguity in situational awareness, and sensitivity to error.

The Direct Methods approach seeks to mitigate these issues, and of late, much attention here at NPS has been given to applying this technique to autonomous system navigation. Recent work includes that of Yakimenko, Horner, Pratt, and Kragelund. Horner and Yakimenko were the

4

first to incorporate Direct Methods into the obstacle avoidance path planning framework. Their 2007 paper [5] utilized sensor data in a closed feedback loop and a known environmental map to provide the situational awareness to the DM-IDVD algorithm. The algorithm then generated a near-optimal trajectory for navigation and obstacle avoidance, updating in real time along the vehicle's path. Yakimenko then expanded on this paper in 2008 [10], incorporating the DM-IDVD method into a formulation of a more generalized approach to real-time, near-optimal, obstacle avoidance, 3D spatial trajectories. Furthering the research into applying the DM-IDVD approach to AUV navigation was the paper by Yakimenko, Horner and Pratt which detailed a control algorithm for autonomously deploying and recovering AUVs [11]. In this paper, the authors apply the DM-IDVD method to calculating trajectory of an AUV out of (deployment) or into (recovery) a submerged docking apparatus, reducing the AUVs dependency on the host surface vessel or swimmers. What each of these approaches had in common is their use of known a priori information of the environment for localization and navigation. The DM-IDVD method was applied, more or less, to the task of obstacle avoidance or waypoint navigation.

1.3.2 Sensor Information Processing

As alluded to previously, the methods and techniques by which sensor data is analyzed and applied is absolutely key to the efficacy of any path planning algorithm. The methodology explored by this thesis utilizes existing techniques for handling a sonar image input and converting that image into a probabilistic model of the environment. Extensive use is made of McChesney's work [12] with manipulating the sonar image collected by the REMUS vehicle's onboard blazed array sonar sensor[2]. In his thesis, McChesney processes a three-dimensional sonar image by first developing noise and signal confidence probability models of the sonar sensor itself, and then applies a probabilistic update process to develop a three dimensional spatial model of the underwater environment. This spatial model is constructed by way of an Occupancy Grid (OG). The Bayesian probabilistic methods used to build and update the OG have been rigorously explored and presented by Elfes [13] and Noykov [14]. The exact method whereby the collected sonar image is used to develop the OG will be discussed in detail in Chapter 2.

In this thesis, information gain (IG) is the key informational parameter under consideration. To that end, it must be clarified that the information stored in the OG is merely a means to an end. As our primary concern is with determining how much information is gained by the

---

[2]While the McChesney thesis deals with feature reconstruction in three dimensions using both horizontal and vertical elements of the BlueView sonar package installed onboard the REMUS vehicle, this thesis will work in two dimensions only, dealing with the horizontal plane.

system for a given candidate trajectory, a means must be utilized with which we can analyze the gain, or divergence, in cumulative information between discrete consecutive updates of the OG. The tools used in the methodology explored by this thesis come from the field of Information Theoretics. Much use is made of the concepts of divergence measures, entropic information, and Fisher information, as presented in the context of autonomous agent path planning in the Levine thesis [15]. Although Levine applies these concepts to a very different approach to path planning (Rapidly-exploring Random Trees), they are equally apropos for the approach used in this thesis.

1.4   Thesis Organization

Chapter 2 of this thesis will discuss the generalized information theoretics and probabilistic methods that will be used to process, update, and quantify the sonar-sensor information. I will first describe the conditional probabilistic process of Bayesian inference. This process will then be applied to the formulation of the Occupancy Grid (OG) and the update process. Next, the critical foundational concepts of information theoretic measures will be discussed. It will cover entropic information (conditional entropy), divergence measures (Kullback-Leibler divergence), information gain, and Fisher information matrices.

Chapter 3 provides a generalized discussion of the Direct Methods approach. It is the path planning methodology used for the thesis. Chapter 4 will tie the concepts and methods discussed in the previous two chapters into a complete path planning routine. First, the methods and techniques by which the simulated sonar image is processed into an environmental spatial model is discussed. Then the process by which the probabilistic measures of entropy and information gain are quantified are elaborated upon. Next, the constraints on the states, controls, and their derivatives will be defined. Optimization parameters and the concomitant cost functions which incorporate the previously discussed information metrics will then be developed. The candidate trajectory reference functions will be presented, followed by a discussion of parameterization of the reference functions in the *virtual domain*. Initial and final conditions on the fixed and variable parameters of the reference functions will then be established. The inverse dynamic equations are then developed. Finally, with all the pieces in place, the optimization routine will be discussed and analyzed. The remaining chapters cover the analysis of the results, the findings therein, and finally, the conclusion including a brief discussion of opportunities for further research relevant to this thesis topic.

CHAPTER 2:
Information Theoretics

As stated in Chapter 1, the ultimate goal of the approach developed by this thesis is to produce an optimal trajectory that maximizes the information gain achieved by that trajectory. Before examining the Direct Methods process in detail and describing the approach developed by the author, a primer on some of the basic information theoretic measures employed in the author's method. In this chapter, a few of these measures will be reviewed. Bayesian inference is a key component of the probabilistic analysis performed during the analysis and quantification of the received sonar image and the construction of the corresponding occupancy grid. Entropic information is an important metric in determining the value of the information received by the system. Divergence measures, such as information gain, enable the quantification of increase in knowledge gained during the process, and as stated before, are key parameters for optimization.

2.1 Probabilistic Measures

2.1.1 Basic Probability Concepts

Before discussing Bayesian inference itself, some basic probabilistic concepts must be elucidated. Let consider a random variable X, and let x denote any specific value that X may be. For example, if we consider a simple coin flip, where X is the random variable that represents the side of the coin that turns up from the flip, then the possible values of x are *heads* or *tails*. The *probability* that X has a value of x is given by

$$P(X = x), \tag{2.1}$$

and is typically shortened to $p_X(x)$, or simply $p(x)$. So for the example of the coin flip, the probability that heads turns up is indicated by $p(X = heads)$, and the probability of tails turning up is $p(X = tails)$. For a "fair" coin, where there is equal probability of achieving either result, $p(X = heads) = p(X = tails) = \frac{1}{2}$. Furthermore, discrete probabilities always sum to unity, thus $\sum_x p(X = x) = 1$ [16].

A *joint distribution* of X and Y is one in which $p(x,y) = p(X = x \text{ and } Y = y)$, which describes the probability of the event where $X = x$ and $Y = y$ simultaneously. If neither random variable

depends on the other for their respective values, we call the random variables *independent*, such that $p(x,y) = p(x)p(y)$.

Conditional probabilities are another form of probabilistic measure that give us information about the likely value of one random variable given known information about another. In other words, a conditional probability is the probability that random variable X takes on a value of x given that we know another random variable Y has a value of y. Such probabilities are usually denoted

$$p(X = x|Y = y),$$ (2.2)

or more commonly

$$p_{X|Y}(x|y) = p(x|y).$$ (2.3)

This is a key fundamental concept, as it will be heavily utilized in the Bayesian inference process to be discussed in the next section. Furthermore, if $p(y) > 0$, then

$$p(x|y) = \frac{p(x,y)}{p(y)},$$ (2.4)

and if X and Y are independent as defined above, we get

$$p(x|y) = \frac{p(x)p(y)}{p(y)} = p(x).$$ (2.5)

This then leads to one last concept relevant to Bayesian inference, the *Law of Total Probability*: If X and Y are independent, and Y is a set of mutually exclusive and exhaustive events (meaning only one event can occur at a time, and all possible events are contained in the set), then for any other event X,

$$p(x) = p(x|y_1)p(y_1) + p(x|y_2)p(y_2) + \cdots + p(x|y_n)p(y_n) = \sum_y p(x|y)p(y)$$ (2.6)

in the discrete case. As we will see in the next section, this law forms the denominator of the Bayesian Inference equation.

2.1.2 Bayesian Inference

Bayesian inference is the core concept of Bayesian decision theory in which information regarding the current state of a random variable is *inferred* from prior knowledge of states of that random variable and present observations. It allows us to iteratively update the specific current,

8

or *posterior*, probability distribution of a random variable based on observational conditional probabilities of certain events, and knowledge of the previous, or *prior* probability distribution. Bayesian inference is defined by *Bayes rule*. Let X be a random variable with possible value, or state, x drawn randomly from a set of possible values, χ. Before any observations are made, the knowledge of x is given entirely by the *prior* probability distribution $p(x)$. We model observations of the system as random variable Y that take on values of y. Given such observations (e.g. sensor data), we may infer the value of x from that of y by way of the *Bayes rule* equation:

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)} = \frac{p(y|x)p(x)}{\sum_{x'} p(y|x')p(x')} \tag{2.7}$$

Here, $p(x|y)$ is called the *posterior probability distribution* and $p(y|x)$ is called the *inverse conditional probability*. This latter component essentially describes the probability of receiving value y given a specific value of x, for example, the likelihood that a specific state x value, represented here by x', causes a specific sensor measurement value of y.

2.2 Entropy and Information Gain

As previously stated, the objective of this thesis is to plan a trajectory that optimizes the information gain achieved by that trajectory. In order to quantify this information gain, we must first describe the measure that it is predicated on, the *information entropy*. In the context of information theoretic measures, the entropy of a random variable X, denoted $\mathbf{H}$, is essentially a measure of the amount of uncertainty associated with the values of X. It is the expectation of the information that each value of the random variable carries, and is defined mathematically by equation (2.8) [17]. If X can take on any value x from set of possible values χ, and $p(x)$ is the probability of X assuming value x, then the entropy of X is given as

$$\mathbf{H}(\mathbf{X}) = \mathbb{E}_X[I(x)] = -\sum_{x \in \chi} p(x) \log p(x). \tag{2.8}$$

where $I(x)$ is the *self information* of x, given by $I(x) = \log p(x)$, which represents the entropy contribution carried by each value of x. [17]

In the special case where random variable X can only take on two possible values, such as the case in the coin flip example, the entropy function simplifies to the *binary entropy function* given by (2.9) [17]:

$$\mathbf{H_b} = -p\log_2 p - (1-p)\log_2(1-p) \tag{2.9}$$
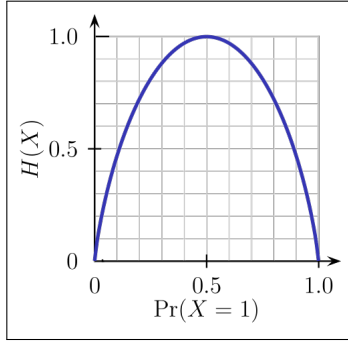
Figure 2.1: Binary Entropy

This function is plotted in Figure 2.1. In this figure, we can see that the maximum entropy occurs where the probability that $X = 1$ ($P(X = 1)$) is 0.5. (Note that since this is a binary function, where X can only equal 1 or 0, then if $P(X = 1) = 0.5$, then $P(X = 0) = 0.5$.) This probability value for maximum binary entropy will be used to initialize the occupancy grid discussed in Section 2.3.

Information Gain, also known as *Kullback–Leibler divergence*, is a measure of the difference between two probability distributions: a "true" probability distribution $p(X)$, and an "assumed" distribution, $q(X)$. If we initially assume that the distribution of values x of random variable X is given by $q(X)$, when the actual correct distribution is $p(X)$, then the difference measure between the two distributions is given by the Kullback–Leibler divergence, defined as

$$D_{KL}(p(x)\|q(x)) = \sum_{x \in X} -p(x)\log q(x) - (-p(x)\log p(x)) = \sum_{x \in X} p(x)\log \frac{p(x)}{q(x)} \qquad (2.10)$$

If $q(x)$ represents our *prior belief* of the probability that $X = x$, and subsequent observations (or measurements) reveal a true, or correct, probability distribution $p(x)$, then the information gain measures how far from the true distribution our initial belief was. Thus by observing and measuring a probabilistic quantity (i.e. random variable) and updating the probability distribution, then those observations carry information about that quantity causing us to increase our knowledge about that quantity, and this gain is quantified as the information gain.

2.3 Occupancy Grids

As robotic sensors are never perfectly accurate, and are thus typically modeled probabilistically, a means for generating a high-confidence spatial map of the sensed environment based on this

Table 2.1: Occupancy Grid Equation Terms

P_{Occ}	$P[s(C) = Occ\|\{r\}_{t+1}]$	Probability that a cell is occupied given the current and all previous measurements
$P1$	$P[s(C) = Occ\|\{r\}_t]$	Probability that a cell is occupied given all previous measurements
$P2$	$P[s(C) = Emp\|\{r\}_t]$	Probability that a cell is empty given all previous measurements
$P3$	$P[r_{t+1}\|s(C) = Occ]$	Probability of receiving the current measurement given that the cell is occupied
$P4$	$P[r_{t+1}\|s(C) = Emp]$	Probability of receiving the current measurement given that the cell is empty

sensor data must be developed based on probabilistic concepts. Elfes developed an iterative method based on Bayesian inference that discretizes the spatial map into a grid of cells. The goal is to determine whether each cell is occupied or empty. The *occupancy probability* of each cell given all previous measurements is then given by

$$P[s(C) = Occ|\{r\}_{t+1}] = \frac{P[r_{t+1}|s(C) = Occ] \cdot P[s(C) = Occ|\{r\}_t]}{\sum\limits_{\forall s(C)} P[r_{t+1}|s(C)] \cdot P[s(C)|\{r\}_t}] \tag{2.11}$$

where

$$r_{t+1}: \quad \text{The current measurement}$$
$$s(C): \quad \text{State of the cell ([Occ]upied or [Emp]ty)}$$
$$\{r\}_t: \quad \text{All measurements up to time t}$$

Expanding the summation in the denominator, and subbing in short variable names from column 1 of Table 2.1, the Bayesian update equation for the occupancy grid cells then becomes

$$P_{Occ} = \frac{P3 \cdot P1}{P4 \cdot P2 + P3 \cdot P1} \tag{2.12}$$

From (2.12), we can see that we initially have 4 unknown variables that we must account for. These unknowns are summarized in Table 2.1 [12]. However, recall that the states may only have two possible states, Occupied or Empty, so $P[s(C) = Occ|\{r\}_t] + P[s(C) = Emp|\{r\}_t] = 1$. So we can substitute $1 - P[s(C) = Occ|\{r\}_t]$ for $P[s(C) = Emp|\{r\}_t]$, thereby reducing the number of variables required to be calculated for each cell to three. The last two components of Table 2.1 are predicated on the probability distributions of the sensor model employed. These
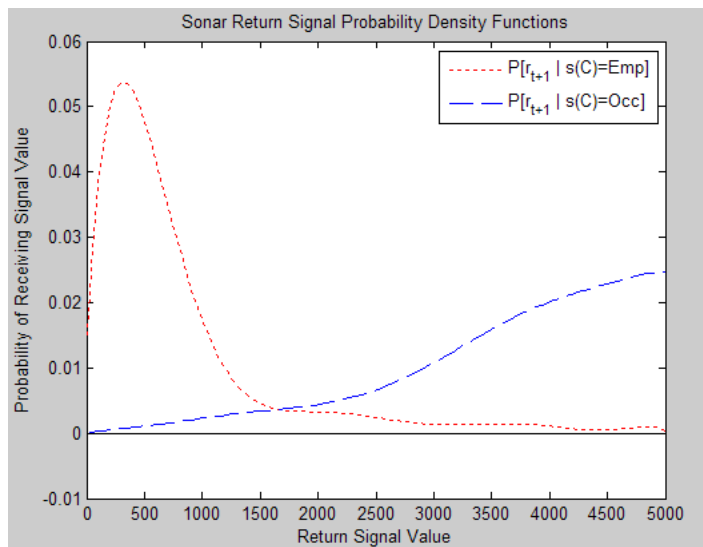
Figure 2.2: Sensor Model Probability Distribution Functions

sensor models probability distributions are typically determined experimentally. An example of the probability distributions for the sonar sensor employed on the NPS REMUS vehicle can be seen in Figure 2.2. The second and third components represent the *prior probability distributions* of occupancy data already existing in the occupancy grid for each cell.

Equation (2.12) represents the iterative process that occurs for each cell during each subsequent collection of sensor data. Thus an initial value is required for the occupancy probability of each cell. Since it is assumed that at time $t = 0$ we know nothing of the occupancy probability, a value that represents this lack of knowledge, or *information* is needed. Recalling from Section 2.2 that this lack of knowledge can be quantified by the *informational entropy*, we initialize all cells of the OG to a value corresponding to maximum entropy. As there are only two possible values that the cells can take, both being exclusive and exhaustive, and since the unconditional probabilities of the cell taking each of these values are equal, then the corresponding maximum entropy is 0.5 (see Section 2.2).

To illustrate the process, consider an example in which a cell begins with an unknown occupancy state. The prior probability that the cell is occupied, $P[s(C) = Occ|\{r\}_t]$, is 0.6, and $P[s(C) = Emp|\{r\}_t] = 1 - P[s(C) = Occ|\{r\}_t] = 0.4$. Now a sonar sweep across this cell receives a return signal, r_{t+1}, with a value of 3000. From the sensor probability density functions in Figure 4.7, the corresponding values for $P[r_{t+1}|s(C) = Occ]$ and $P[r_{t+1}|s(C) = Emp]$ are

approximately 0.01 and 0.001 respectively. Now, substituting these values into (2.12):

$$P[s(C) = Occ|\{r\}_{t+1}] = P_{Occ} = \frac{0.01 \cdot 0.6}{0.001 \cdot (1 - 0.6) + 0.01 \cdot 0.6} = 0.9375$$

we achieve the posterior occupancy probability of the cell conditioned on the value of the sonar signal received, which in this case is equal to 0.9375. Each cell is then updated in the same manner using the prior occupancy probabilities and updated with the sensor model probability values based on new sensor measurements for each sweep.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 3:
Direct Method of Inverse Dynamics in the Virtual Domain

This chapter describes a method, known as the *Direct Method of Inverse Dynamics in the Virtual Domain* (DM-IDVD), whereby a set of candidate spatial trajectories for an autonomous agent is generated in real-time for short-term maneuvers of the agent. The trajectories are generated by way of chosen reference functions that consist of linear functions of the parameters of the system. They are then subjected to an optimization routine that seeks to minimize a given cost function by way of built in or predeveloped minimization computational routines or algorithm, such as *fminsearch* MATLAB function, or the Hookes-Jeeves minimization algorithm. The DM-IDVD process results in the efficient solution of a two-point boundary value problem representing the trajectory that is optimal in both the chosen optimization parameter and the set of constraints based on vehicle kinematics, obstacle avoidance criteria, etc. The efficiency of the routine allows for real-time trajectory optimization within the computational capabilities of most modern autonomous systems. [18]

3.1 Problem Formulation

The DM-IDVD process begins by assuming that there exists a set of admissible trajectories described by the state vector $\mathbf{z}(t)$:

$$\mathbf{z}(t) = [z_1(t), z_2(t), \ldots, z_r(t)]^T \in S$$

$$S = \{z(t) \in Z^r \subset E^r\}, \quad t \in \lfloor t_0, t_f \rfloor$$

All trajectories in the set must satisfy

1. the system of ordinary differential equations (typically defined by the vehicle kinematics):

$$\dot{z}_i = f_i(t, z, \mathbf{u}, \mathbf{c}), \quad i = 1, 2, \ldots, r$$

where $\mathbf{u}$ is the vector of controls given by $\mathbf{u}(t) = \{u_1(t), u_2(t), \ldots, u_l(t)\}^T, l < r, \mathbf{u} \in U^l \subset E^l$, and $\mathbf{c}$ is the vector of vehicle technical characteristics given by $\mathbf{c} = \{c_1, c_2, \ldots, c_p\}^T$, $\mathbf{c} \in C^p \subset E^p$;

2. initial and final conditions on the states and controls[3]:

$$z(t_0) \equiv z_0 \in S_0, \quad S_0\{z_0 \in Z^r \subset E^r\} \tag{3.1}$$

$$\mathbf{u}(t_0) \equiv \mathbf{u}_0 \in R_0, \quad R_0 = \{\mathbf{u}_0 \in U^l \subset E^l\} \tag{3.2}$$

$$z(t_f) \equiv z_f \in S_f, \quad S_f\{z_f \in Z^r \subset E^r\} \tag{3.3}$$

$$\mathbf{u}(t_f) \equiv \mathbf{u}_f \in R_f, \quad R_f = \{\mathbf{u}_f \in U^l \subset E^l\} \tag{3.4}$$

3. constraints on:
   (a) state space:
   $$\eta(t,z) = \{\eta_1(t,z), \eta_2(t,z), \ldots, \eta_w(t,z)\}^T \geq \mathbf{0} \tag{3.5}$$

   (b) controls:
   $$\xi(t,z,\mathbf{u}) = \{\xi_1(t,z,\mathbf{u}), \xi_2(t,z,\mathbf{u}), \ldots, \xi_v(t,z,\mathbf{u})\}^T \geq \mathbf{0} \tag{3.6}$$

   (c) and their derivatives:
   $$\pi(t,z,\mathbf{u},\dot{\mathbf{u}}) = \{\pi_1(t,z,\mathbf{u},\dot{\mathbf{u}}), \pi_2(t,z,\mathbf{u},\dot{\mathbf{u}}), \ldots, \pi_\sigma(t,z,\mathbf{u},\dot{\mathbf{u}})\}^T \geq \mathbf{0} \tag{3.7}$$

Given the set of trajectories satisfying these requirements, the objective then is to determine the best (near-optimal) trajectory $z_{opt}(t)$ from within the set and that trajectory's corresponding controls $\mathbf{u}_{opt}(t)$ that minimize some cost function J. [18]

3.2 Trajectory Generation

The next step in the Direct Methods process is to express the candidate trajectories of the states as a differentially flat function of some abstract parameter τ. (This virtual parameter, typically referred to as the *virtual arc*, will be discussed in more detail in Section 3.3.) Since the main idea of direct methods is to consider the solution as a finite set of variables (functions) [18], we first assume that the admissible functions can be expressed as an infinite power series $z_i(\tau) = \sum_{k=0}^{\infty} a_{ik} \tau^k$, a Fourier series $z_i(\tau) = a_{i0}/2 + \sum_{k=1}^{\infty}(a_{ik} \cos k\tau + b_{ik} \sin k\tau)$, or more generally, by any series function of the form $z_i(\tau) = \sum_{k=1}^{\infty} a_{ik} \varphi_{ik}(\tau)$ where $\varphi_{ik}(\tau)$ is any suitable basis function. In fact, the chosen reference functions may be any combination of basis functions (e.g. monomials, trigonometric functions, etc.) so long as the set of trajectories generated by them meet the requirements in the preceding section. Direct methods then simplify

---

[3]Although this definition declares the terminal parameters as completely defined, in reality, they may not be. In this case, these "free variables" may be added to the set of *optimization parameters* (OP's).

the problem by considering the solutions to be functions of finite series vice infinite. The solution in this case is then merely a function of a set of unknown coefficients. That is to say that $z_i(\tau) = f(a_{i0}, a_{i1}, \ldots, a_{im}, b_{io}, b_{i1}, \ldots, b_{in}, \tau)$, where i is the state parameter index and m and n are the orders of the basis functions.

Consider, for instance the general 3D case where the coordinate state vector for an autonomous agent is given by $\mathbf{x} = [x(\tau), y(\tau), z(\tau)]$. The candidate trajectories of the agent can be expressed as a polynomial of degree N:

$$x_i(\tau) \equiv P_i(\tau) = \sum_{k=0}^{N} a_{ik} \tau^k \tag{3.8}$$

where $x_1(\tau) \equiv x(\tau), x_2(\tau) \equiv y(\tau)$, and $x_3(\tau) \equiv z(\tau)$. The degree N of the polynomial is dependent on the number of boundary conditions that must be satisfied so that all coefficients a_{ik} are determined algebraically instead of being varied. The variable τ is left as a varied parameter. In general, the minimum order n of polynomial required is determined by the orders of the time derivatives of the initial and final coordinates (d_0 and d_f respectively):

$$n = d_0 + d_f + 1 \tag{3.9}$$

Boundary conditions at the initial and final (terminal) points of the desired trajectory that must be satisfied typically include constraints on initial and final position, velocity, and acceleration, i.e. $x_{io}, x_{io}', x_{io}'', x_{if}, x_{if}', x_{if}''$ [19]. In the case that each of these represents a given boundary condition to be satisfied, then the minimum order N of polynomial P in (3.8) is 5, since $d_0 = d_f = 2$. Thus all of the coefficients of P will be uniquely defined by these boundary conditions, leaving τ as the only varied parameter. Of course, if needed, higher order derivatives may be added to meet desired states at initial and/or final points. This simply requires increasing the polynomial degree by the number of additional constraint derivative orders. For example, fixing the final jerk x_{if}''' at zero (as is the case in many terminal trajectory problems) increases d_f by 1, requiring a polynomial of order 6.

Thus far, we have considered the *virtual arc* τ as the only varied parameter. This, however, limits the flexibility of our reference functions, as it provides only one optimization variable for varying our reference trajectories within the set of admissible trajectories. Greater flexibility may be achieved in our reference trajectory by increasing the number of varied parameters to be considered in the optimization routine. This can be accomplished by "fictitious" boundary conditions at the end points. For example, to increase the flexibility in the reference trajectories

17

of the 3D terminal trajectory case discussed above, we may add a 3rd order derivative x_{0f}''', or jerk constraint, to the initial point. Thus d_0 becomes 3, and $N = d_0 + d_f + 1 = 7$. This additional "fictitious" derivative can now be used as a varied parameter, giving us greater control over the shape of the candidate trajectories.

3.3   The Virtual Domain

We now turn our attention to the reason behind parameterizing the reference functions with respect to an arbitrary, or *virtual* parameter τ. Up to this point, we have only concerned ourselves with the flexibility of the trajectory reference function itself. We must now consider why parameterizing the reference functions with respect to time t actually limits our flexibility. Let us consider what would happen if this were indeed the case by analyzing the speed profile of the resulting time-parameterized trajectory. In this case, the speed at any time t along the trajectory would be given by

$$V(t) = \sqrt{u(t)^2 + v(t)^2 + w(t)^2} = \sqrt{\dot{x}^2 + \dot{y}^2 + \dot{z}^2} = \sqrt{\dot{P}_x^2 + \dot{P}_y^2 + \dot{P}_z^2} \qquad (3.10)$$

Thus for any candidate spatial trajectory given by P, we have a single unique and unalterable speed profile along that trajectory. This is undesirable since we might want to be able to vary the speed profile independently of the spatial trajectory. By parameterizing the trajectory with respect to an abstract argument, in this case the virtual arc parameter $\tau (\tau \neq t)$, we introduce a speed factor that allows for just such independence between speed and the spatial trajectory. This speed factor λ is given by

$$\lambda(\tau) = \frac{d\tau}{dt}, \qquad (3.11)$$

which is essentially the *virtual speed*. Now since

$$\dot{x}_i(\tau) = \frac{dx_i(\tau)}{dt} = \frac{dx_i(\tau)}{d\tau}\frac{d\tau}{dt} = x_i'(\tau)\lambda(\tau), \qquad (3.12)$$

we have

$$V(\tau) = \lambda(\tau)\sqrt{x'(\tau)^2 + y'(\tau)^2 + z'(\tau)^2} = \lambda(\tau)\sqrt{P_x'^2 + P_y'^2 + P_z'^2} \qquad (3.13)$$

Thus by varying the speed factor λ, we're able to vary the speed profile independently of the spatial trajectory.

The reason that the capability to vary the speed profile is desirable warrants further discussion. In most cases, the minimum and maximum speeds of a vehicle are constrained by V_{max} and

V_{min}. In the case of the time-dependent speed given by (3.10), the direct correlation between the spatial trajectory and speed along the trajectory means that the speed constraints will also constrain the spatial trajectory, severely limiting our set of candidate trajectories. Computing the trajectories in the τ domain, however, allows us to satisfy these constraints by adjusting $\lambda(\tau)$ without directly affecting the spatial trajectory, such that

$$V_{min} \leq \lambda(\tau)\sqrt{P_x'^2 + P_y'^2 + P_z'^2} \leq V_{max}.$$

Likewise, consider possible constraints on centrifugal acceleration $a^{cf} = kV^2$, where k is the curvature of the trajectory. Here again, solving in the time domain ties the spatial trajectory directly to speed V, and thus a^{cf}. Moving to the τ domain, however, allows us to again satisfy this constraint independently of the spatial trajectory by adjusting $\lambda(\tau)$, since

$$a^{cf}(\tau) = k(\tau)V^2(\tau) = k(\tau)\lambda^2(\tau)(P_x'^2 + P_y'^2 + P_z'^2) \leq a_{max}^{cf}.$$

Yakimenko [20] makes note of some important points concerning this virtual parameterization. First, if the virtual arc length τ_f is on the order of physical path length s_f, then the virtual speed $\lambda(\tau)$ will be correspondingly congruent with the physical speed along the trajectory. Thus if we set λ_0 equal to V_0, we can expect $\tau_f \sim s_f$. This conclusion enables us to make a sound initial guess on τ_f during the optimization portion of the routine. Second, we can see from the relationship between τ and *time* given by equation (3.11) that time can be expressed as

$$t = \int_0^\tau \frac{d\tau}{\lambda(\tau)}.$$

Thus varying the speed factor $\lambda(\tau)$ not only changes the speed profile, but the time scale as well. Finally, we can use the relationship between the virtual domain and time domain in equation (3.11) to determine the derivatives in the time domain of any time-variant parameter [21]. Given parameter ζ, the first and second time-derivatives of this parameter are

$$\dot{\zeta}(\tau) = \frac{d\zeta}{d\tau}\frac{d\tau}{dt} = \zeta'(\tau)\lambda(\tau) \quad and \quad \ddot{\zeta}(\tau) = \lambda(\lambda'\zeta' + \lambda\zeta'') \tag{3.14}$$

We may then invert equation (3.14) to transfer the boundary conditions from the time domain into the τ domain:

$$\zeta' = \frac{\dot{\zeta}}{\lambda} \quad and \quad \zeta'' = \frac{\ddot{\zeta}}{\lambda^2} - \frac{\lambda'\dot{\zeta}}{\lambda} \tag{3.15}$$

19

3.4 Inverse Dynamics

Once the set of candidate trajectories has been computed, the next step in the DM-IDVD routine is to compute the components of the state vector and the controls vector at discrete points along each candidate trajectory. In so doing, not only do we ensure that the constraints of (3.5)–(3.7) are not violated, but we also explicitly return the controls required to follow each candidate trajectory. What we are essentially doing is computing the time-histories of the states and controls along the trajectory. This is accomplished by way of inverting the dynamic equations of motion for the vehicle. Let us again consider the case of a simple 6 Degree of Freedom (6DOF) vehicle operating in 3D cartesian space. Suppose the kinematics of the vehicle are given by

$$
\begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{z}(t) \end{bmatrix} = {}^u_b R \begin{bmatrix} u(t) \\ v(t) \\ w(t) \end{bmatrix}
\tag{3.16}
$$

where ${}^u_b R$ is the rotation matrix from the body frame $\{b\}$ to the NED local tangent plane frame $\{u\}$, and is defined by two Euler angles, pitch ($\theta(t)$) and yaw ($\psi(t)$) (for simplicity, we will neglect roll angle ($\phi(t)$)):

$$
{}^u_b R = \begin{bmatrix} \cos \psi(t) \cos \theta(t) & -\sin \psi(t) & \cos \psi(t) \sin \theta(t) \\ \sin \psi(t) \cos \theta(t) & \cos \psi(t) & \sin \psi(t) \sin \theta(t) \\ -\sin \theta(t) & 0 & \cos \theta(t) \end{bmatrix}
\tag{3.17}
$$

Before applying inverse dynamics, we must transfer (3.16) to the τ domain (assuming a constant surge velocity U_0):

$$
\lambda(\tau) \begin{bmatrix} x'(\tau) \\ y'(\tau) \\ z'(\tau) \end{bmatrix} = {}^u_b R \begin{bmatrix} U_0 \\ v(\tau) \\ w(\tau) \end{bmatrix}
\tag{3.18}
$$

If we also assume that pitch angle is small enough such that $\sin \theta(t) \approx 0$ and $\cos \theta(t) \approx 0$, (3.17) becomes

$$
{}^u_b R(\tau) = \begin{bmatrix} \cos \psi(\tau) & -\sin \psi(\tau) & 0 \\ \sin \psi(\tau) & \cos \psi(\tau) & 0 \\ 0 & 0 & 1 \end{bmatrix}
\tag{3.19}
$$

Note that as all kinematic parameters and equations have now been converted to the τ domain, for simplicity the notation for functional τ-dependency will be hereafter omitted. Inverting the

kinematic equations (3.16) produces the following inverse dynamics equation:

$$
\begin{bmatrix} U_0 \\ v \\ w \end{bmatrix} = \lambda \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}
\tag{3.20}
$$

The remaining inverse dynamics equations are arrived at by solving (3.20) for the three unknown parameters, v, w, and ψ. By inspection, we can readily ascertain that

$$
w = \lambda z',
\tag{3.21}
$$

whereas the solutions for the other two unknown parameters are not as easily achieved. By way of a geometric analysis of the scalar product of the vectors on the right hand side of (3.20), we arrive at the following solutions for v and ψ:

$$
v = \sqrt{\lambda^2(x'^2 + y'^2) - U_0^2}
\tag{3.22}
$$

and

$$
\psi = \Psi - \tan^{-1}\frac{v/\lambda}{U_0/\lambda} = \Psi - \tan^{-1}\frac{v}{U_0}
\tag{3.23}
$$

where

$$
\Psi = \tan^{-1}\frac{y'}{x'}.
\tag{3.24}
$$

With the kinematic equations thusly inverted, we are now able to verify that the constraints on these states are not violated by the candidate trajectories.

3.5 Reference Function Coefficients

States at the boundary conditions are now evaluated so that we may determine the coefficients of the reference functions (3.8). Initial and final conditions on the coordinates (x_{i0} and x_{if}) are given by equations (3.1) and (3.3). Known or given boundary conditions on velocity components of surge, sway, and heave define the first-order τ-derivatives of the coordinate states (x'_{i0}, x'_{if}) by way of (3.18):

$$
\begin{bmatrix} x'_{0;f} \\ y'_{0;f} \\ z'_{0;f} \end{bmatrix} = \frac{^u_b R_{0;f}}{\lambda_{0;f}} \begin{bmatrix} U_0 \\ v_{0;f} \\ w_{0;f} \end{bmatrix}
\tag{3.25}
$$

21

where the initial and final values of yaw required to determine $_b^u R_{0;f}$ above are given by (3.23):

$$\psi_{0;f} = \Psi_0 - \tan^{-1} \frac{v_{0;f}}{U_0} \tag{3.26}$$

and those for pitch are found by:

$$\theta_{0;f} = \gamma_0 + tan^{-1} \frac{-w_{0;f}}{\sqrt{U_0^2 + v_{0;f}^2}}. \tag{3.27}$$

The second order derivatives at the initial point (x_{i0}'') are typically provided by vehicle motion sensors (accelerometers), although they may be explicitly stated as a given initial condition, while the final second order derivatives (x_{if}'') are usually given as defined desired final conditions on acceleration. Likewise, the initial and final higher order derivatives required by the chosen reference functions are are either assigned "guessed" values if they represent the variable parameters (OP's), or are explicitly given as dictated by the constraints on trajectory behavior or constraints on the controls at the initial and terminal points. All of the derivatives must be converted into the τ domain as required by the method describe above. With all necessary states and their τ derivatives have been defined for the initial and terminal points, along with guesses for the variable parameters (including τ_f), the coefficients are then found by solving the appropriate system of equations defined by (3.8) for these coefficients.

3.6 Discretization

In order to numerically calculate the remaining states over the length of the virtual arc from $\tau = 0$ to $\tau = \tau_f$, it is necessary to discretize the trajectory into N evenly spaced points (in the τ domain) with intervals of

$$\Delta\tau = \frac{\tau_f}{N-1}, \tag{3.28}$$

such that

$$\tau_j = \tau_{j-1} + \Delta\tau, \quad j = 2,\dots,N, \ (\tau_1 = 0). \tag{3.29}$$

The Δt for each interval may now be determined:

$$\Delta t_{j-1} = sqrt \frac{(x_j - x_{j-1})^2 + (y_j - y_{j-1})^2 + (z_j - z_{j-1})^2}{U_0^2 + v_{j-1}^2 + w_{j-1}^2}, j = 1,\dots,N \tag{3.30}$$

The speed factor λ for each interval by way of the discrete version of (3.11):

$$\lambda_j = \frac{\Delta\tau}{\Delta t_{j-1}} \qquad (3.31)$$

The value of λ in (3.25) may be assumed to be any reasonable value, such as $\lambda_{0;f} = 1$, since this value merely scales the virtual domain. In other words, the higher the value assigned to λ, the larger the resulting value of τ_f. This result is due to the time-virtual domain relationships given by (3.11) and (3.13). Thus we can see that $\frac{\lambda_{0;f}}{\tau_f} = \frac{U_0}{s_f}$, where s_f is the total physical path length along the trajectory. [10] Once the trajectory has been discretized and λ_j for each point computed by (3.31), we may compute the states and controls at all intermediate point at each time stamp along the trajectory by way of the inverse dynamic equations determined previously ((3.21) through (3.23) in the 6DOF 3D vehicle case above). In some cases, numeric analysis may be required to calculate the time derivatives of certain parameters in order to perform constraint checks on these parameters. Such is the case with Ψ in the 3D example case above, where we must numerically differentiate (3.24) in order to check a given constraint on Ψ such as $|\dot{\Psi}(t)| \leq \dot{\Psi}_{max}$. The net result is a time-history of the states and controls at uniform discrete points along the trajectory.

3.7 Optimization

Obviously, the whole of the entire above process generates the necessary set of trajectories subject to (3.1) to (3.4), we must now turn our attention to satisfying the constraints given by (3.5) through (3.7), along with any optimization parameters required by the situation. As the goal of Direct Methods is to determine a single near-optimal trajectory subject to these constraints and OPs, we must invoke some sort of optimization routine that will optimize a defined performance index and penalty (cost) function. This routine will accept the set of trajectories (more precisely, the state and control time histories), constraints, and OPs as input, and output the trajectory that most closely (near-optimally) satisfies the given requirements.

Let us first consider the cost function. We desire that our solution trajectory meet certain requirements or not violate certain constraints. The cost function is how we quantify such constraint violations or satisfaction of requirements. The cost function determines how much a given constraint or requirement is violated, and calculates an associated penalty value. Let's consider, for example, an example of an Unmanned Underwater Vehicle (UUV), defined by the equations discussed previously in the 6DOF 3D case. Suppose we place the obvious constraints

23

on vehicle depth z (measured in the NED coordinate frame)

$$zmin < z_j < 0, \tag{3.32}$$

pitch θ

$$|\theta(t)| \leq \theta_{max}, \tag{3.33}$$

and yaw rate $\dot{\psi}$

$$|\dot{\psi}(t)| \leq \dot{\psi}_{max}. \tag{3.34}$$

Penalties for each may be expressed as

$$
\begin{aligned}
&\min_{j} \left(0; z_j - z_{\min}\right)^2 \\
&\min_{j} \left(0; -z_j\right)^2 \\
&\max_{j} \left(0; |\theta_j| - \theta_{j\,max}\right)^2 \quad and \\
&\max_{j} \left(0; |\dot{\psi}_j| - \dot{\psi}_{j\,max}\right)^2.
\end{aligned}
\tag{3.35}
$$

The resulting cost function multiplies these by a related scaling (weighting) parameter k:

$$
\Delta = \begin{bmatrix} k^z, k^z, k^\theta, k^{\dot{\psi}} \end{bmatrix}
\begin{bmatrix}
\min_{j} \left(0; z_j - z_{\min}\right)^2 \\
\min_{j} \left(0; -z_j\right)^2 \\
\max_{j} \left(0; |\theta_j| - \theta_{j\max}\right)^2 \\
\max_{j} \left(0; |\dot{\psi}_j| - \dot{\psi}_{j\max}\right)^2
\end{bmatrix}
\tag{3.36}
$$

Other parameters that may be penalized subject to minimums and/or maximums might be speed components ($u_{min} < u_j < u_{max}$, $v_{min} < v_j < v_{max}$, etc.), bank angle ($|\phi_j| < \phi_{max}$), or the controls ($|\delta_j| \leq \delta_{max}$), as well as their derivatives (actuator dynamics) ($|\dot{\delta}_j| \leq \dot{\delta}_{max}$).

The performance index factors in those parameters that are to be minimized or maximized based on the specific requirements of the problem, such as final time t_f, or some other parameter specific to the application, such as in the case with this thesis, Information Gain. In this case, since we will attempt to maximize the IG, the performance index for IG will simply be a weighting factor w^{ig} times the actual calculated value for IG.

Once the overall cost function and performance index is formulated, optimization proceeds by utilizing some optimization algorithm, such as those mentioned at the beginning of this chapter. An example of this, and the method chosen for use in this thesis, is the *fminsearch* algorithm in MATLAB. Optimization is achieved by inputing initial guesses for the variable (optimization) parameters, choosing the required number of iterations, and running the *fminsearch* algorithm on the function that contains the trajectory generation code, the inverse dynamics calculations, and the combined cost function/performance index (hereafter referred to simply as PI). The routine then seeks to minimize the PI by varying values of the "guessed" optimization parameters. The end result is a solution containing the final guesses for the OP's and the time-histories of the states/controls corresponding to the near-optimal (minimized) PI.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 4:
Algorithm Implementation and Testing

This chapter describes the path planning algorithm for the REMUS AUV developed for this thesis. The algorithm implements the previously described concepts of Direct Methods, Bayesian inference, probabilistic analysis, occupancy grids, and information gain. The algorithm takes a known "world map" of the environment as input, as well as user specified constraints and boundary conditions on the vehicle's kinematic and spatial parameters. The net result is a trajectory that satisfies the specified constraints on the vehicle and optimizes the information gain achieved by the vehicle's sensors. Figure 4.1 shows a simplified block diagram of the routine. The orange box is the DM-IDVD path planning and optimization routine. The white box within this box constitutes the optimization loop that is run within the optimization function. The gray bubble corresponding to the gray box in the optimization loop displays the probabilistic analysis subroutine, which performs the sonar imaging, occupancy grid updating, information gain calculation, heuristics violation analysis, and obstacle avoidance subroutine. Each component will be further described in this chapter. The entire routine is run using a set of MATLAB scripts and functions developed by the author specifically for this thesis, as well as a couple of off-the-shelf scripts sourced from the Mathworks website which offer simple functionality for specific tasks (such as generating a Bresenham line in the sonar sweep and obstacle avoidance subroutines). All of the pertinent MATLAB code used in this thesis is presented in Appendix A.

4.1 Problem Setup

The simulation environment chosen for this thesis was the testing tank located in the basement of Halligan Hall on the campus of the Naval Postgraduate School. The tank measures approximately 10 meters by 20 meters by 2 meters deep. The tank is modeled in MATLAB in matrix form for the purposes of generating simulated sonar images and the occupancy grid. All positional coordinates of the vehicle are analyzed with respect to the center of gravity of the vehicle, which, for the sake of simplicity, corresponds to the vertical (z-axis) center of rotation.

As will be discussed in Section 4.2.1, the simulation was run with the vehicle starting with a known and fixed set of initial conditions on position and heading. While the choice for the initial state is completely arbitrary, that for the final condition is not. In theory, the number of candidate endpoints is quasi-infinite, bounded only by the constraints of the test space which,

Path Planning Routine for the Inverse Dynamics in the Virtual Domain
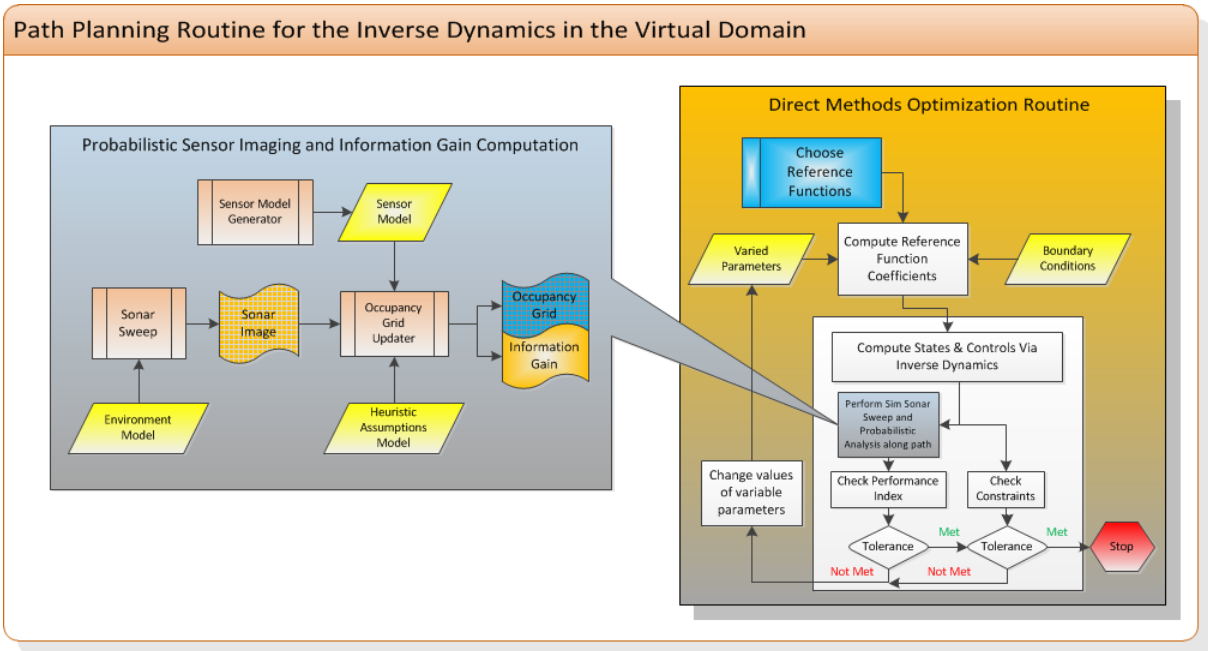
Figure 4.1: Overall Trajectory Generation and Optimization Routine

in the case of this thesis, is the test tank. Although the simulation constrains the trajectories of the vehicle to the confines of the test tank, a means must be addressed by which an endpoint of the set of candidate trajectories evaluated by the direct methods routine may be chosen from the quasi-infinite set of endpoints. Thus another constraint, a time horizon (TH), is incorporated. The TH is a notional period of time over which we will limit the vehicles motion and perform the required analysis. All trajectories of the candidate trajectory sets will be confined to the set of endpoints that the vehicle can possibly reach given constraints on it's surge (u) and sway (v) velocities. Since u_{max} is greater than v_{max} for the REMUS vehicle, the set of possible endpoints is confined to lie within an arc represented by the TH multiplied by the u_{max}. A graphical representation of the random endpoints can be seen in Figure 4.2.

Now, while it is possible to implement the DM-IDVD trajectory optimization routine with free endpoint coordinates given as variable parameters, the computational requirements for such a routine would be exponentially increased, as this also leaves the first order time derivatives of the coordinates at the endpoints free to vary as well. Thus this approach is not feasible for this application, especially given the computational burden already placed on the resources by the subordinate components of the routine added to the standard DM-IDVD algorithm by this thesis (i.e. the sonar imaging, probabilistic analysis, and information theoretics). Thus, when setting
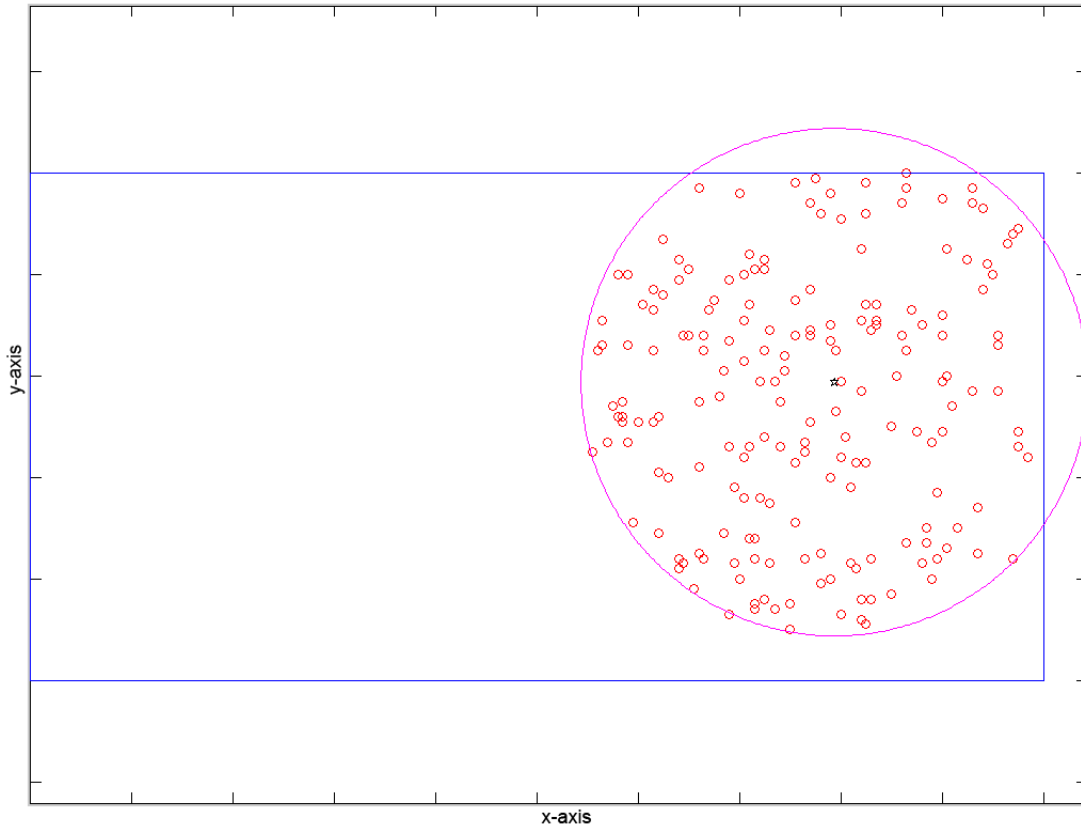
Figure 4.2: Randomly Generated Endpoints (red circles are the endpoints, black star is the start point; pink circle is the time horizon arc, blue rectangle is the test tank)

up the problem, it became necessary to further constrain the endpoint coordinates to a smaller region. In order to accomplish this, an initial analysis of the information space was performed to determine where the endpoints are that achieve the greatest information gain for simple trajectories. This was accomplished by generating a set of random endpoints that lie within the confines defined by the tank walls and the TH arc. A simplified iterative routine was run that traverses the vehicle from the fixed starting point to each of the random end points. Each endpoint trajectory was also iterated over a fixed number for randomly generated final yaw angles (ψ_f), in order to maximize our knowledge of the information space over all variable parameters. A map of the information space for each value of ψ_f was generated by evaluating the information gain achieved along that specific trajectory. A simplified algorithmic representation of this method is shown in Algorithm 1. The resulting map is a 3D representation of IG as a function of the endpoint coordinates for each value of ψ_f. Two dimensionally, an example of this can be seen in Figure 4.3. This figure is a "heat map" where each cell of the figure represents

29

Algorithm 1 Information Space Initial Analysis Algorithm
___
1. For $i = 1$ to number for random ψ_f's
   2. Generate empty IG Array
   3. For $k = 1$ to number of random endpoints
     4. Generate straight-line trajectory between start and endpoint(k)
     5. At each discrete point along trajectory:
       6. Perform sonar sweep subroutine to generate sonar image
       7. Run OG Update subroutine on sonar image
         $\rightarrow$ Update OG
         $\rightarrow$ Calculate IG at this point and add it to previous IG value
       8. Store cumulative IG(k) value in cell corresponding to endpoint(k)
   9. end loop
10. end loop
___

a cartesian coordinate and the value information gain of each coordinate is represented by a color, with cooler colors representing lower values and hotter colors representing higher values. A 3D version of this information gain mapping can be seen in Figure 4.4. By inspection of this IG mapping for each of the values of ψ_f, it became clear that maximal information gain was achieved with trajectories terminating in the Southwest corner of the TH-constrained operating space. Thus the values for the final coordinates are placed in this general location throughout the rest of the algorithm testing. In a real-world application where this a priori data is not achievable, other methods, such as heuristic evaluations, known-space-constrained explorations, etc., may be used to determine the desired end point.

4.2 Direct Methods Problem Formulation

4.2.1 Definition of States, Constraints, and Boundary Conditions

As discussed in Section 3.1, the first step was to define the vehicle state vector, controls, and equations of motion in state space form. As this thesis analyzes only two-dimensional trajectories in the horizontal plane, the state vector **x** was defined as

$$\mathbf{x}(t) = [x(t), y(t), \psi(t), u(t), v(t)]^T, \tag{4.1}$$

and the corresponding ordinary differential equations of motion are

$$\begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \end{bmatrix} = {}^u_b R \begin{bmatrix} u(t) \\ v(t) \end{bmatrix} \tag{4.2}$$

Figure 4.3: Information Gain Heat Map for $\psi_f = -45°$ (NED) Red colors indicate higher IG values, blue colors indicate lower IG values

where ^u_bR is given by

$$^u_bR(t) = \begin{bmatrix} \cos\psi(t) & -\sin\psi(t) \\ \sin\psi(t) & \cos\psi(t) \end{bmatrix} \tag{4.3}$$

The controls vector is given by

$$\mathbf{u} = [n_{prop}, n_{flt}, n_{slt}] \tag{4.4}$$

where n_{prop}, n_{flt}, and n_{slt} are the controls (thrusts) on the main propeller, forward lateral cross-body thruster, and stern lateral cross-body thruster, respectively. Next, the boundary conditions at the initial and final points were defined. The initial pose of the vehicle was arbitrarily defined

31

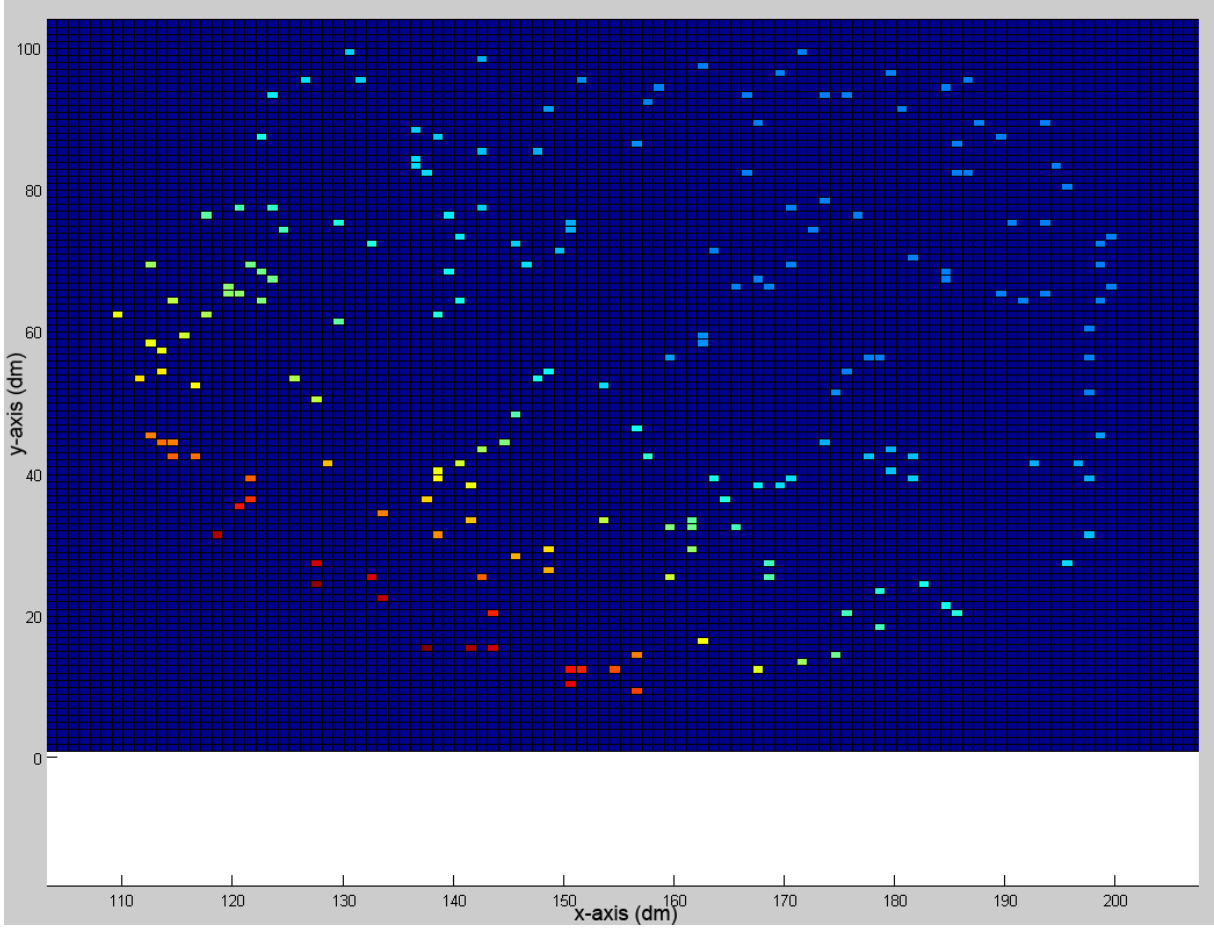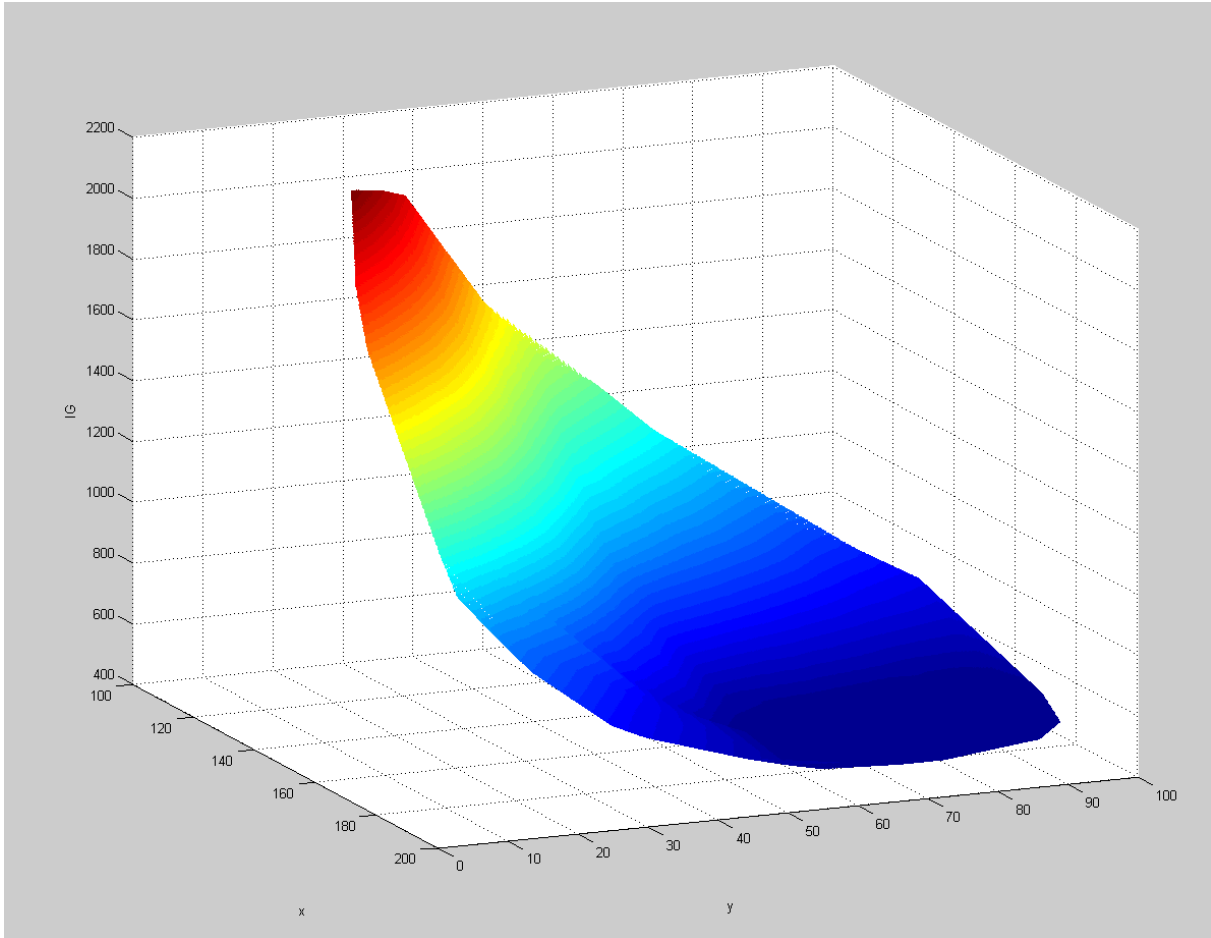Figure 4.4: Three Dimensional Map of Information Gain for $\psi_f = -45°$ (NED) Red colors indicate higher IG values, blue colors indicate lower IG values

with a fixed starting point and heading:

$$x(t=0) \equiv x_0 = x_{init}$$

$$y(t=0) \equiv y_0 = y_{init}$$

$$\psi(t=0) \equiv \psi_0 = psi_{init}$$

32

where x_{init}, y_{init}, and psi_{init} are the values for these parameters specified in the main MATLAB script (see Appendix A). Likewise, the final pose is defined by:

$$x(t = t_f) \equiv x_f = x_{final}$$
$$y(t = t_f 0) \equiv y_f = y_{final}$$
$$\psi(t = t_f) \equiv \psi_f = psi_{final}$$

While the value used for psi_{final} was initially chosen somewhat arbitrarily in the interest of simplifying the problem (and later implemented as a variable parameter), as discussed in Section 4.1, the choice of values for x_{final} and y_{final} is not arbitrary. These values were chosen based on the initial information space analysis described in that section.

Kinematic constraints of the vehicle were sourced from Doherty [22]:

Table 4.1: Kinematic Constraints

Constraint	Value
Max Surge Velocity (v_{max})	2.88 m/s
Max Sway Velocity (u_{max})	0.5 m/s
Max Yaw Rate (ψ_{max})	14.5 deg/sec

4.2.2 Reference Functions

The first step in the DM-IDVD process, represented by the first block in Figure 4.1, is to choose the reference functions for the key spatial parameters of the trajectory. The key parameters of concern in this case were Euclidean spatial coordinates x and y, and yaw angle ψ. Thus reference functions for these parameters were developed. Additionally, a reference function for the *speed factor* λ was also chosen to allow for maximum flexibility in the speed profile along the trajectory. As discussed in Section 3.3, the reference functions are parameterized in the virtual domain as a function of the *virtual parameter* τ. The reference function for the coordinates x_i ($i = \{1,2\}$, *so* $x \equiv x_1$ *and* $y \equiv x_2$) are shown in equation (4.5) below. The chosen reference function for the spatial coordinates is a third order polynomial combined with two trigonometric sine functions. The addition of the sine functions increases the flexibility in varying the curvature of the resulting trajectory. This reference function is then differentiated twice to allow the final acceleration in each coordinate to be varied. (The resulting cosine

33

terms in the second equation of (4.5) arise from the first derivative of the sine terms in the first equation.)

$$
\begin{aligned}
x_i(\bar{\tau}): \quad & P_i(\bar{\tau}) = a_0^i + a_1^i \bar{\tau} + a_2^i \bar{\tau}^2 + a_3^i \bar{\tau}^3 + b_1^i \sin(\pi\bar{\tau}) + b_2^i \sin(2\pi\bar{\tau}) \\
x'_i(\bar{\tau}): \quad & \tau_f P_i'(\bar{\tau}) = a_1^i + 2a_2^i \bar{\tau} + 3a_3^i \bar{\tau}^2 + \pi b_1^i \cos(\pi\bar{\tau}) + 2\pi b_2^i \cos(2\pi\bar{\tau}) \\
x''_i(\bar{\tau}): \quad & \tau_f^2 P_i''(\bar{\tau}) = 2a_2^i + 6a_3^i \bar{\tau} - \pi^2 b_1^i \sin(\pi\bar{\tau}) + 4\pi^2 b_2^i \sin(2\pi\bar{\tau})
\end{aligned}
\tag{4.5}
$$

where

$$
\bar{\tau} = \frac{\tau}{\tau_f}
$$

The parameter $\bar{\tau}$ is used as opposed to τ in order to simplify the process of solving for the coefficients at τ_0 and τ_f in the sine and cosine components of the reference function. The same parameter was used for all of the reference functions for consistency, despite the lack of trigonometric terms in the ψ and λ reference functions. The next steps given by equations (4.6) and (4.7) substitute τ_0 and τ_f for τ into (4.5) to develop the matrix equation (4.8). (Note that although equations (4.6) through (4.8) are those for the x coordinate, those for y are of the same form.)

$$
\begin{aligned}
x(\bar{\tau})|_{\tau=0}: \quad & x_0 = a_0^x \\
x'(\bar{\tau})|_{\tau=0}: \quad & \tau_f x_0' = a_1^x + \pi b_1^x + 2\pi b_2^x \\
x''(\bar{\tau})|_{\tau=0}: \quad & \tau_f^2 x_0'' = 2a_2^x
\end{aligned}
\tag{4.6}
$$

$$
\begin{aligned}
x(\bar{\tau})|_{\tau=\tau_f}: \quad & x_f = a_0^x + a_1^x + a_2^x + a_3^x \\
x'(\bar{\tau})|_{\tau=\tau_f}: \quad & \tau_f x_f' = a_1^x + 2a_2^x + 3a_3^x - \pi b_1^x + 2\pi b_2^x \\
x''(\bar{\tau})|_{\tau=\tau_f}: \quad & \tau_f^2 x_f'' = 2a_2^x + 6a_3^x
\end{aligned}
\tag{4.7}
$$

34

The above equations are then put into matrix form as seen in (4.8):

$$
\Rightarrow
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & \pi & 2\pi \\
0 & 0 & 2 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 0 & 0 \\
0 & 1 & 2 & 3 & -\pi & 2\pi \\
0 & 0 & 2 & 6 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
a_0^x \\
a_1^x \\
a_2^x \\
a_3^x \\
b_1^x \\
b_2^x
\end{bmatrix}
=
\begin{bmatrix}
x_0 \\
x_0' \tau_f \\
x_0'' \tau_f^2 \\
x_f \\
x_f' \tau_f \\
x_f'' \tau_f^2
\end{bmatrix}
\tag{4.8}
$$

The following boundary conditions, as well as the initial guesses for the varied parameters x_f'' and y_f'' are then substituted into (4.8) and the reference function coefficients are solved for (second step of DM-IDVD illustrated by the second block of Figure 4.1).

$$
\begin{aligned}
x_0 &= x_{init} & x_f &= x_{final} \\
x_0' &= 0 & x_f' &= 0 \\
x_0'' &= 0 & x_f'' &= \text{var}
\end{aligned}
\tag{4.9}
$$

$$
\begin{aligned}
y_0 &= y_{init} & y_f &= y_{final} \\
y_0' &= 0 & y_f' &= 0 \\
y_0'' &= 0 & y_f'' &= \text{var}
\end{aligned}
\tag{4.10}
$$

where x_{init}, x_{final}, y_{init}, and y_{final} are the chosen trajectory start point and end point values discussed in Section 4.2.1. The reference function for ψ was chosen to be a simple fifth order polynomial. Since it was desired to vary the initial and final second derivatives of ψ, a fifth order polynomial was necessary due to the requirement given by (3.9). The development process of the reference function including solving for the coefficients is identical to that described above.

Reference Function for ψ:

$$
\begin{aligned}
\psi(\bar{\tau}) &= P_\psi(\bar{\tau}) = a_0^\psi + a_1^\psi \bar{\tau} + a_2^\psi \bar{\tau}^2 + a_3^\psi \bar{\tau}^3 + a_4^\psi \bar{\tau}^4 + a_5^\psi \bar{\tau}^5 \\
\psi'(\bar{\tau}) &= \tau_f P_\psi{}'(\bar{\tau}) = a_1^\psi + 2a_2^\psi \bar{\tau} + 3a_3^\psi \bar{\tau}^2 + 4a_4^\psi \bar{\tau}^3 + 5a_5^\psi \bar{\tau}^4 \\
\psi''(\bar{\tau}) &= \tau_f^2 P_\psi{}''(\bar{\tau}) = 2a_2^\psi + 6a_3^\psi \bar{\tau} + 12a_4^\psi \bar{\tau}^2 + 20a_5^\psi \bar{\tau}^3
\end{aligned}
\tag{4.11}
$$

As before, τ_0 and τ_f are substituted into (4.11) to get the system of equations corresponding to the boundaries:

$$
\begin{aligned}
\psi(\bar{\tau})|_{\tau=0}: & \quad \psi_0 = a_0^\psi \\
\psi'(\bar{\tau})|_{\tau=0}: & \quad \tau_f \psi_0' = a_1^\psi \\
\psi''(\bar{\tau})|_{\tau=0}: & \quad \tau_f^2 \psi_0'' = 2a_2^\psi
\end{aligned}
\tag{4.12}
$$

$$
\begin{aligned}
\psi(\bar{\tau})|_{\tau=\tau_f}: & \quad \psi_f = a_0^\psi + a_1^\psi + a_2^\psi + a_3^\psi + a_4^\psi + a_5^\psi \\
\psi'(\bar{\tau})|_{\tau=\tau_f}: & \quad \tau_f \psi_f' = a_1^\psi + 2a_2^\psi + 3a_3^\psi + 4a_4^\psi + 5a_5^\psi \\
\psi''(\bar{\tau})|_{\tau=\tau_f}: & \quad \tau_f^2 \psi_f'' = 2a_2^\psi + 6a_3^\psi + 12a_4^\psi + 20a_5^\psi
\end{aligned}
\tag{4.13}
$$

$$
\Rightarrow
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 2 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 \\
0 & 1 & 2 & 3 & 4 & 5 \\
0 & 0 & 2 & 6 & 12 & 20
\end{bmatrix}
\begin{bmatrix}
a_0^\psi \\
a_1^\psi \\
a_2^\psi \\
a_3^\psi \\
a_4^\psi \\
a_5^\psi
\end{bmatrix}
=
\begin{bmatrix}
\psi_0 \\
\psi_0' \tau_f \\
\psi_0'' \tau_f^2 \\
\psi_f \\
\psi_f' \tau_f \\
\psi_f'' \tau_f^2
\end{bmatrix}
\tag{4.14}
$$

The boundary conditions on ψ and the first and second derivatives are:

$$
\begin{aligned}
\psi_0 &= psi_{init} & \psi_f &= psi_{final} \\
\psi_0' &= 0 & \psi_f' &= 0 \\
\psi_0'' &= \text{var} & \psi_f'' &= \text{var}
\end{aligned}
\tag{4.15}
$$

where, as before, the values for psi_{init} and psi_{final} were developed in Section 4.2.1, and the initial and final second derivatives are left as varied parameters. As with x and y, these boundary conditions along with the initial guesses on ψ_0'' and ψ_f'' are substituted into (4.14) and the coefficients solved for.

The reference function for the speed factor λ is of the exact same form as that for ψ, and is identically developed, and thus will not be described in detail here. The boundary conditions on λ are also of the same form, with $\lambda_0 = lam_{init}$ and $\lambda_f = lam_{final}$ being set equal to one.

This allows for a direct correspondence between the speed factor and initial and final speeds, as discussed in Section 3.3. The initial and final first derivatives of λ are set to zero, and the initial and final second derivatives are variable parameters.

4.3 Inverse Dynamics

Once the reference functions are fully developed as functions of τ as discussed in the preceding section, they are evaluated over $\tau = [0; \tau_f]$. The result is a fully defined and discretized trajectory in x, y, and ψ, with the necessary τ-domain derivatives being evaluated concomitantly. The next step, illustrated by the third block of Figure 4.1, is to compute the remaining states and controls via inverse dynamics. Values for heading (Ψ), path angle (β), surge velocity (u), sway velocity (v), total speed (V), and time (t), as well as the time derivatives of the angles are calculated at each discrete point k along the trajectory by the following set of inverse dynamic equations:

$$
\begin{aligned}
\Delta\tau &= \frac{\tau_f}{k-1} \\
\tau(1) &= 0 \\
\tau(k>1) &= \tau(k-1)+\Delta\tau \\
dt &= \frac{2\Delta\tau}{\lambda(k-1)+\lambda(k)} \\
t(1) &= 0 \\
t(k>1) &= t(k-1)+dt \\
\Psi(k) &= \tan^{-1}(\frac{x'(k)}{y'(k)}) \\
\beta(k) &= \Psi(k)-\psi(k) \\
V(k) &= \lambda(k)\sqrt{(x'(k))^2+(y'(k))^2} \\
u(k) &= V(k)\sin(\beta(k)) \\
v(k) &= V(k)\cos(\beta(k))
\end{aligned}
\tag{4.16}
$$
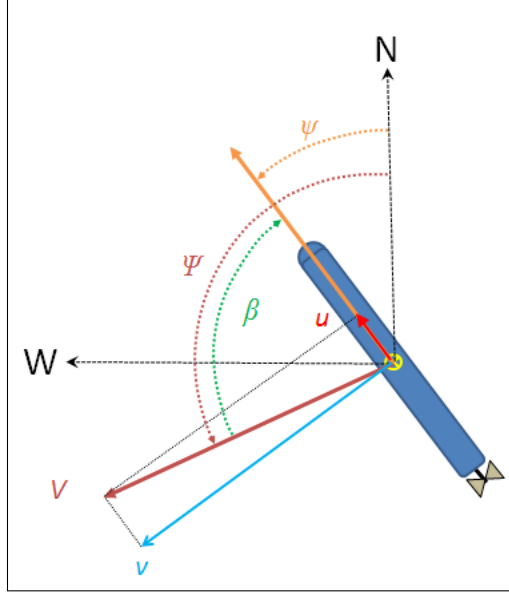
Figure 4.5: Trajectory Angles in the Horizontal Plane

and the time derivatives of the angles are calculated by

$$
\begin{aligned}
\Psi'(k) &= \frac{x'(k)y''(k) - y'(k)x''(k)}{(y'(k))^2 + (x'(k))^2} \\
\dot{\Psi}(k) &= \lambda(k)\Psi'(k) \\
\dot{\psi}(k) &= \lambda(k)\psi'(k) \\
\dot{\beta} &= \dot{\Psi} - \dot{\psi}
\end{aligned}
\tag{4.17}
$$

The angular relationships are arrived at by the motion dynamics of the vehicle in the horizontal plane. Consider Figure 4.5. In this figure, V is the velocity vector of the vehicle in the NED local tangent plane (LTP), and u and v are the surge and sway velocities respectively. The yaw angle ψ is defined as the direction that the positive x-axis of the vehicle in the body centered frame {b} makes with the Northerly axis of the LTP. The heading angle Ψ is the angle between the North axis of the LTP and the velocity vector V. β is defined as the angle between the the velocity vector and the positive x-axis of the vehicle in {b}. Thus $\beta = \Psi - \psi$.

4.4   Path Probabilistic Analysis

The probabilistic analysis portion of the routine occurs immediately after the inverse dynamics calculations. It consists of several subroutines that are executed at each discrete point along each

38

trajectory. This part takes as inputs the vector of coordinate values (x, y) of the geometric center of the vehicle along the entire trajectory, and the corresponding vector of yaw angles ψ for each point. It executes subroutine functions at each point for performing a sonar sweep, updating the occupancy grid, and calculating information gain, as well as performing the necessary analyses for heuristics violations and obstacle avoidance.

4.4.1   Generating the Sonar Image

In order to generate a simulated sonar image of the environment, a *"World Model"* was first constructed to represent the spatial environment that the vehicle was to be operated in. For this thesis, that environment was to be a model of the Center for Autonomous Vehicles Research (CAVR) water test tank located in the basement of Halligan Hall on the NPS campus. This open top tank measures approximately 10 meters by 20 meters. The tank is modeled in MATLAB with a simple matrix. The matrix cell values at and outside the tank walls have a value of 1, while those inside the walls (representing water space) have a value of 0. The sonar image is generated by a MATLAB function that performs a sonar sweep against the World Model (*sweep.m*). The sonar sweep function casts a single sonar ray along a Bresenham line that has a length equal to the effective range of the sonar. If all World Model cells along this ray hold a value of zero, then the function chooses a random value from an equally distributed range of possible signal strength values for a "no-object-in-beam" condition. Conversely, if a cell with a value of 1 is found along the ray, then the function chooses a random value from an equally distributed range of possible signal strength values for a "no-object-in-beam" condition. The ranges for "object-in-beam" and "no-object-in-beam" signal strength values are bounded by the minimum and maximum likely signal strengths for each condition. In the "object-in-beam" case, the function then determines the World Model matrix coordinates for the first cell along the ray with a value of 1, and stores the returned signal value in the Sonar Image matrix cell corresponding to the same coordinates of the World Model. This ray-tracing routine is repeated for a discrete number of rays about the entire swath of the sonar sensor, which, for the FLS on the REMUS, is $\pm 45°$ of the bow. An example of the generated cumulative sonar image, with one object in the tank, is shown in Figure 4.6. This image is a compendium of all of the individual sonar images developed for each sonar sweep for the entire trajectory.

4.4.2   Updating the Occupancy Grid

The occupancy grid is initialized as a matrix with the same dimensions as the given World Model, with all cells assigned the value of 0.5 (see Section 2.3). It is then updated at each point
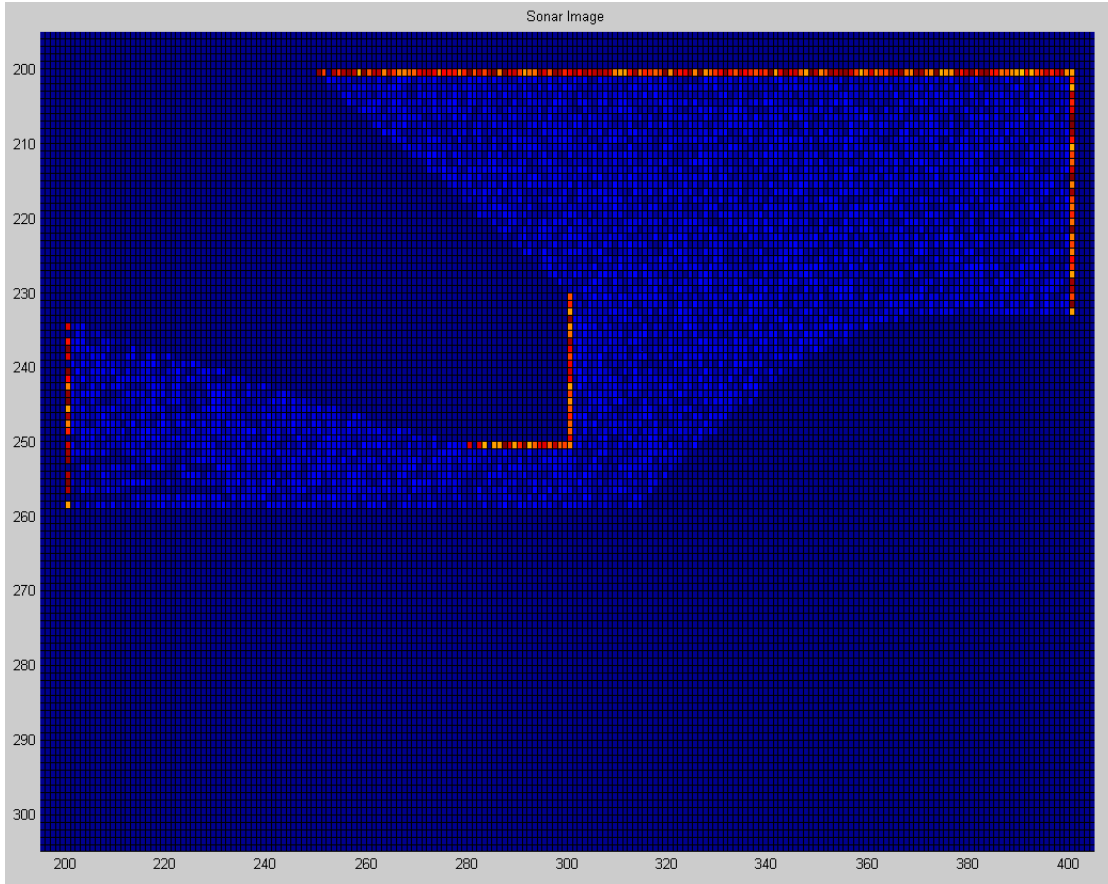
Figure 4.6: Sample Cumulative Sonar Image

along the trajectory by an OG update subroutine (*OGupdate.m*). This function takes the sonar image generated in the previous subroutine and performs the Bayesian inference described in Section 2.3 and defined by equation (2.12) to update the occupancy probability of each cell. In order to obtain $P[r_{t+1}|s(C) = Occ]$ and $P[r_{t+1}|s(C) = Emp]$, a probabilistic model of the sensor must first be built. This is performed by yet another subroutine function (*SensorModelGeneration.m*). In this function, the probabilities that a given sensor value is obtained given either an "object-in-beam" or "no-object-in-beam" state are modeled as continuous probability density functions (pdf). Each of these models was generated based on data from McChesney[4]. The pdf's for the sonar sensor model are plotted in figure Figure 4.7. The occupancy grid updater processes each individual cell of the sonar image and determines the $P[r_{t+1}|s(C) = Occ]$ and

---

[4]McChesney utilized histogram models for sensor modeling. Continuous pdf's were used in this thesis due to the lack of availability of his histogram data and insufficient time for actual in-tank sensor data collection. For the objectives of this thesis, the models used are deemed sufficient for proof-of-concept testing.
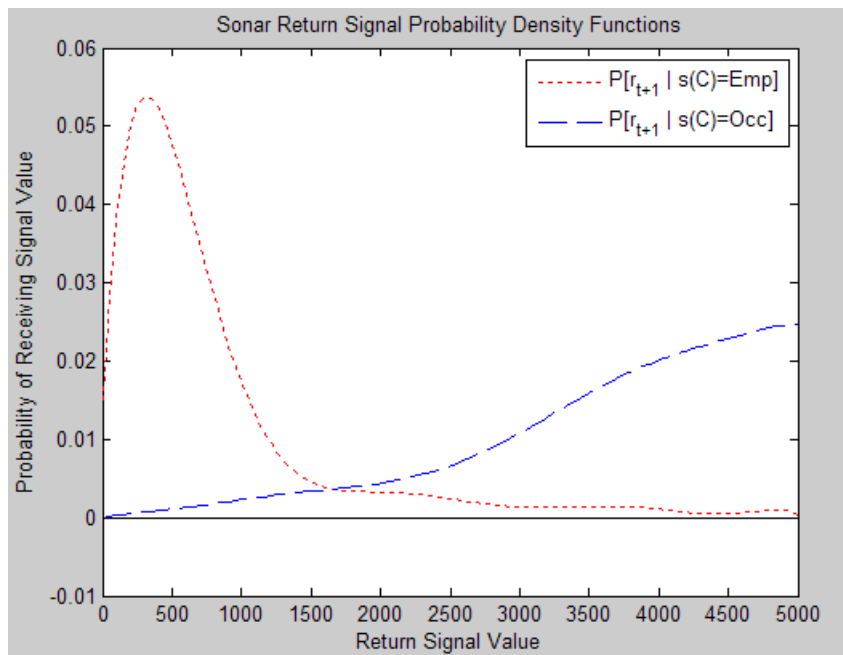
Figure 4.7: Sensor Model Probability Distribution Functions

$P[r_{t+1}|s(C) = Emp]$ based on the signal value stored in the cell and the probability value of receiving this signal value based on the generated sensor model. The remaining unknown variable in equation (2.12), $P[s(C) = Occ|\{r\}_t]$, is the previous occupancy probability value of that cell. The newly calculated occupancy probability value for this cell will then be the prior value for the next iteration of the Bayesian inference. Thus as more sonar sweeps across each cell happen, the occupancy probability for each cell is constantly being updated, eventually converging on the actual occupancy state of the cells. In this way, the occupancy grid is constantly increasing in confidence. A sample occupancy grid, based on the data used to produce the sonar image in Figure 4.6 is shown in Figure 4.8.

4.4.3   Calculating Information Gain

The same function that updates the occupancy grid also calculates the incremental information gain at each step. As the occupancy probability of each cell is continually resolved by the Bayesian inference, our knowledge of the actual state of each cell is constantly increasing, while the information entropy is correspondingly decreasing. By applying the concepts discussed in sections 2.1.2 and 2.2 the amount of change of probability values of each cell at each iteration directly corresponds to the information gain achieved during that iteration. The information gain is thus calculated during each update as the difference between the prior and posterior
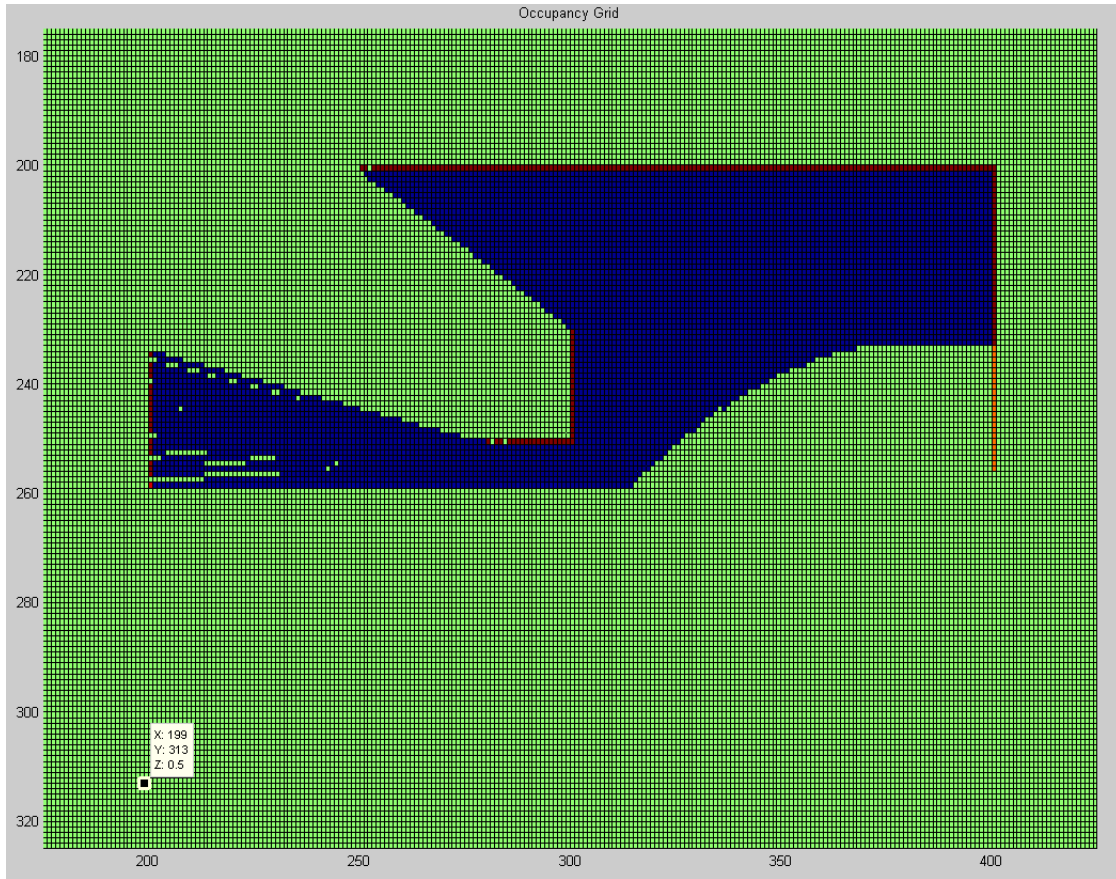
Figure 4.8: Sample Occupancy Grid

occupancy probabilities of each cell. The net information gain over the entire trajectory is then the cumulative sum of the information gain achieved over every iteration for all cells.

4.4.4 Heuristic Boundaries

At the start of the routine, the vehicle starts with no information about its environment. Even after the first sonar sweep, the only knowledge it has is a low-confidence probabilistic estimate of what lies in the sonar cone of that initial sweep. In order to allow for freedom of motion at this beginning phase of the path, certain assumptions must be made about the region immediately surrounding the vehicle.

In this vein, an initial "box" of free space is established around the vehicle so that it may maneuver through unknown space until sufficient data has been collected to permit normal exploration-based maneuvering based on collected data. The heuristic bounding box assumes

42

that the walls detected during the initial sonar sweep extend beyond the edges of the sonar cone by a fixed amount sufficient to allow for an assumption of no objects or obstacles in a path traveling to the immediate left or right of the vehicle. The forward looking sonar dictates a bias towards forward exploratory motion, thus heuristic boundaries astern of the vehicle originate at the sternmost point of the vehicle. With these considerations in mind, one side of the heuristic bounding box at the vehicle's stern is defined as a North-South line in the local tangent plane (LTP) frame originating at the sternmost point of the vehicle and extending in the N-axis direction corresponding to that of the N-axis component of the vehicle centerline vector. The other stern-side is similarly defined in the E-W direction in the LTP. The remaining two sides are subject to the shape of whatever is seen during the initial sonar sweep, and thus are more difficult to explicitly define. It should be noted that if a secondary sensory system were utilized, such as a side-scan sonar, the augmented sensory coverage to the port and starboard side of the AUV would likely provide sufficient information about the adjacent environment at the start of the routine such that these heuristic boundaries would be unnecessary for an AUV with cross-body thrusters.

In the case of this thesis, however, the vehicle starts in a corner of the tank, so the first assumption above is applied to the walls of the tank seen during the initial scan. The remaining two (forward) heuristic boundaries are then defined by extending the lines of the walls seen by the sonar beyond the sonar cone. These lines originate at the corner where the right and upper tank walls meet, extending beyond the initial sonar spread and terminating where the left and lower boundaries intersect with the walls. The resulting boundaries form a box that can be seen as the red dotted lines in Figure 4.9. These boundaries are used in two ways. First, the occupancy grid cells along the heuristic boundaries at the walls are initialized to a value of 0.75, vice 0.5. Increasing this initial value reduces the possible information gain achieved by sensing the walls within the heuristic bounding box, thus weighting subsequent explorations away from these wall sections. Basically since we are already assuming the existence of the wall along the boundaries, we want the information gain accrued when sensing these walls to be smaller than that achieved by exploring unknown areas. Second, the boundaries at the stern of the vehicle are used to penalize the vehicle for backing down into unknown (unexplored) territory. At the outset, the area astern of the vehicle that lies beyond the stern heuristic boundaries is completely unknown, and so there obviously exists the possibility for obstacles or obstructions in those regions. Any generated trajectory that causes the vehicle to back down into this space

43
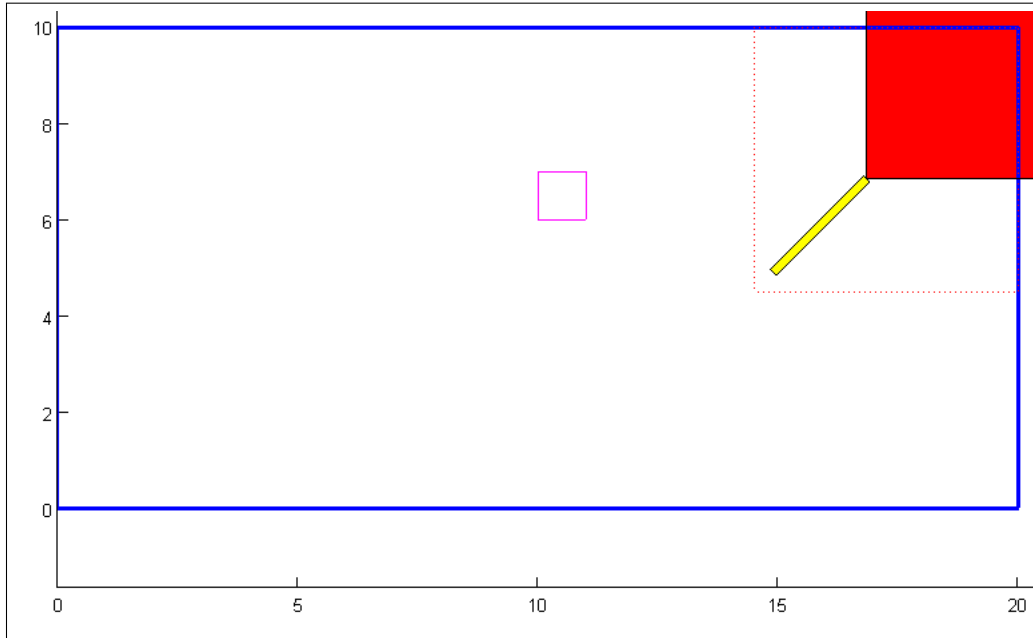
Figure 4.9: Heuristic Boundaries, represented by the red-dashed lines. The blue rectangle represents the test tank walls; the pink square is an obstacle in the tank; The red box represents the sonar cone of the FLS.

before it has been explored is penalized proportionally to the extent by which the stern violates this boundary. The penalty function for this will be discussed further in Section 4.5.

4.4.5   Obstacle Avoidance

An obstacle avoidance subroutine is necessary to prevent the vehicle from coming into contact with any obstacles or objects in the water. Since a vehicle with cross-body thrusters, such as the NPS REMUS, is capable of significant lateral translation, it is not sufficient to base the avoidance criteria simply on obstacles ahead of the vehicle. Thus a method was developed that circumscribes a circular "avoidance perimeter" about the centroid of the vehicle with a radius equal to the length of the vehicle. Each trajectory is then analyzed at each point along the path for any intrusion of *known* object boundaries within this intrusion perimeter. The penalty assigned to this intrusion is proportional to the penetration depth into the perimeter. The data used for "known" objects is sourced from the OG, meaning that only objects or obstacles that have been "seen" by the sonar are used in this analysis. This fact, combined with the sideslip capability of the vehicle, led to the requirement for two other subroutines that bias the trajectory into explored space, and limit the amount of sideslip into unexplored space. These subroutines will be discussed further in Section 5.2.

44

4.5 Optimization Scheme (*fminsearch*)

The routine now proceeds with the optimization portion of the DM-IDVD process. A cost function (CF) that calculates penalties for violating given constraints is calculated, as well as a performance index (PI) of the optimized parameter, information gain. As stated previously, the entire calculative routine is evaluated within MATLAB's *fminsearch( )* algorithm. This function is a gradient-free minimization routine that seeks to find a local minimum of the input function. In the case of the DM-IDVD routine, the input function $f(x)$ (*trajectory.m*), consists of all calculations examined thus far, and terminates with the calculation of the proceeding CF and PI. The value to be minimized is thus the arithmetic sum of these two trajectory function outputs. The *fminsearch* function also requires an initial guess on the neighborhood of values for the variable parameters in the input function. It then iterates by varying the values of these variable parameters until either the local minimum of the function value $f(x)$ maximum (user-specified) number of iterations is reached. Values are also specified for the convergence tolerance for both $f(x)$ and x.

4.5.1 Cost Functions and Performance Index

Penalties in the cost function were initially specified for the final time, yaw rate, surge velocity, sway velocity, heuristics violations, and allision avoidance:

Time: Penalty for exceeding desired trajectory time. This desired time is specified as the *time horizon TH*. Since this isn't a critical penalty, a lower relative weighting factor is assigned to it.

$\dot{\psi}$:   Penalty for exceeding max yaw rate.

u: Penalty for exceeding max surge velocity.

v: Penalty for exceeding max sway velocity.

Heuristics Violation: Penalty for violating heuristics boundaries. This prevents backing into the unknown region astern of the vehicle at the start of the trajectory.

Allision Avoidance: Object/obstacle proximity penalty. The closer the path comes to an object or obstacle, the higher this penalty. This is based on a specified avoidance radius centered on the vehicle centroid.

Table 4.2: Cost Function Components

Penalty	Formula
Time	$Cost_T = (time(end) - TH)^2$
$\dot{\psi}$	$Fine_{\dot{\psi}} = \max([0, (abs(\dot{\psi}) - \dot{\psi}_{max})])^2$
u	$Fine_u = \max([0, (abs(u) - u_{max})])^2$
v	$Fine_v = \max([0, (abs(v) - v_{max})])^2$
Heuristics Violation	$Cost_H = ViolArea^*$
Allision Avoidance	$Cost_{AP} = AP^{**}$

Notes: *–ViolArea is the calculated are between the lower heuristic boundary and the arc swept by the vehicle's stern in violation of this boundary. **–AP is the calculated by a subroutine that assigns a cumulative penalty of all of the individual proximity violations at each point along the trajectory, which are directly proportional to the magnitude of the incursion of the object into a circle described about the vehicle centroid with a given avoidance radius.

Table 4.2 show the formulas used in calculating the penalties in the cost function.

The *performance index* is simply the inverse of the cumulative information gain for the generated trajectory, and is calculated as $Cost_{IG} = \frac{1}{IG}$. The inverse is used to maintain consistency with the other cost functions in that it is desired to minimize each within the *fminsearch* function. Thus minimizing the inverse of IG means that IG will, in fact, be maximized.

Each of these CF/PI factors is multiplied by a weighting factor and summed. This sum of products is the value that is actually minimized by the *fminsearch* function. Once the minimal value is found, the "optimal" trajectory that produced it is returned, including all state parameters (*x*, *y*, and *psi*), as well as the states and controls calculated by the inverse dynamics. These outputs are the final product of the routine–an optimal trajectory that satisfies all constraints imposed on the problem and maximizes the optimization parameter, Information Gain.

CHAPTER 5:
Data Analysis and Findings

The routine described in Chapter 4 was built and run entirely in the MATLAB computing environment. All pertinent MATLAB functions and scripts are contained in Appendix A on page 73. The following sections discuss the numerical results of the simulation, as well as the resulting trajectories for three simulation scenarios of the REMUS vehicle in the test tank with zero, one, and two obstacles in the path of the vehicle. Constraints on surge (u_{max}), sway (v_{max}), and yaw rate (ψ_{max}) were sourced from Doherty [22]. Initial guesses on the varied parameters and their corresponding final optimal values for the trajectory generation runs are shown in the corresponding tables for each set of runs. Prior to running the full simulation, the initial guesses were made empirically by running several simulation runs with a reduced set of cost functions (IG, EV excluded) to validate the trajectory generation routine. The final values for the varied parameters were used as a baseline for subsequent complete optimization runs. These values were adjusted as required until consistent results at the end points were observed. The simulation was then run repeatedly, varying the initial guesses of the variable parameters as well as the values for the weighting factors for the cost functions and performance index. The resulting state and kinematic parameters were then observed, and any adjustments required to ameliorate or minimize constraint violations were made to the weighting factors. This process was repeated until a satisfactory trajectory was obtained.

The following sections discuss the three scenarios analyzed: no object in the tank, one object in the tank, and two objects in the tank. Three representative runs from each scenario are discussed, including the values for the variable parameters and weighting coefficients, as well as the results for each run.

5.1 Scenario 1: No obstacles

The set of runs for the first scenario were made with no obstacles in the tank. Values for the pertinent variable parameters for each run are shown in Table 5.1. The values for the weighting factors for each run are shown in Table 5.2.

Table 5.1: Variable Parameter Initial Guess Values for Scenario 1

	1	2	3	4	5	6	7	8	9	10
Run 1	10	1	0.7854	0.5	2.3562	0.126	-0.054	0.0017	0.0009	-1.5708 rad
Run 2	10	1	0.7854	0.5	-0.7854	0.2	-0.05	0.017	0.0009	-1.5708
Run 3	10	1	0.7854	0.5	-0.7854	0.126	-0.054	0.017	0.0009	-1.5708

Variable Parameter Key:

1. Virtual Arc Length (τ_f)
2. Magnitude of final acceleration (m/s^2)
3. Direction of final acceleration (radians)
4. Magnitude of initial acceleration (m/s^2)
5. Direction of initial acceleration (radians)
6. Initial yaw acceleration (ψ_i'') (rad/s^2)
7. Final yaw acceleration (ψ_f'') (rad/s^2)
8. Initial λ''
9. Final λ''
10. Final yaw angle (radians)

Table 5.2: Cost Function Weighting Coefficients for Scenario 1

	1	2	3	4	5	6	7	8
Run 1	1	10	1	10	1	10	1	1
Run 2	1	10	1	100	10	10	1	1
Run 3	1	10	1	10	100	1	1	1

Weighting Factor Key:

1. Time
2. Yaw Rate ($\dot{\psi}$)
3. Surge Velocity (u)
4. Sway Velocity (v)
5. β
6. Heuristics Violation
7. Allision Avoidance
8. Explored Space

The set of runs of the full routine for scenario 1 were done with the vehicle initially positioned in the upper right section of the tank. The center of the vehicle was positioned one and a half ve-
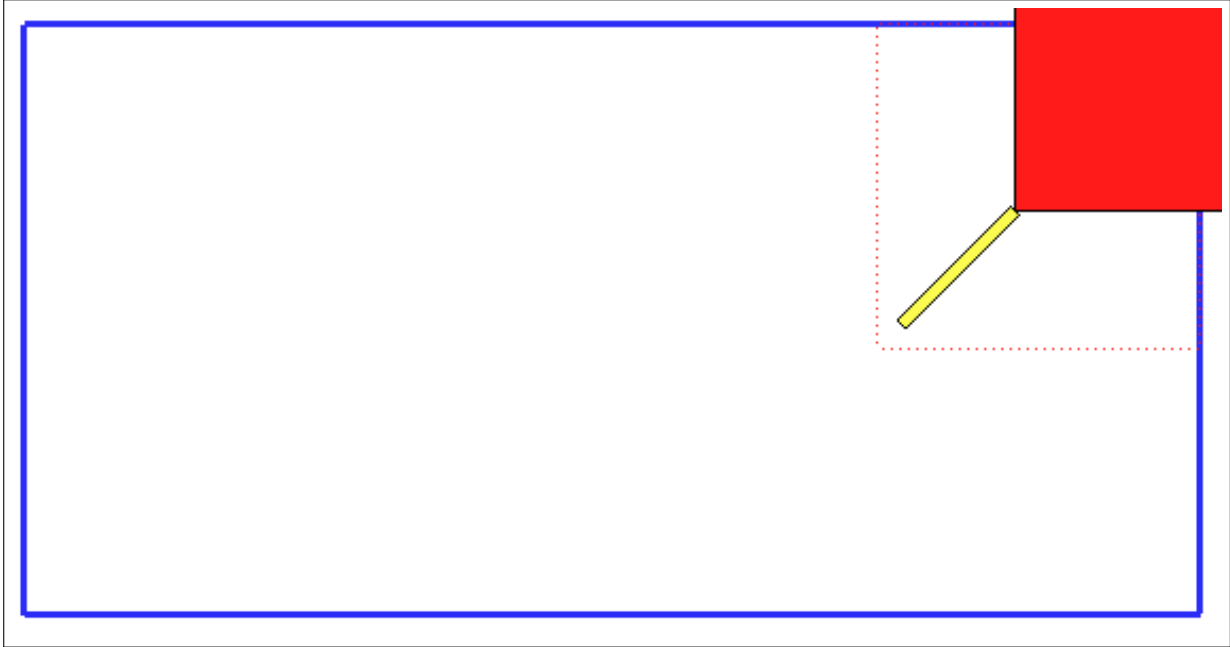
Figure 5.1: Scenario 1, All Runs, Initial Pose

hicle lengths from both the top and right (East) walls of the tank, and the yaw angle was chosen to be $+45°$ from due North, thus pointing the vehicle at the corner of the tank, as illustrated in Figure 5.1. This pose was chosen to simplify the heuristics assumptions and to put the vehicle into the maximally constrained initial state. The end point was then chosen to conform to the initial information space analysis discussed in Section 4.1. The probabilistic analysis portion of the routine, which included the sonar imaging subroutines, Bayesian inference, occupancy grid updating, heuristics analysis, and obstacle avoidance subroutines, was found to be quite computationally intensive, causing very long run times for a sufficient number of iterations. Because of this, the runs were initially performed without the probabilistic analysis of information gain to allow for quick validation of the basic setup of the algorithm. With no object in the tank, and no probabilistic analysis, the routine initially generated unsatisfactory trajectories. Without the sonar imaging and resulting occupancy grid, there was no capability for allision avoidance. Thus several of the initially generated trajectories actually crossed the boundaries of the tank walls before reaching the final point. Run 1 of this set is an example of this result, as seen in Figure 5.2. The values for the varied parameters and CF weighting coefficients were then adjusted and another set of runs were made. The trajectory for Run 2, illustrated in Figure 5.3, resulted in trajectories that remained within the confines of the tank, but violated constraints. With the trajectories at least now remaining within the tank, the probabilistic analysis subroutines were
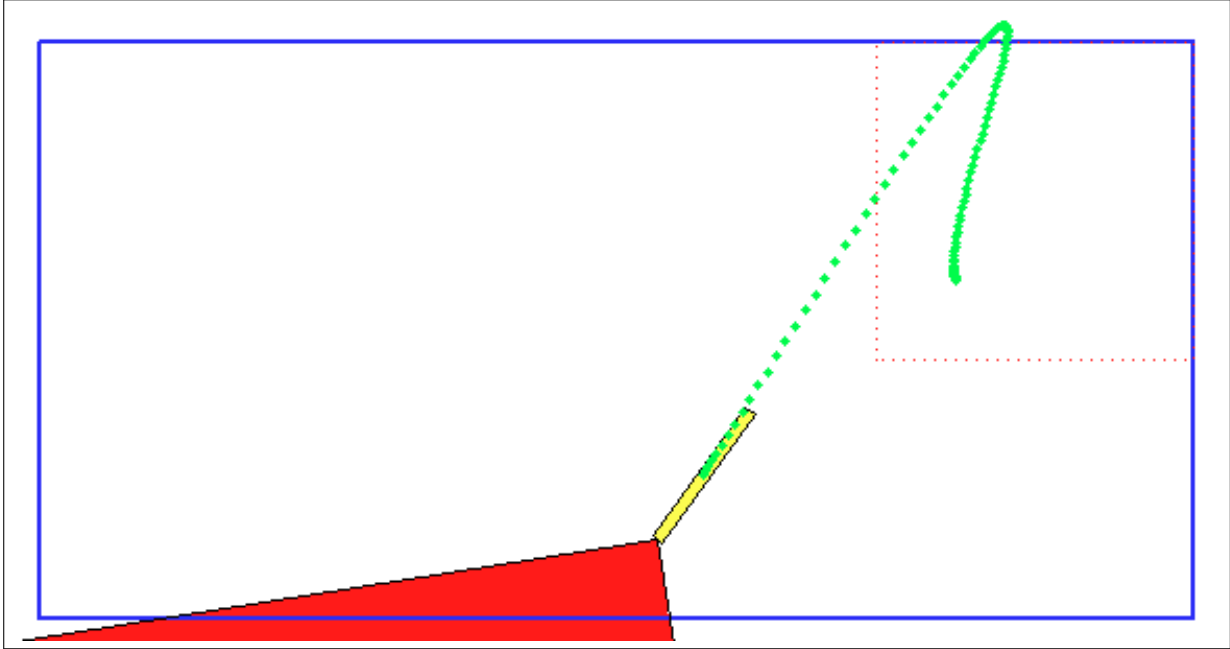
Figure 5.2: Scenario 1, Run 1 Trajectory

reimplemented back into the overall routine, and another set of runs was performed. Again, the CF weighting factors were repeatedly adjusted in an attempt to eliminate any constraint violations, however this was not achieved in this scenario. Figure 5.4 shows the best trajectory achieved for the initial conditions in this scenario, and the corresponding occupancy grid can be seen in Figure 5.5. Figure 5.6 shows the violation of the max sway velocity (v), while Figure 5.7 shows the max yaw rate violation for this run. Subsequent attempts to ameliorate this problem failed to generate a fully satisfactory trajectory that met all constraints.

5.2 Scenario 2: One Obstacle

The set of runs for the second scenario were made with a single obstacle in the tank placed in the nominal path of the vehicle. Values for the pertinent variable parameters for each run are shown in Table 5.3. The values for the weighting factors for each run are shown in Table 5.4.

Table 5.3: Variable Parameter Initial Guess Values for Scenario 2

	1	2	3	4	5	6	7	8	9	10
Run 1	10	1	0.7854	0.5	2.3562	0.126	-0.054	0.0017	0.0009	-1.5708
Run 2	10	1	0.7854	0.5	-0.7854	0.126	-0.054	0.0017	0.0009	-1.5708
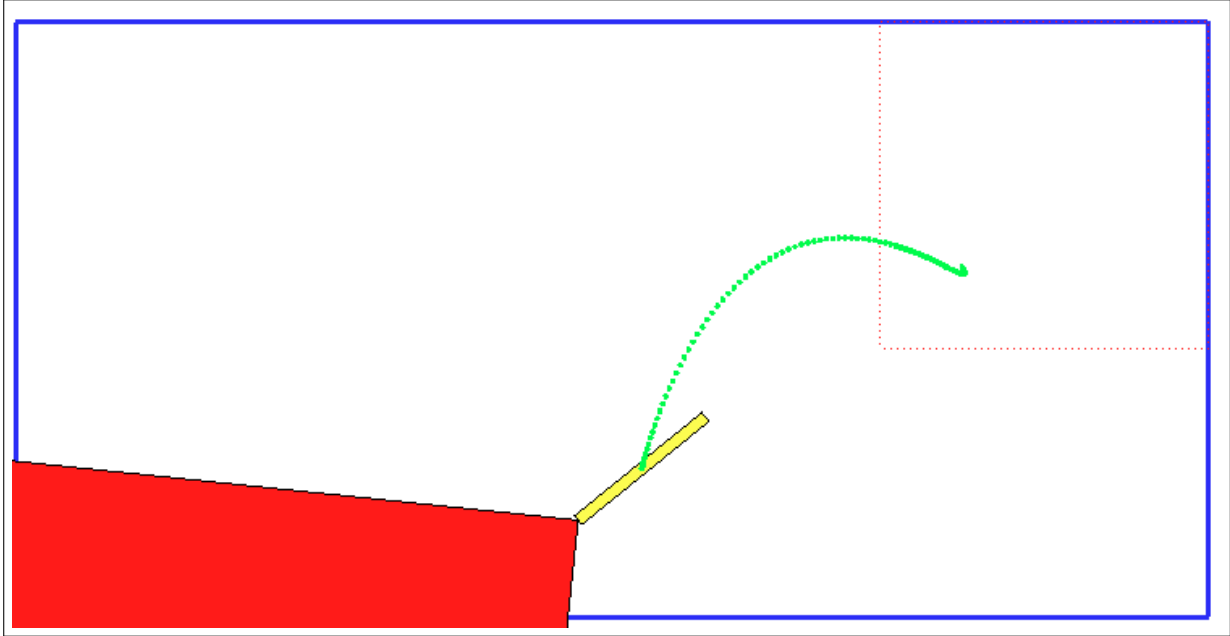
Figure 5.3: Scenario 1, Run 2 Trajectory

Table 5.4: Cost Function Weighting Coefficients for Scenario 2

	1	2	3	4	5	6	7	8
Run 1	1	10	1	10	1	10	1	1
Run 2	1	10	1	10	100	10	1	1

Although no satisfactory runs were made in the initial scenario, an attempt was made to continue testing with a single obstacle in the tank, in the hopes that further adjustments to the pertinent parameters combined with an obstacle in the tank would drive down the constraint violations. Initial conditions on the vehicle position and pose for the first two runs of this scenario were the same as those for the first scenario, illustrated in Figure 5.1. Figure 5.8 shows an example of the resulting trajectory with the parameters set as indicated in Tables 5.3 and 5.4. As seen in this figure, the generated trajectory starts in the Easterly direction. Numerous attempts made to modify the initial guesses for the initial acceleration angle to eliminate this behavior were fruitless. Trajectories that were forced to start in the Northwesterly direction continued to violate kinematic constraints.

When the initial guesses on the variable parameters were adjusted to force the start of the trajectory away from the corner, the algorithm repeatedly generated trajectories that caused some degree of allision with the obstacle during its transit to the final point. Most of these trajectories
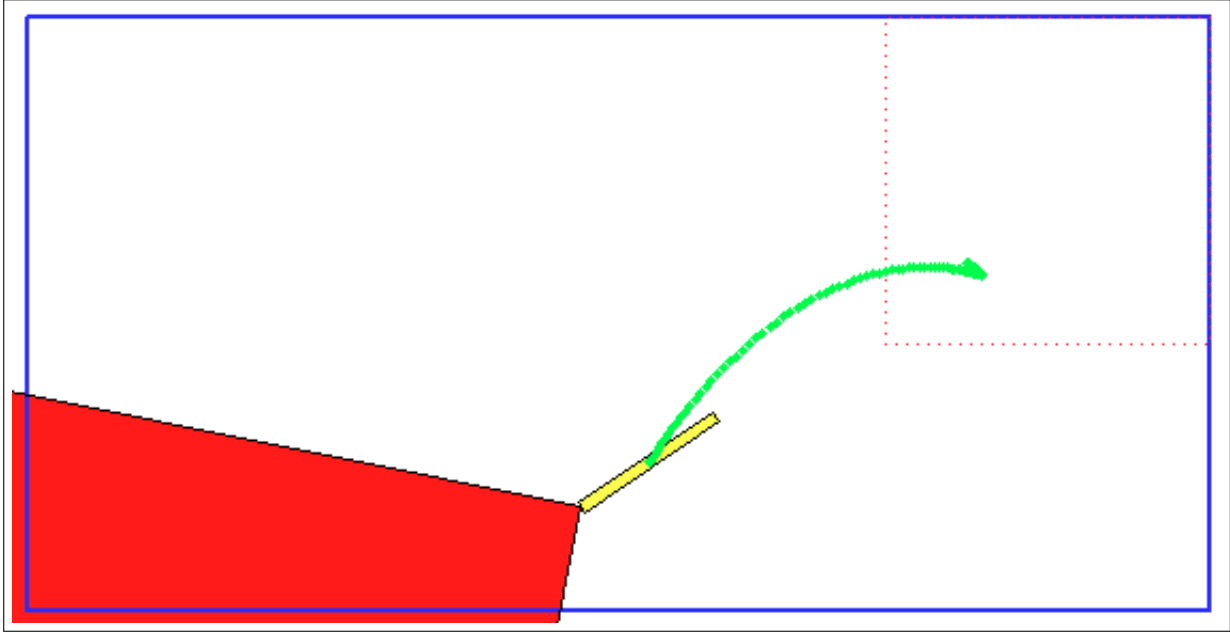
Figure 5.4: Scenario 1, Run 3 Trajectory

were predominantly side-slip trajectories, with little surge velocity. Inspection of the resulting penalties revealed an unusual result: the value of the allision avoidance penalty was zero. Upon viewing the generated sonar images and trajectory animation, the reason became clear–in order to satisfy constraints on time and side slip, the generated trajectories were fairly close to linear trajectories between the initial and final point, with a small amount of curvature. However, due to constraints on maximum yaw rate, the vehicle was not able to slew around to the point where the object was in the sonar field of view. Thus the vehicle was not "seeing" the obstacle at all. Figure 5.9 illustrates one such trajectory in which the stern of the vehicle actually allides with the object in the tank. Looking at the occupancy grid for this run shown in Figure 5.10, it is clear that the vehicle never even saw the obstacle. Two new penalties were then added to the cost function to account for this. First, a penalty on β, the angle between the yaw angle and heading, was specified. This fine penalizes the trajectory any time the magnitude of β ($|\beta|$) is greater than $45°$. This tries to keep the heading of the vehicle within $\pm 45°$ of the positive x-axis of the vehicle, thus favoring trajectories that "drive into" the sonar cone. Upon implementation of this, however, it was noted that, due to the flexibility of the trajectory in side-slip allowed by the cross body thrusters, the algorithm may still generate trajectories that travel in a direction parallel to one side of the sonar cone (i.e. $\beta \approx 45°$) at some point along the trajectory. Thus an obstacle might still lie in the path of the vehicle, and the sonar would never see it. So a
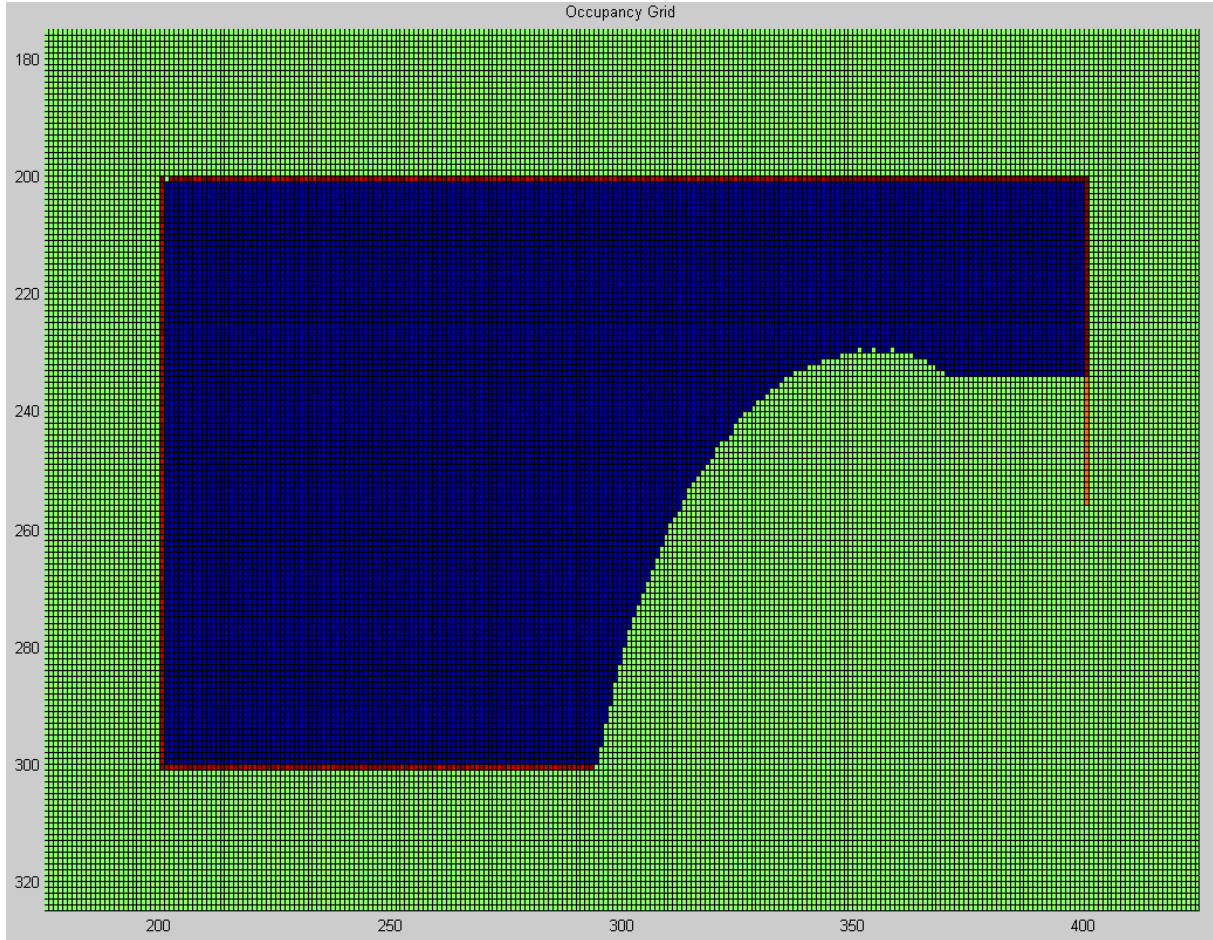
Figure 5.5: Scenario 1, Run 3 Occupancy Grid

second additional cost function penalty was added to favor trajectories that caused the centroid of the vehicle[5] to only traverse through "explored space" that has been swept by the sonar. The formulas used for these penalties are given in Table 5.5.

Table 5.5: Formulas for Added β and Explored Space Cost Function Penalties

Penalty	Formula
β	$Fine_\beta = \max([0, (abs(\beta) - \beta_{max})])^2$
Explored Space	$Cost_{EV} = 1/(1 + EV^*)$

---

[5]The centroid was chosen for simplicity of calculation within the routine. In theory, any point aft of the centroid along the vehicle centerline would be sufficient to impart a "moment" on the vehicle that would help "steer" it into explored territory.
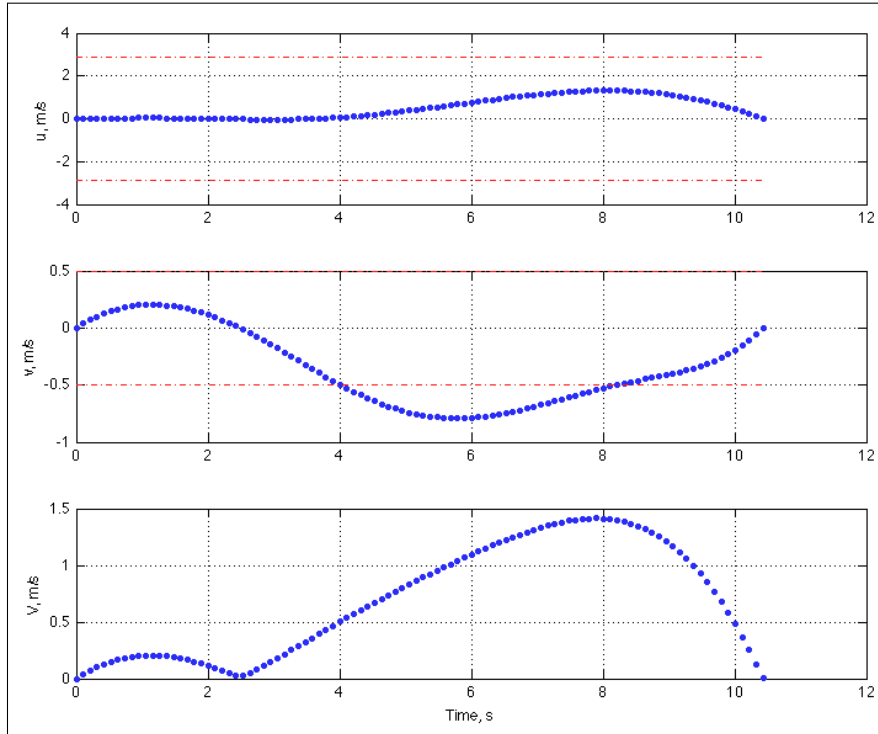
Figure 5.6: Scenario 1, Run 3 v_{max} Violation
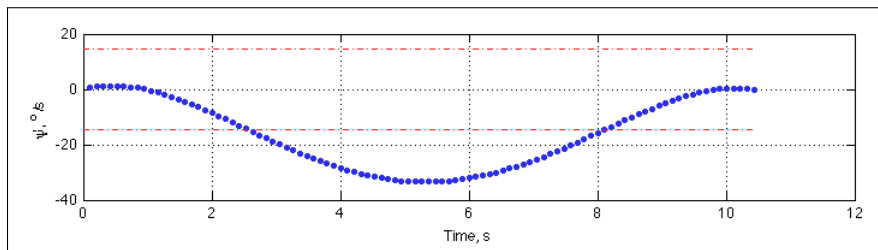
Figure 5.7: Scenario 1, Run 3 $\dot{\psi}_{max}$ Violation

Notes:*–EV is calculated by a separate subroutine that determines if, and by how far, the vehicle centroid lies outside of areas on the OG that have values not equal to 0.5, and thus have been swept by the sonar.

As before, some amount of adjusting of the weighting coefficient for these cost function terms was performed. However, satisfactory results were never obtained. The addition of the two additional penalties caused even more "competition" between the components of the cost function, and finding the right balance of weighting coefficient values proved to be incredibly difficult. Continued adjustments of the initial guesses on the variable parameters, especially those on ψ_i'' and the direction of initial acceleration, failed to sufficiently ameliorate this behavior. It seemed

54
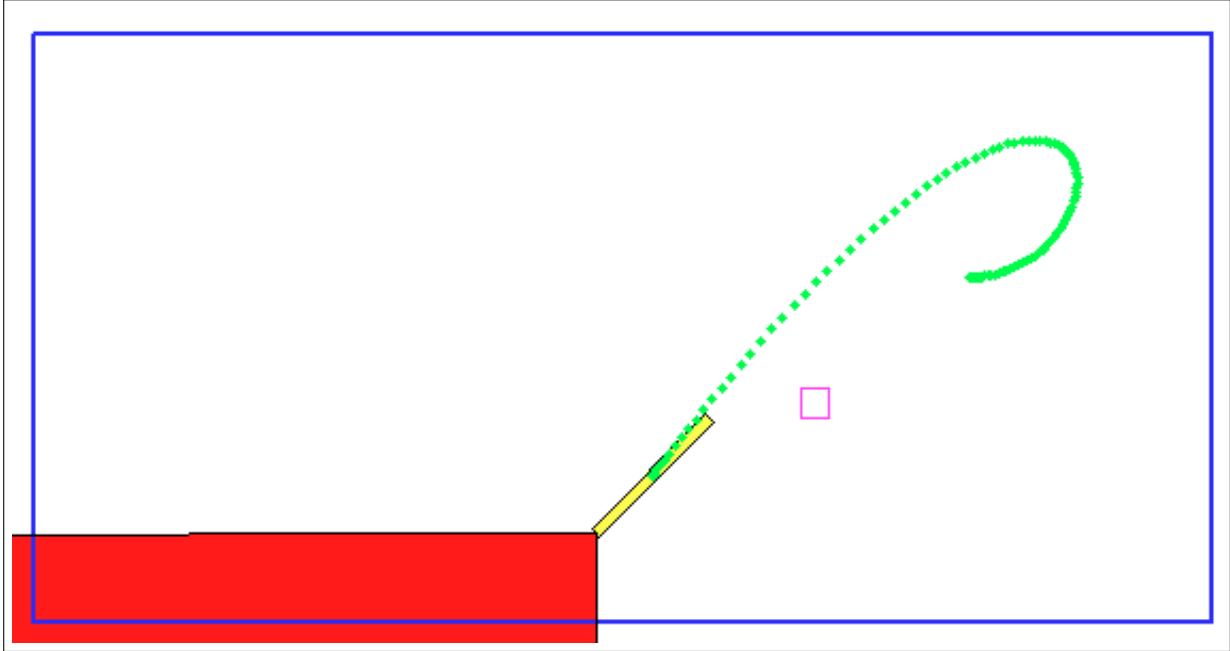
Figure 5.8: Scenario 2, Run 1 Trajectory

that it may be physically impossible to start with the given initial pose of the vehicle and steer around an obstacle to a final point that lie to the SouthWest of the vehicle, while minimizing violations of the set of physical constraints on u, v, and $\dot{\psi}$, as well as the heuristics. Further adjustments to the initial guesses on the variable parameters did improve the situation somewhat, but the generated trajectories still ended up being unsatisfactory.

Careful consideration was given to the reason for the failure of the algorithm to generate satisfactory trajectories. Clues were found when adjusting the weighting coefficient on each physical constraint during attempts to reduce violations of these constraints. For example, increasing the coefficient for the sway velocity to bring v within the v_{max} constraints consistently caused a larger violation of the constraint on $\dot{\psi}$. This lent credence to the hypothesis that the initial conditions were far too geometrically restrictive. Essentially, this meant that, with the initial pose of the vehicle facing the corner, it was practically physically impossible for the vehicle to traverse from the starting pose and position to the arbitrary final point. Thus the initial conditions were modified such that the vehicle was initially placed in a pose at the same starting coordinates, but facing due North. Promising results were obtained when starting from this pose, as the constraint violations immediately became much smaller, and so experimentation was then continued with this new starting pose.
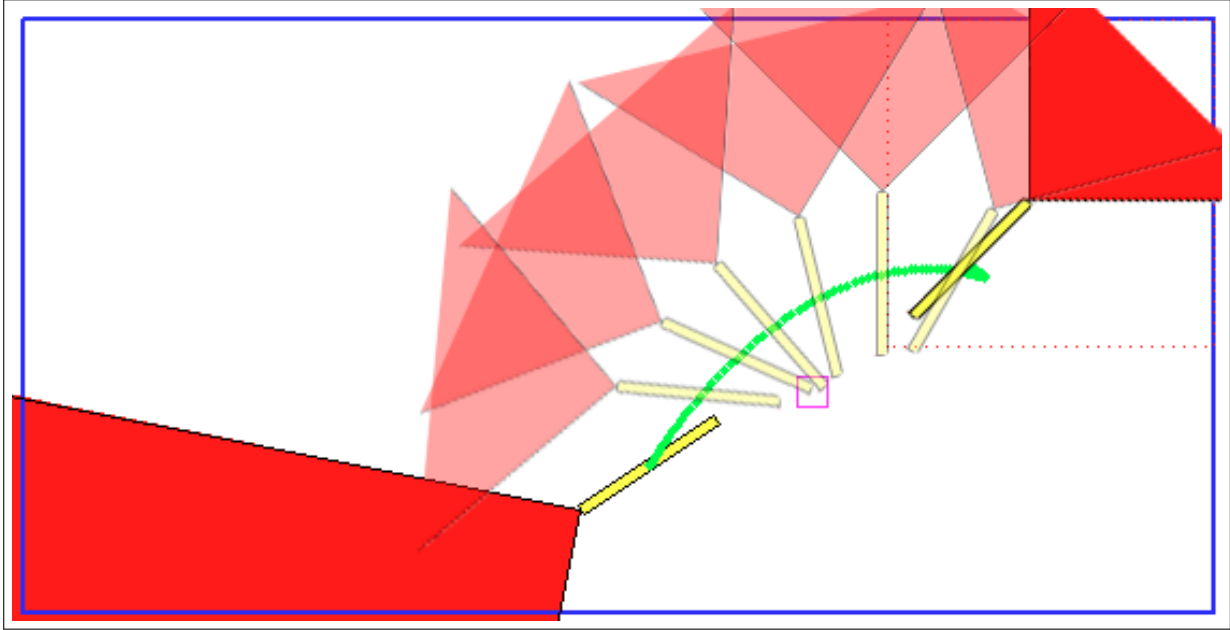
Figure 5.9: Scenario 2, Run 2: Vehicle hits object in tank

5.3 Scenario 3: Two Obstacles, North-facing Starting Pose

With the initial conditions now given by the North-facing starting pose, as shown in Figure 5.11, satisfactory results were eventually obtained for the single-obstacle case. The next step was to place two obstacles in the tank to determine if the algorithm would generate a satisfactory trajectory between both obstacles while optimizing the information gain. The same values from the immediately preceding runs were used for the initial values for the weighting coefficients and variable parameters. Additionally, the resulting animated plot of all of the initial two-obstacle runs revealed that the vehicle favored an initial rotation to starboard (clockwise), leading to many of these large violations. No amount of adjustment of either the weighting coefficients or variable parameter initial guesses eliminated this behavior. Again, the vehicle seemed to be too tightly physically constrained. The relatively short distance of just a few meters between the start point and end point, combined with the extra obstacle to contend with, proved too restrictive for the algorithm to handle given the physical constraints imposed on the vehicle. Again, another change was made to the boundary conditions, this time at the end point. The time horizon was increased to allow for a larger final range of motion, and the trajectory end point was adjusted accordingly, towards the West end of the tank. The two obstacles were moved such that they allowed for a sufficient impediment to travel between the start and end point. With this revised setup, the simulation was run, and the results were immediately clear–greatly reduced
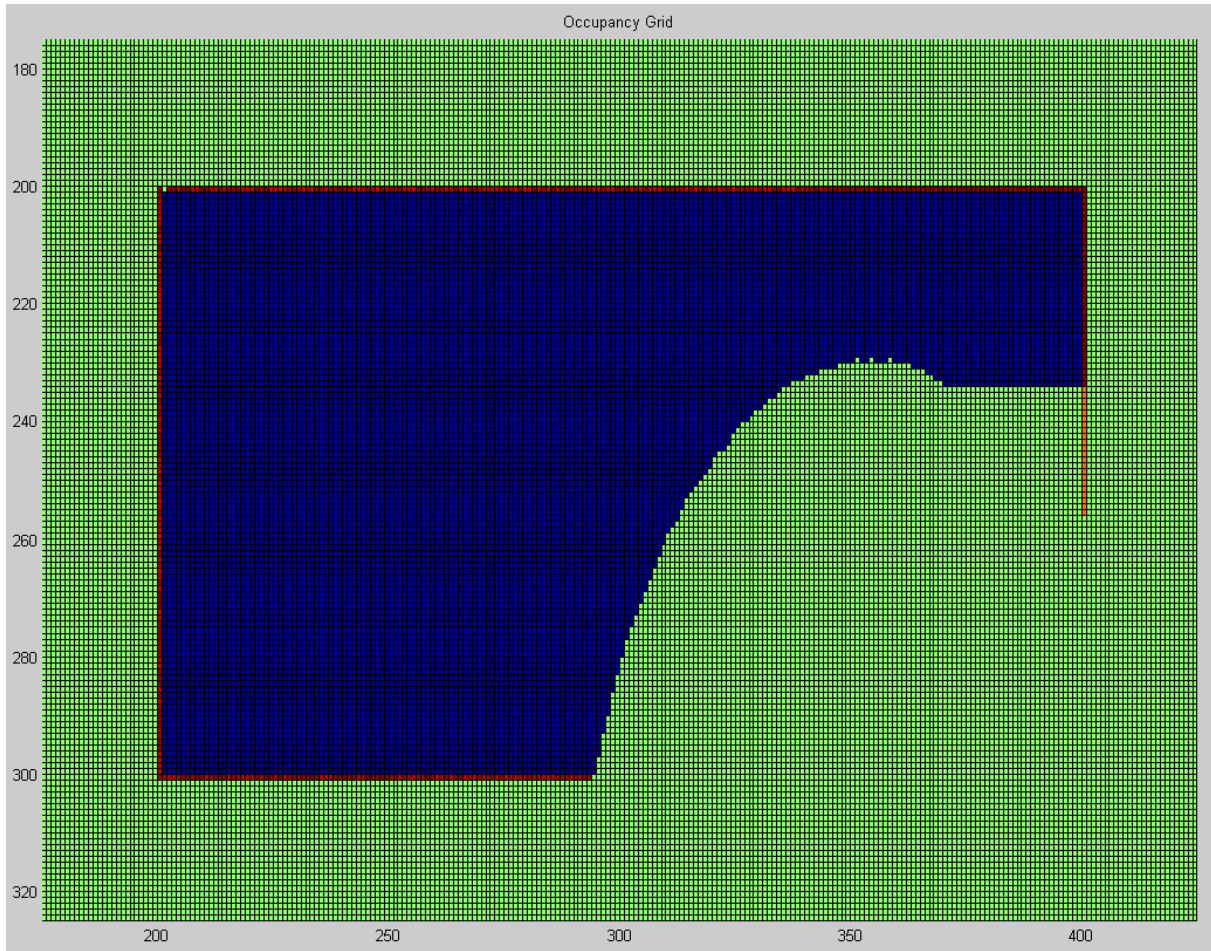
Figure 5.10: Scenario 2, Run 2 Occupancy Grid

physical constraint violations, and no allision with any object or wall. An example of an early run from this scenario can be seen in Figure 5.12. As before, initial testing resulted in some non-zero violation penalties. Most significant in the early runs were the fact that the vehicle still alluded with the upper obstacle.

Subsequent runs with adjusted values for the variable parameter initial guesses and weighting factors continued to improve the resulting trajectory and CF penalty values. An example of a better run can be seen in Figure 5.13. The vehicle successfully avoids both obstacles, but constraints in both v and ψ were still significant, as seen in Figure 5.14 (v), and Figure 5.15 (ψ).
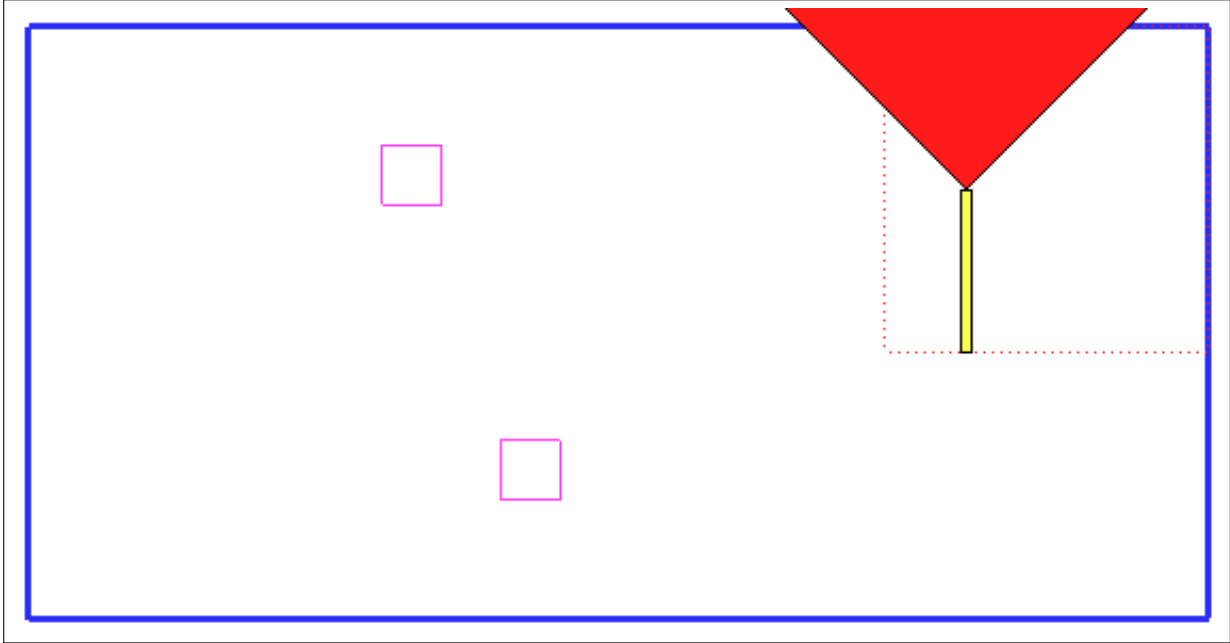
Figure 5.11: Scenario 3, All Runs, Initial Pose

Although the initial results of runs using the revised initial setup, with the Northward vehicle pose and increased trajectory length, were promising, they still did not meet the requirements for success. Physical constraints were still violated, and the performance index penalty was not low enough (approx. 4×10^{-4}). As before, adjustments were made to the weighting coefficients until satisfactory results were finally obtained. Values for the variable parameter initial guesses and weighting coefficients can be seen in Table 5.6 and Table 5.7.

Table 5.6: Variable Parameter Initial Guess Values for Scenario 3

	1	2	3	4	5	6	7	8	9	10
Run 1	10	1	0.7854	0.05	-2.3562	0.126	-0.054	0.0017	0.0009	-1.5708
Run 2	15	0.5	-0.7854	0.5	2.3562	-0.5	-0.054	0.0017	0.0009	-1.5708
Run 3	15	0.5	0	0.25	2.3562	-0.1	-0.054	0.0017	0.0009	-1.5708

Table 5.7: Cost Function Weighting Coefficients for Scenario 3

	1	2	3	4	5	6	7	8
Run 1	1	10	1	10	1000	10	1	1
Run 2	1	1	1	10	1	1	1	1
Run 3	1	100	1	10000	1	1	0.01	10

Figure 5.16 displays the final near-optimal trajectory of the vehicle. The initial point is indicated by the red star to the upper right. As can be seen, the vehicle successfully navigates

58
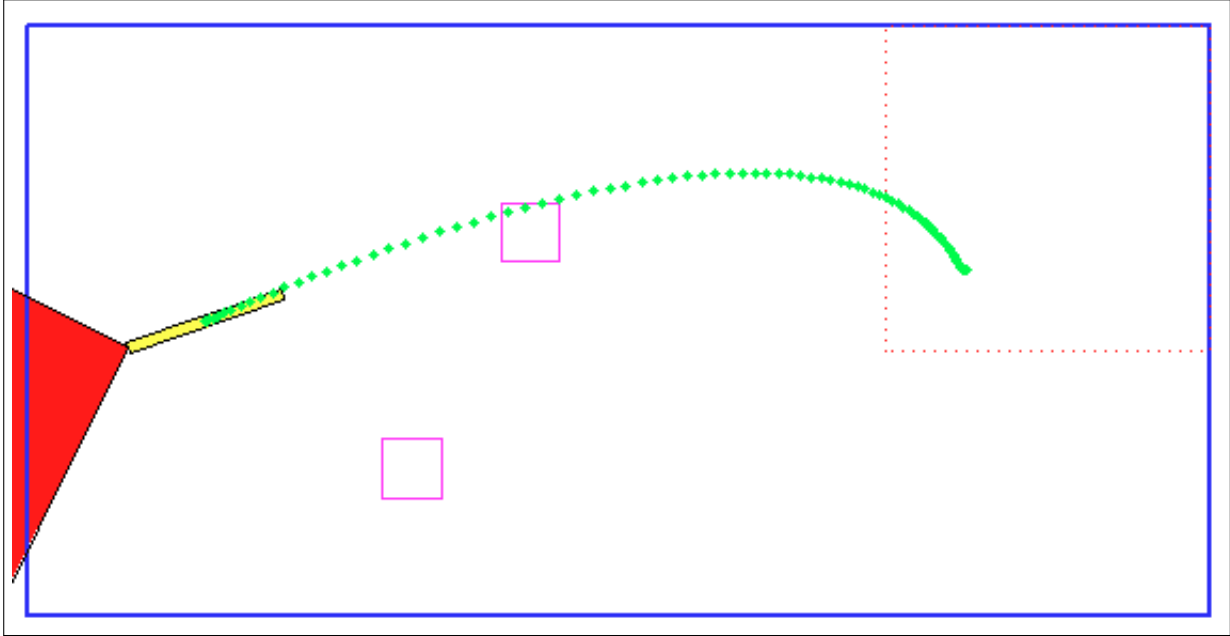
Figure 5.12: Scenario 3, Run 1 Trajectory

towards the defined end point without alliding with the fixed obstacles or the walls of the tank. Figure 5.17 shows the corresponding occupancy grid for the final successful run. Although this trajectory successfully navigated the trajectory with no constraint violations and an optimized PI, examining Figure 5.17 reveals that the sonar didn't actually "see" the lower obstacle from a planning perspective. From Figure 5.18, we can see that the vehicle, constrained in yaw rate, was not able to slew around sufficiently to sweep the sonar across the lower obstacle. A more complex trajectory with different values on the constraints would be required to see this obstacle. However, given that the algorithm successfully generated a trajectory that satisfied all of the requirements, notably the constraints and performance index optimization, this trajectory is deemed to be a valid near-optimal trajectory for the given conditions.

5.4 Computational Performance

One major issue that was readily apparent during the entire process was the computational intensity of the routine. Specifically, the subroutines associated with the sonar sweep, occupancy grid updater, and obstacle avoidance were especially intensive in both cpu cycles and memory requirements. Figure 5.19 shows output of the built in MATLAB profiler for the top 20 functions used in executing the entire algorithm routine. The profiler profiles every line of code as it runs and documents number of function calls, self time, and overhead time. By inspection, it

59
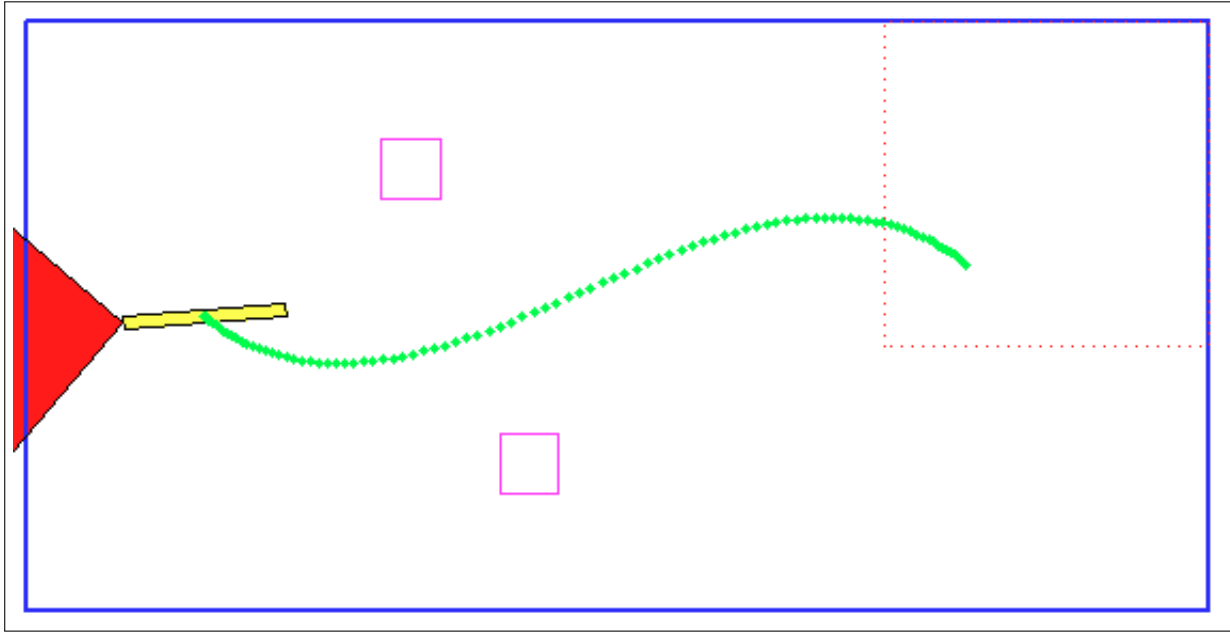
Figure 5.13: Scenario 3, Run 2 Trajectory

is readily apparent that just a few of the subroutine functions utilize a lions share of the computational time, and the impact is significant. In the final run that generated the satisfactory trajectory detailed in this section, the total computation time was 50 seconds[6]. As seen in the profiler output, 40% was used for the occupancy grid calculations (*OGupdate*), 37% for the sonar imaging (*sweep*), and 21% was used in the obstacle avoidance subroutine (*Avoidance*).

---

[6]All runs were performed in MATLAB 7.12.0.635 (R2011a) 32bit, on a PC running Microsoft Windows XP Professional with a 2.66GHz Intel Core2 Duo CPU with 3.25GB of RAM. Attempts were made to run the algorithm on an NPS distributed computing cluster, however, the MATLAB license on the cluster prevents actual distributed computing, and so minimal performance gains were actually realized.
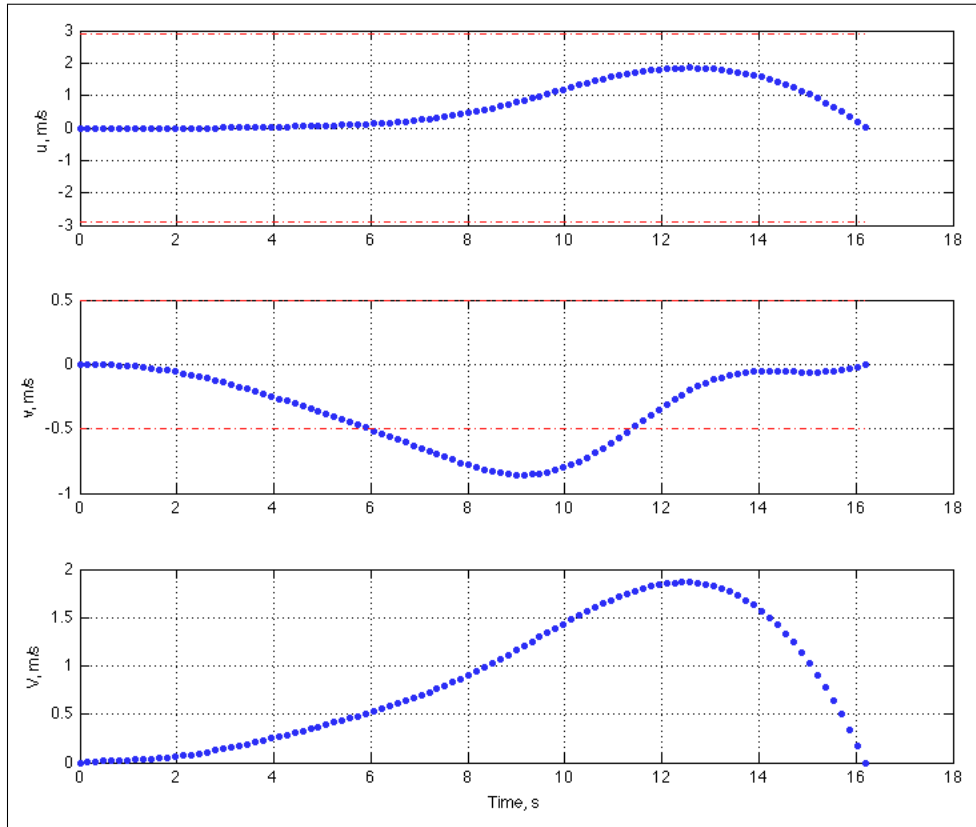
Figure 5.14: Scenario 3, Run 2 v_{max} Violation
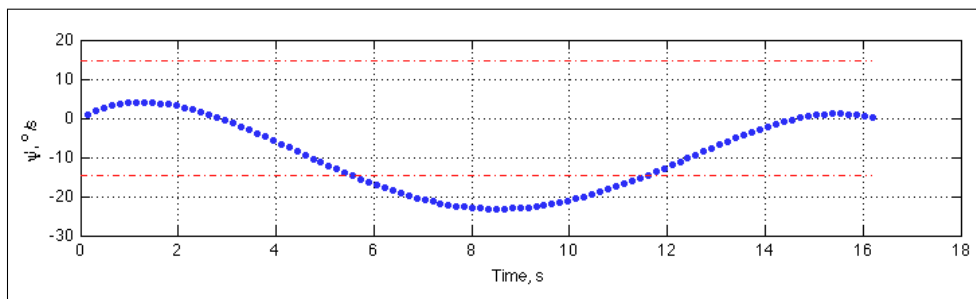
Figure 5.15: Scenario 3, Run 2 $\dot{\psi}_{max}$ Violation

61
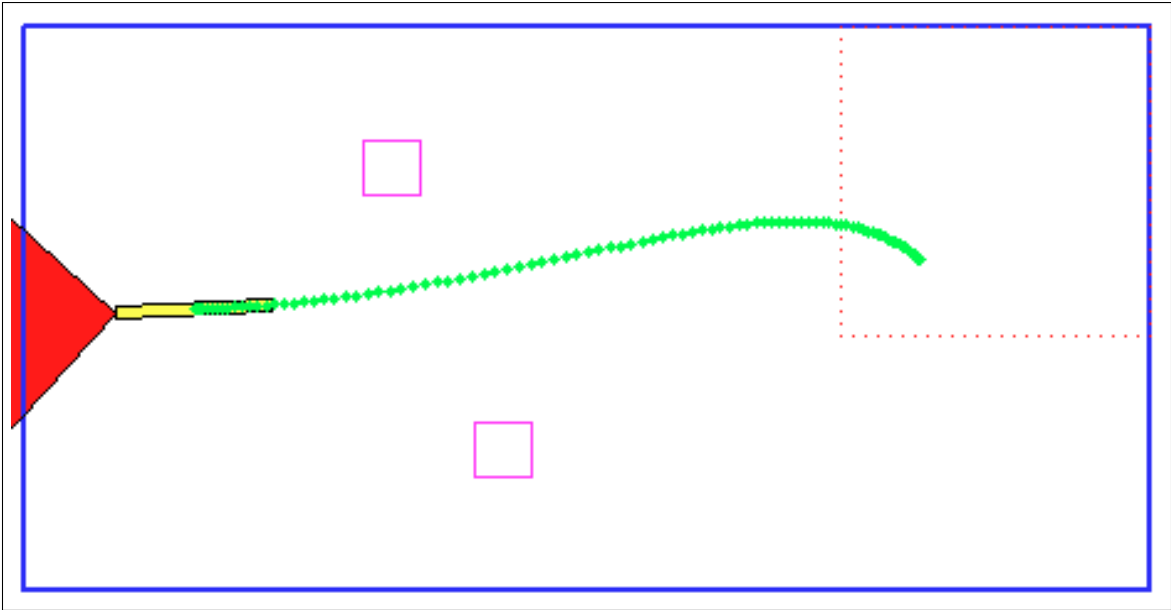
Figure 5.16: Final Near-Optimal Vehicle Trajectory
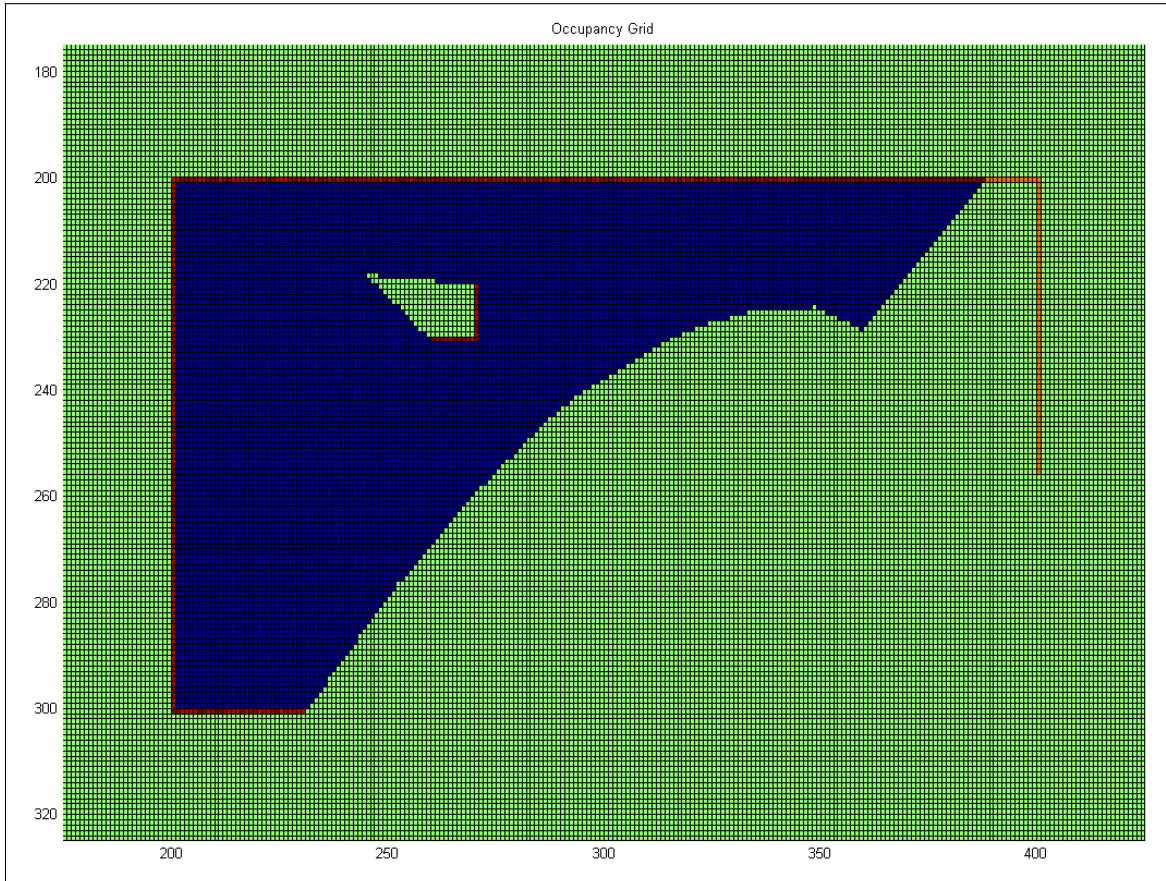
Figure 5.17: Occupancy Grid for Near-Optimal Trajectory
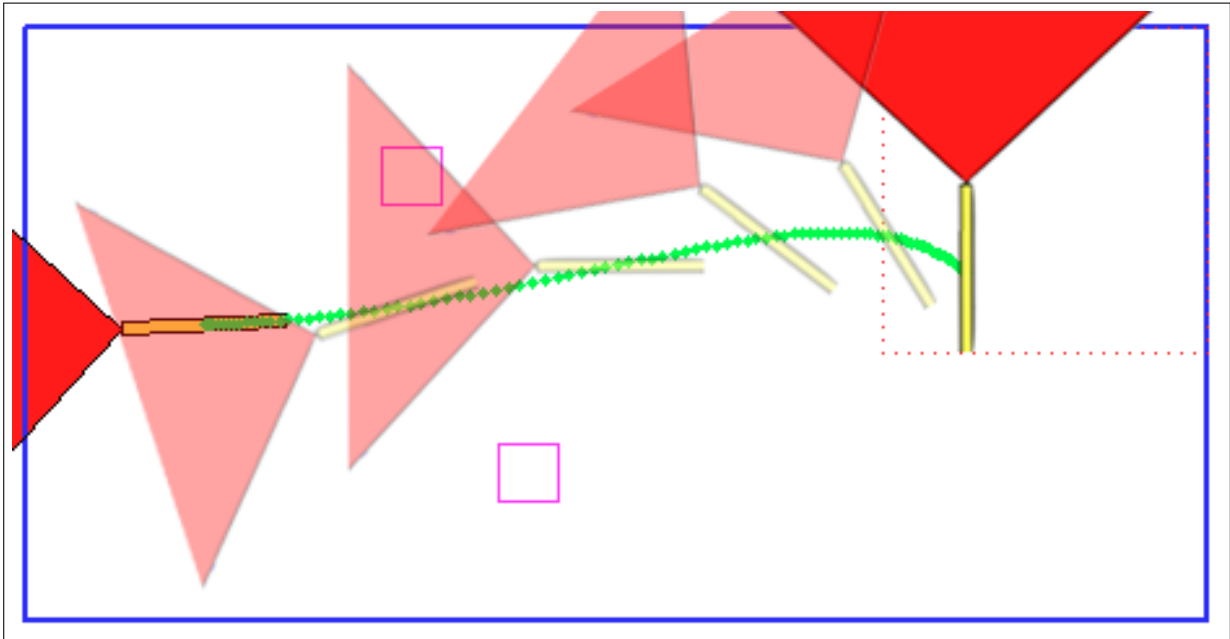
Figure 5.18: Vehicle Orientation at Several Points Along Final Near-Optimal Trajectory

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
fminsearch	1	3051.526 s	0.251 s	
trajectory	235	3051.213 s	1.751 s	
OGupdate	235	1242.796 s	1155.515 s	
sweep	23500	1133.336 s	1021.028 s	
Avoidance	23500	647.725 s	311.098 s	
bresenham	10622000	448.935 s	448.935 s	
polyval	4486742	87.405 s	87.390 s	
sym.subs	940	22.236 s	0.080 s	
sym.subs>mupadsubs	940	21.130 s	0.046 s	
sym.subs>tryFunctionHandle	940	21.084 s	0.345 s	
mupadmex (MEX-file)	3978	19.255 s	19.255 s	
sym.char	940	16.293 s	0.125 s	
InitMatrices	235	2.745 s	2.745 s	
sym.subs>getUnknowns	940	1.872 s	0.078 s	
sym.subs>makeFhandle	940	1.157 s	0.750 s	
sym.subs>getVars	940	1.026 s	0.110 s	
RefFuncs	1	0.969 s	0.078 s	
sym.findsym	940	0.916 s	0.094 s	
sym.subs>evalableY	940	0.872 s	0.201 s	
cell.setdiff	1880	0.735 s	0.296 s	

Figure 5.19: MATLAB Profiler Output for Final Optimal Trajectory Run

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 6:
Conclusions and Recommendations

6.1 Conclusions

As seen in Chapter 5, valid results were obtained for the problem of generating an optimal trajectory that not only optimized the key parameter of interest, Information Gain, but also satisfied all constraints imposed on the trajectory, such as kinematic and obstacle avoidance constraints. However, the stated goal of generating these optimal trajectories in real time may be jeopardized by the computational intensity of the algorithm. As discussed in Section 5.4, just a few MATLAB routines occupied a significant share of the computation time. These subroutines are key to the sonar imaging, probabilistic analysis of information gain, and obstacle avoidance. As each of these routines performs numerous calculations sequentially on individual cells of large matrices, some computational rigor is expected. An important aspect to consider, however, is that much of the intensive computation being performed by MATLAB in the simulation environment will actually be done by dedicated hardware on the REMUS vehicle. For example, the sonar imaging that consumed so much computation time in MATLAB will be performed by the Blueview FLS and it's dedicated image processing hardware. This alone would drastically reduce the load on the algorithm itself. Similar gains would likely be achieved by other aspects of the hardware on the vehicle, especially given that compiled code runs orders of magnitude faster than MATLAB interpreted code.

A key observation made during the algorithm test and evaluation phase of this thesis was the sensitivity of the trajectory to the adjustments made to the weighting coefficients and variable parameter initial guesses. Many adjustments had to be made to several of these parameters in order to generate physically valid trajectories for each scenario. In some cases, adjusting a weighting coefficient to minimize one penalty resulted in an increase in another penalty. An example of this is when adjustments made to reduce the β violation penalty resulted in higher yaw rates, increasing the $\dot{\psi}_{max}$ violation. Further thesis research is required to investigate the interplay between, and sensitivity of, each of the variable parameters and weighting coefficients.

Another interesting observation is that the generated trajectories were often not intuitive in that they were frequently non-linear and deviated significantly from the relatively simple trajectories expected for reactive obstacle avoidance. As discussed in the thesis, the DM-IDVD routine

67

generates full sets of candidate trajectories by varying kinematic parameters at the boundaries of the trajectory, such as speed, acceleration, etc. This feature, combined with the fact that the coordinates, yaw angle, and speed factor of the trajectories are represented by reference functions combining high order polynomials and trigonometric functions, produces these that are far more flexible and complex, both spatially and kinematically.

In conclusion, this thesis successfully proved the concept of utilizing the Direct Method of Inverse Dynamics in the Virtual Domain to calculate information-optimal trajectories based on a probabilistic analysis of information gain. However, the viability of utilizing this method for *real-time* trajectory generation was not established. Time constraints precluded further code optimization and compilation into executable code to be run on the vehicle, and so in-tank testing on the REMUS vehicle itself was never achieved.

6.2   Recommendations for Future Work

As discussed in the conclusions, there may well be significant opportunity for performance improvements by optimizing the MATLAB code used to program the routine. Careful scrutiny might be given to specific functions and subroutines within the algorithm to determine where and how code optimizations may be made to improve the efficiency of the routine. Hardware-in-the-loop simulation may also significantly speed up the computations in the algorithm as well. Specifically, the author recommends examining alternative methods for the following subroutines/functions: ray tracing (performed in both the sonar imaging and allision avoidance subroutines); occupancy grid cell probabilistic calculations (specifically where the sensor signal probabilities are evaluated from the sensor model probability density functions); and symbolic math manipulations, specifically wherever the MATLAB *polyval()* functions is utilized.

Following any required code optimizations, the obvious next step should be to compile the algorithm into computer code compatible with the computer resources on the REMUS vehicle and perform actual in-tank testing in the CAVR AUV test tank. Following the completion of successful in-tank testing, open water testing would then be needed to ascertain the viability of the routine in the presence of disturbances.

Concomitant with on-vehicle testing of the algorithm would be implementation of methods required to reduce real-world positional uncertainty caused by sleep-state drift and inherent sensor error and noise. As methods for performing this task, such as Kalman filters, are already

well known and developed, augmenting the algorithm developed by this thesis with a positional uncertainty reduction scheme should be fairly trivial.

THIS PAGE INTENTIONALLY LEFT BLANK

REFERENCES

[1] P. S. Sochaczewski and J. Hyvarinen, "Down deep: Environmentalists fight to protect the fantastic microscopic creatures that dwell on the oceans bottom," *E : the Environmental Magazine*, vol. 7, no. 4, pp. 15–15, 1996.

[2] T. Allen, J. T. Conway, and G. Roughead, "A cooperative strategy for 21st century seapower," *United States Naval Institute Proceedings*, vol. 133, no. 11, pp. 14–20, 2007.

[3] J. A. Walsh and R. M. Smith, *Navy Unmanned Undersea Vehicle (UUV) Master Plan*. NUWC, 2004.

[4] T. B. Curtin, D. M. Crimmins, J. Curcio, M. Benjamin, and C. Roper, "Autonomous underwater vehicles: Trends and transformations," *Marine Technology Society Journal*, vol. 39, no. 3, pp. 65–75, 2005.

[5] D. Horner and O. Yakimenko, "Recent developments for an obstacle avoidance system for a small auv," in *IFAC Proceedings Volumes*, vol. 7. IFAC, 2007, pp. 19–25.

[6] A. R. V. Reet, "Contour tracking control for the remus autonomous underwater vehicle," Master's thesis, Naval Postgraduate School, June 2005.

[7] D. L. Hemminger, "Vertical plane obstacle avoidance and control of the remus autonomous underwater vehicle using forward looking sonar," Master's thesis, Naval Postgraduate School, June 2005.

[8] A. Healey, "Guidance laws, obstacle avoidance, artificial potential functions," in *Advances in Unmanned Marine Vehicles*, G. Roberts and R. Sutton, Eds. IEEE, 2006, ch. 2.

[9] T. H. Furukawa, "Reactive obstacle avoidance for the remus autonomous underwater vehicle utilizing a forward looking sonar," Master's thesis, Naval Postgraduate School, June 2006.

[10] O. Yakimenko, "Real-time computation of spatial and flat obstacle avoidance trajectories for uuv's," in *Navigation, Guidance and Control of Underwater Vehicles*, vol. 2. IFAC, 2008.

[11] O. Yakimenko, D. Horner, and D. Pratt, "Auv rendezvous trajectories generation for underwater recovery," in *Control and Automation, 2008 16th Mediterranean Conference on*, June 2008, pp. 1192–1197.

[12] N. A. McChesney, "Three-dimensional feature reconstruction with dual forward looking sonars for unmanned underwater vehicle navigation," Master's thesis, Naval Postgraduate School, 2009.

[13] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer*, vol. 22, no. 6, pp. 46 –57, jun 1989.

[14] S. Noykov and C. Roumenin, "Occupancy grids building by sonar and mobile robot," *Robotics and Autonomous Systems*, vol. 55, no. 2, pp. 162 – 175, 2007.

[15] D. S. Levine, "Information-rich path planning under general constraints using rapidly-exploring random trees," Master's thesis, Massachusetts Institute of Technology, June 2010.

[16] W. B. Sebastian Thrun and D. Fox, *Probabilistic Robotics*.   MIT Press, 2005.

[17] D. J. C. MacKay, *Information Theory, Inference, and Learning Algorithms*.   Cambridge University Press, 2003.

[18] O. Yakimenko, "Direct method for rapid prototyping of near-optimal aircraft trajectories," *Journal of Guidance, Control, and Dynamics*, vol. 23, no. 5, pp. 865–875, September-October 2000.

[19] O. Yakimenko and S. Kragelund, "Real-time optimal guidance and obstacle avoidance for umvs," in *Autonomous Underwater Vehicles*, N. A. Cruz, Ed.   InTech, 2011, ch. 4.

[20] O. Yakimenko, "Time and space decoupling," http://faculty.nps.edu/oayakime/AE4902/IME/4a/decoupling.html.

[21] N. Slegers and O. A. Yakimenko, "Terminal guidance of autonomous parafoils in high wind-to-airspeed ratios," *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, vol. 225, no. 3, pp. 336–346, 2011.

[22] S. M. Doherty, "Cross body thruster control and modeling of a body of revolution autonomous underwater vehicle," Master's thesis, Naval Postgraduate School, 2011.

APPENDIX A:
MATLAB Code

A.1   Main Function: *Main_opt.m*

```matlab
%***********************************************************************
% MAIN_OPT.M
% THIS IS THE MAIN SCRIPT. RUN THIS TO PERFORM THE FULL SIMULATION.
%
% Written by: LT Adam Wiseman
% All code was written by the author with the exception of bresenham.m and
% the code for generating the animated plot. Credits for bresenham.m are
% inside that script, and credit for the animated plot code and the
% associated scripts/functions code goes to Dr. Les Carr of the Math Dept.
% at NPS.
%
% Last Edited: 23 March 2012
%***********************************************************************

%% Prep——clear workspace, declare global variables
clear all
close all
clc
profile on —history

global L D W H R psi_dot_max u_max v_max dt TH M N
global xinit yinit psii beta_max %psif
global OG SnrImg
global AnimFlag ObsFlag ProbFlag x y VARS

%% Switches——turn certain features on/off
AnimFlag=1;      % Set to 0 to disable display of the REMUS animation
ProbFlag=1;      % Set to 0 to run routine without probabilistic analysis
                 % (speeds up trajectory validation process)
ObsFlag=1;       % Set to 0 to disable obstacles in tank

%% Vehicle and Tank Dimensions
L=2.75;  % vehicle length in meters
```

```matlab
34  D=0.19;   % vehicle diameter in meters
35  W=20;     % tank length in meters
36  H=10;     % tank width in meters
37
38  R=2*H;        % This is used to control size of tank matrix oversize and
39                % sonar beam range. It was originally set to the
40                % range of the sonar, but this resulted in overly large data
41                % matrices, and was thus reduced in length, with no detriment
42                % to the overall routine within the constraints of the test
43                % tank dimensions. Future testing in larger environments will
44                % require this to be set back to the sonar range.
45
46  M=10*(2*R+H);  % Row space for model grids/matrices
47  N=10*(2*R+W);  % Column space for model grids/matrices
48  % Note:  The multiplication factor of 10 allows for finer grids in the
49  % sonar image and occupancy grid, and minimizes rounding errors caused when
50  % rounding non-integer coordinate values to integer values for matrix cell
51  % addressing. This multiplication factor is also found in some of the
52  % subroutines that require coordinate-to-matrix cell index translation.
53
54  %% Initialization of parameters
55  %***** Time Horizon (s) *****
56  TH=15;  % Nominal time of trajectory from start point to end point.
57  %***************************
58  dt=.1;          % Time step (s)
59  rf=5;           % Refinement factor for refining data mesh. This simply
60                  % decreases the step size of each maneuver.  Raise this
61                  % value to reduce the step size.
62  IG=[];          % Initialize Information Gain (for computational speed)
63
64  % Constraints
65  dp=14.5;                   % Max yaw rate (deg/sec)
66  psiTH=TH*dp;               % Max yaw over time horizon (deg)
67  psi_dot_max = dp*pi/180;   % maximum yaw rate, rad/s
68  v_max = .5;                % maximum sway velocity, m/s
69  u_max = 2.88;              % surge velocity in m/s
70  beta_max=45*pi/180;        % absolute value of max desired beta
71                             % (to keep trajectory within sonar cone)
72
73  %****INITIAL PARAMETER VALUES!!!****
74  xinit=W-3*L/2;             % x-coordinate of origin of distribution circle
```

```matlab
75      % xinit=190;             % For testing purposes only
76 yinit=H-3*L/2;              % y-coordinate of origin of distribution circle
77      % yinit=80;              % For testing purposes only
78
79 psiinitdeg = 0;            % Initial Heading in degrees
80 psiinit = pi/180*psiinitdeg;% Initial Heading in radians
81
82 %****FINAL PARAMETER VALUES!!!****
83 xfin=3;
84 yfin=5;
85 psi_d=-45;                 % Final yaw angle in degrees
86 psifin=psi_d*pi/180;       % Final yaw angle in radians
87
88 %% Heuristic Boundaries
89 % Note:  HB are completely artificially generated here.  In the future, it
90 % will most likely be necessary to generate these on the fly based on
91 % initial sonar sweeps.
92
93 if ProbFlag
94     global BB UB RB LB RBx LBx UBy BBy
95
96     % Upper Heuristic Bound (top wall)
97     UB=R;
98
99     % Lower (Bottom) Heuristic Bound
100    BBy=yinit-L/2;
101    BB=H+R-BBy;
102
103    % Right Heuristic Bound (Right wall)
104    RB=W+R;
105
106    % Left Heuristic Bound
107    LBx=xinit-L/2;
108    LB=LBx+R;
109
110    % Bounding box corners for plotting purposes
111    UBy=H+R-UB;
112    RBx=RB-R;
113    xHCorners=[LBx LBx RBx RBx LBx];
114    yHCorners=[BBy UBy UBy BBy BBy];
115 end
```

```matlab
%% Initialize Sensor model
if ProbFlag
    SensorModelGeneration;
end

%% Object
global xl xr yl yu xl2 xr2 yl2 yu2
% Object 1
xl=6;        % Left boundary of object 1
xr=7;        % Right boundary of object 1
yl=7;        % Lower boundary of object 1
yu=8;        % Upper boundary of object 1

% Object 2
xl2=8;       % Left boundary of object 2
xr2=9;       % Right boundary of object 2
yl2=2;       % Lower boundary of object 2
yu2=3;       % Upper boundary of object 2

%% DIRECT METHODS OPTIMIZATION (IDVD)
%*************************************************************************
% Everthing below this is part of the DM process.
%*************************************************************************

%% Defining optimization problem
global Fine_v Cost_T Fine_YawRate Fine_u Fine_beta
global wT wY wu wv wb

wT  = 1e0;              % weighting coefficient for time of arrival
wY  = 1e2;             % weighting coefficient for yaw rate
wu  = 1e0;              % weighting coefficient for surge velocity u
wv  = 1e3;             % weighting coefficient for sway velocity v
wb  = 1e0;              % weighting coefficient for beta

if ProbFlag
    global wG wH wA wE Cost_EV Cost_H Cost_IG Cost_AP avrad
    wG  = 1e1;          % weighting coefficient for Information Gain
    wH  = 1e0;          % weighting coefficient for Heuristics Violations
    wA  = 1e0;          % weighting coefficient for allision avoidance
    wE  = 1e1;          % weighting coefficient for unexplored region penalty
```

```matlab
157
158     avrad=L;              % Obstacle Allision Avoidance radius
159  end
160
161  %% Setting the boundary conditions
162  global vxi posxi vxf posxf
163  global vyi posyi vyf posyf lami lamf
164
165  posxi = xinit;   posyi = yinit;    % initial position
166  posxf = xfin;    posyf = yfin;     % final position
167  vxi = 0;         vyi = 0;          % components of initial velocity
168  vxf = 0;         vyf = 0;          % components of final velocity
169  % accxi = 0;       accyi = 0;       % components of initial acceleration
170  % accxf = 0;       accyf = 0;       % components of final acceleration
171  % betai=atan2(vxi,vyi)-psiinit; betaf=atan2(vxf,vyf)-psifin;
172  psii=psiinit;   % psif=psifin;     % initial and final yaw angles
173  lami=1;          lamf=1;           % initial and final lambdas
174
175  %% Guessing on the varied parameters
176
177  guess(1)=15;              % virtual arc length (tauf)
178  guess(2)=0.5;             % magnitude of final accel (accxf)
179  guess(3)=-0*pi/4;         % direction of final accel (accyf)
180  guess(4)=0.25;            % magnitude of initial accel (accxf)
181  guess(5)=3*pi/4;          % direction of initial accel (accyf)
182  guess(6)=-.1;             % initial psi double prime (psippi)
183  guess(7)=-0.054;          % final psi double prime (psippf)
184  guess(8)=.0017;           % initial lambda-dbl-prime
185  guess(9)=.0009;           % final lambda-dbl-prime
186  guess(10)=-pi/2;          % final psi
187
188  %% Define the Reference Function Coefficients
189  RefFuncs;     %Note:  Doing this the other way by including the coefficient
190               %vectorss in the trajectory function causes an error.  Keeping
191               %it this way for now. However, some computational optimization
192               %may be acheived by including the output of this function
193               %directly into the trajectory.m script to eliminate the
194               %symbolic math computations performed in this function. The
195               %coefficient vectors have been left in trajectory.m but
196               %commented out to allow for future implementation. If the
197               %other method is used, this line may be commented out.
```

```matlab
198
199  %% Calling the optimization routine
200  maxiter=1000;          % maximum number of iterations within fminsearch
201  maxfuns=5*maxiter;     % maximum number of function evaluations
202  % Setting the options for fminsearch:
203  opt=optimset('Display','iter','TolX',1e-2,'TolFun',1e-2,...
204                              'MaxIter',maxiter,'MaxFunEvals',maxfuns);
205  t = cputime;    % Used to calculate and display actual computational time
206
207  %***********************************************************************
208  % Calling the fiminsearch optimization function:
209  [guess_opt,fval,exitflag] = fminsearch('trajectory',guess,opt);
210  %***********************************************************************
211
212  % calculating and displaying computational time:
213  time_elapsed = cputime-t;
214  fprintf('fminsearch Elapsed Time: %4.2g minutes.\n\n',time_elapsed/60)
215  profile off
216
217  %% Displaying cost functions and penalties
218
219  if ProbFlag
220      fprintf(' Information Gain Cost function      :  %6.2g\n',Cost_IG)
221      fprintf(' Heuristics Violation Cost function     :  %6.2g\n',Cost_H)
222      fprintf(' Allision Avoidance Cost function      :  %6.2g\n',Cost_AP)
223      fprintf(' Unexplored Territory Cost function     :  %6.2g\n',Cost_EV)
224  end
225  fprintf(' Time Cost function      :  %6.2g\n',Cost_T)
226  fprintf(' Penalty in sway v    :  %6.2g\n',Fine_v)
227  fprintf(' Penalty in surge u   :  %6.2g\n',Fine_u)
228  fprintf(' Penalty in YawRate      :   %6.2g\n',Fine_YawRate)
229  fprintf(' Penalty in beta      :   %6.2g\n\n',Fine_beta)
230
231  %% Displaying optimal parameters
232
233  % Display initial guesses on varied parameters
234  disp('Initial Guesses on Values of Varied Parameters:')
235  fprintf('Arc length = %6.2f\n',guess(1))
236  fprintf('Final x,y-accel magnitude = %6.2f\n',guess(2))
237  fprintf('Final x.y-accel direction = %6.2f deg\n',guess(3)*180/pi)
238  fprintf('Initial x,y-accel magnitude = %6.2f\n',guess(4))
```

```
239  fprintf('Initial x,y-accel direction = %6.2f deg\n',guess(5)*180/pi)
240  fprintf('Initial psi accel = %6.2f\n',guess(6))
241  fprintf('Final psi accel = %6.2f\n',guess(7))
242  fprintf('Initial lambda dbl prime = %6.2f\n',guess(8))
243  fprintf('Final lambda dbl prime = %6.2f\n',guess(9))
244  fprintf('Final psi = %6.2f\n\n',guess(10)*180/pi)
245
246  % Display final values of varied parameters
247  disp('Final Values of Varied Parameters:')
248  fprintf('Arc length = %6.2f\n',guess_opt(1))
249  fprintf('Final x,y-accel magnitude = %6.2f\n',guess_opt(2))
250  fprintf('Final x.y-accel direction = %6.2f deg\n',guess_opt(3)*180/pi)
251  fprintf('Initial x,y-accel magnitude = %6.2f\n',guess_opt(4))
252  fprintf('Initial x,y-accel direction = %6.2f deg\n',guess_opt(5)*180/pi)
253  fprintf('Initial psi accel = %6.2f\n',guess_opt(6))
254  fprintf('Final psi accel = %6.2f\n',guess_opt(7))
255  fprintf('Initial lambda dbl prime = %6.2f\n',guess_opt(8))
256  fprintf('Final lambda dbl prime = %6.2f\n',guess_opt(9))
257  fprintf('Final psi = %6.2f\n\n',guess_opt(10)*180/pi)
258
259  %% Plotting the results
260  PlotResults
261
262  %% END
263  profile off
264  % The following line saves all variables in the workspace to a .mat file
265  % for future reference. Change this each run to avoid overwriting
266  % previous saves
267
268  % save('Run_3-22_1_incwv_facingN-fminsearch-1000iter')
```

A.2   Trajectory Generation and Optimization Function: *trajectory.m*

```
1  function PI=trajectory(guess)
2  %% This function computes states and controls for the current guess.
3  % It is the input function for fminsearch, and cannot be run alone. It is
4  % the function to be optimized by fminsearch.
5  global VARS ProbFlag
6  global vxi posxi vxf posxf lami lamf
7  global vyi posyi vyf posyf psii % psif betai betaf
```

```matlab
 8  global x y time V u v psi Psi tau lam beta psi_dot
 9
10  %% Initialize Matrices/Grids——see notes in script
11
12  InitMatrices;
13
14  %% Current values of varied parameters
15  tauf   = guess(1);                   % virtual arc length
16  accxf  = guess(2)*cos(guess(3));     % final x accel
17  accyf  = guess(2)*sin(guess(3));     % final y accel
18  accxi  = guess(4)*cos(guess(5));     % initial x accel
19  accyi  = guess(4)*sin(guess(5));     % initial y accel
20  psippi = guess(6);                   % initial beta double prime
21  psippf = guess(7);                   % final beta double prime
22  lamppi = guess(8);                   % initial lambda—dbl—prime
23  lamppf = guess(9);                   % final lambda—dbl—prime
24  psif   = guess(10);                  % final psi
25
26  %% Defining parameters in M/NN nodes in the virtual domain
27
28  global a apsi alam Mp Np Ol
29  % (x,y) Reference Function Coefficients
30
31  % Note:  Putting the coefficient matrices in here gives me an error, so I'm
32  % keeping the RefFuncs subroutine for now. If the errors can be resolved,
33  % it would be computationally quicker to eliminate the symbolic math
34  % computations in RefFuncs and simply use the coefficient vectors below.
35
36  % syms tf x0 xp0 xpp0 xf xpf xppf
37  % a = [x0;
38  %      (—xpp0/3 — xppf/6)*tf^2 — x0 + xf;
39  %      (tf^2*xpp0)/2;
40  %      —tf^2*(xpp0/6 — xppf/6);
41  %      ((xpp0 + xppf)*tf^2)/(4*pi) + ((2*xp0 — 2*xpf)*tf)/(4*pi);
42  %      ((xpp0 — xppf)*tf^2)/(24*pi) + ((6*xp0 + 6*xpf)*tf)/(24*pi) + ...
         (12*x0 — 12*xf)/(24*pi)];
43  %
44  ax=subs(a,{'x0','xp0','xpp0','xf','xpf','xppf','tf'},...
45                                  {posxi,vxi,accxi,posxf,vxf,accxf,tauf});
46  ay=subs(a,{'x0','xp0','xpp0','xf','xpf','xppf','tf'},...
47                                  {posyi,vyi,accyi,posyf,vyf,accyf,tauf});
```

```
48
49  % (psi) Reference Function Coefficients
50  % syms psi0 psip0 psipp0 psiff psipf psippff tf
51  % apsi=[psi0
52  %          psip0*tf;
53  %          (psipp0*tf^2)/2;
54  %          (psippff/2 — (3*psipp0)/2)*tf^2 + (—6*psip0 — 4*psipf)*tf — ...
       10*psi0 + 10*psiff;
55  %          ((3*psipp0)/2 — psippff)*tf^2 + (8*psip0 + 7*psipf)*tf + ...
       15*psi0 — 15*psiff;
56  %          (psippff/2 — psipp0/2)*tf^2 + (—3*psip0 — 3*psipf)*tf — 6*psi0 ...
       + 6*psiff];
57
58  ap=subs(apsi,{'psi0','psip0','psipp0','psiff','psipf','psippff','tf'},...
59                                      {psii,0,psippi,psif,0,psippf,tauf});
60
61  % (lambda) Reference Function Coeffiecients
62  % syms lam0 lamp0 lampp0 lamff lampff lamppff tf
63  % alam=[lam0;
64  %          lamp0*tf;
65  %          (lampp0*tf^2)/2;
66  %          (lamppff/2 — (3*lampp0)/2)*tf^2 + (— 6*lamp0 — 4*lampff)*tf — ...
       10*lam0 + 10*lamff;
67  %          ((3*lampp0)/2 — lamppff)*tf^2 + (8*lamp0 + 7*lampff)*tf + ...
       15*lam0 — 15*lamff;
68  %          (lamppff/2 — lampp0/2)*tf^2 + (— 3*lamp0 — 3*lampff)*tf — ...
       6*lam0 + 6*lamff];
69
70  al=subs(alam,{'lam0','lamp0','lampp0','lamff','lampff','lamppff','tf'},...
71                                      {lami,0,lamppi,lamf,0,lamppf,tauf});
72
73  % If using the commented out coefficient vectors above, uncomment the
74  % following lines as well:
75
76  % Mp=length(a);
77  % Nb=length(apsi);
78  % Ol=length(alam);
79
80  for i=1:Mp—2
81      cx(i)=ax(Mp—1—i);
82      cy(i)=ay(Mp—1—i);
```

81

```matlab
83  end
84  for i=1:Np
85      cp(i)=ap(Np+1-i);
86  end
87  for i=1:Ol
88      cl(i)=al(Ol+1-i);
89  end
90
91  taubar=linspace(0,1); % taubar=tau/tauf (See report for why)
92
93  % Evaluate complete ref funcs to obtain vectors of state parameters:
94  x=polyval(cx,taubar)+ax(5)*sin(pi*taubar)+ax(6)*sin(2*pi*taubar);
95  y=polyval(cy,taubar)+ay(5)*sin(pi*taubar)+ay(6)*sin(2*pi*taubar);
96  psi=polyval(cp,taubar);
97  lam=polyval(cl,taubar);
98
99  NN=length(x);
100 %% Defining parameters' derivatives in NN nodes in the virtual domain
101 cx_prime=cx.*[3:-1:0]*eye(4,3);
102 cy_prime=cy.*[3:-1:0]*eye(4,3);
103
104     x_prime=polyval(cx_prime,taubar)+ax(5)*pi*cos(pi*taubar)+ax(6)*2*pi*...
105                                                 cos(2*pi*taubar);
106     y_prime=polyval(cy_prime,taubar)+ay(5)*pi*cos(pi*taubar)+ay(6)*2*pi*...
107                                                 cos(2*pi*taubar);
108     x_prime=x_prime/tauf;
109     y_prime=y_prime/tauf;
110
111 cx_dblprime=cx.*[6 2 0 0]*eye(4,2);
112 cy_dblprime=cy.*[6 2 0 0]*eye(4,2);
113
114     x_dblprime=polyval(cx_dblprime,taubar)-ax(5)*pi^2*sin(pi*taubar)-...
115                                             ax(6)*(2*pi)^2*sin(2*pi*taubar);
116     y_dblprime=polyval(cy_dblprime,taubar)-ay(5)*pi^2*sin(pi*taubar)-...
117                                             ay(6)*(2*pi)^2*sin(2*pi*taubar);
118     x_dblprime=x_dblprime/tauf^2;
119     y_dblprime=y_dblprime/tauf^2;
120
121 cp_prime=cp.*[5:-1:0]*eye(6,5);
122
123     psi_prime=polyval(cp_prime,taubar);
```

```matlab
124    psi_prime=psi_prime/tauf;
125
126 cl_prime=cl.*[5:-1:0]*eye(6,5);
127
128    lam_prime=polyval(cl_prime,taubar);
129    lam_prime=lam_prime/tauf;
130
131 %% Computing the states and controls using *Inverse Dynamics*
132
133 del_tau = tauf/(NN-1);
134 tau(1)  = 0;
135 time(1) = 0;
136 Psi     = atan2(x_prime,y_prime);    % Computing heading, rad
137 Psi(1)  = Psi(2); Psi(end)=Psi(end-1);
138 beta=Psi-psi;
139 V(1)    = lam(1)*sqrt(x_prime(1)^2+y_prime(1)^2);    % Total speed
140 u(1)    = V(1)*cos(beta(1));                         % surge velocity
141 v(1)    = V(1)*sin(beta(1));                         % sway velocity
142
143 Psi_prime(1) = (x_prime(1)*y_dblprime(1)-y_prime(1)*...
144                             x_dblprime(1))/(y_prime(1)^2+x_prime(1)^2);
145 Psi_dot(1) = lam(1)*Psi_prime(1);
146 psi_dot(1) = lam(1)*psi_prime(1);
147
148 for j=2:NN
149     tau(j)  = tau(j-1)+del_tau;
150     dt      = 2*del_tau/(lam(j-1)+lam(j));
151     time(j) = time(j-1)+dt;
152     V(j)= lam(j)*sqrt(x_prime(j)^2+y_prime(j)^2);    % Total speed
153     u(j)    = V(j)*cos(beta(j));                     % surge velocity
154     v(j)    = V(j)*sin(beta(j));                     % sway velocity
155     Psi_prime(j)=(x_prime(j)*y_dblprime(j)-y_prime(j)*...
156                             x_dblprime(j))/(y_prime(j)^2+x_prime(j)^2);
157     Psi_dot(j)=lam(j)*Psi_prime(j);
158     psi_dot(j) = lam(j)*psi_prime(j);
159 end
160
161 Psi_dot(end)=Psi_dot(end-1);
162 beta_dot = Psi_dot - psi_dot;
163
164 %
```

```matlab
165 VARS=[lam' tau' time' x' y' u' v' V' x_prime' y_prime' beta' beta_dot'...
166                                        psi' psi_dot' Psi' Psi_dot'];
167
168 %%Sonar Imaging, Probabilistic Analysis, Occupancy Grid, Obstacle Avoidance
169 if ProbFlag
170
171     %*************************************************************************
172     % Stuff below this is part of the probabilistic analysis process.
173     %*************************************************************************
174
175     global OG SnrImg PrOpoly PrEpoly
176     global L IG UB BB RB LB AP RBx BBy ViolArea avrad
177
178     %% Information Gain Analysis
179
180     for i=1:NN
181         % Perform simulated sonar sweep
182         sweep(psi(i),x(i),y(i));
183
184         % Weight OG values at heuristic boundaries—this reduces the
185         % information gain quantified by objects that we are already
186         % assuming to be there.
187         if i == 1
188             OG(10*UB,10*LB:10*RB)=.75;
189             OG(10*UB:10*BB,10*RB)=.75;
190         end
191     end
192     %% Update Occupancy Grid and Calculate Information Gain
193
194         [IGC1]=OGupdate(SnrImg,PrOpoly,PrEpoly);  % IGC1: IG Calculation
195         IG=IGC1;
196
197     %% Stern Points used for Heuristic Checks
198     % Calculate the coordinates of the stern of the vehicle at each point
199     % along the trajectory and determine if the stern points cross the
200     % heuristics boundaries.
201
202     for i=1:NN
203             xS(i)=x(i)—L/2*sin(psi(i)); % Calc x—coord of stern for ...
204                 each x_cg
                if xS(i) > RBx
```

```
205                    xViol(i)=1;
206            else
207                    xViol(i)=0;
208            end
209            yS(i)=y(i)-L/2*cos(psi(i)); % Calc y-coord of stern for ...
                   each y_cg
210            if yS(i) < BBy
211                    yViol(i)=1;
212            else
213                    yViol(i)=0;
214            end
215    end
216
217    %% Heuristic Violation Area Analysis
218    % Uses the vectors of stern point coordinates determined above to
219    % calculate the total violation area between the arc swept by the
220    % vehicle stern and the heuristics boundaries.
221
222        ViolArea=HeurViolArea(xS(:),yS(:),xViol(:),yViol(:));
223
224 %*****END PROBABILISTIC (INFORMATION GAIN) ANALYSIS STUFF*****************
225 %***********************************************************************
226    %% Allision Avoidance
227    % Calculate the amount of penetration of an object/obstacle within a
228    % circle centered on the center of the vehicle with a give avoidance
229    % radius (defined in Main_opt.m script).
230
231    AvoidancePenalty=zeros(NN,1); % Initialize vector (for speed)
232
233    for i=1:NN
234        AvoidancePenalty(i)=Avoidance(beta(i),x(i),y(i),avrad);
235    end
236
237    % Cumulative avoidance penalty along trajectory:
238    AP=sum(AvoidancePenalty);
239    %% Favoring Explored Areas
240    % Calculate penalty for trajectorys that drive the vehicle outside of
241    % areas already swept by sonar (i.e. "explored areas"). This biases the
242    % resulting trajectories into known, or "explored" space.
243    global EV
244    for i=1:NN
```

```
245     [me(i),ne(i)]=xy2mn(x(i),y(i));  % Convert x,y coords to matrix coords
246     OGcval=OG(round(10*me(i)),round(10*ne(i)));   % Determine OG cell value
247                                                   % at each x,y
248     ev(i)=abs(OGcval-0.5);           % Calculate absolute difference between
249                                      % OGVal and the OG value for an
250                                      % unexplored cell (0.5).
251     end
252     EV=sum(ev);
253 end
254 %% Calculate Performance Index
255 PI = PerformanceIndex;
256 return
257
258 function PI=PerformanceIndex
259 %% This function computes the combined performance index
260 %************************************************************************
261 %  This function performs the actual calculations of the weighted penalties
262 %  for the Cost Function and Performance Index. The output is the value
263 %  that is actually being optimized within the fminsearch function by
264 %  changing the values of the variable parameters.
265 %************************************************************************
266
267 global wu wv  wb wY ProbFlag TH wT time Cost_T
268 global u v u_max v_max  psi_dot psi_dot_max
269 global Fine_u Fine_v Fine_YawRate
270 global Fine_beta beta beta_max
271
272     Cost_T           = (time(end)-TH)^2;
273     Fine_YawRate     = max([0,(abs(psi_dot)-psi_dot_max)])^2;
274     Fine_u           = max([0,(abs(u)-u_max)])^2;
275     Fine_v           = max([0,(abs(v)-v_max)])^2;
276     Fine_beta        = max([0,(abs(beta)-beta_max)])^2;
277     if ProbFlag
278         global Cost_IG Cost_H Cost_AP Cost_EV  wG wH wA wE
279         global ViolArea AP IG EV
280         Cost_H           = ViolArea;
281         Cost_AP          = AP;
282         Cost_IG          = 1/IG;
283         Cost_EV          = 1/(1+EV);
284     end
285
```

```matlab
286    if ProbFlag
287        PI = wG*Cost_IG+wH*Cost_H+wA*Cost_AP+wu*Fine_u+wv*Fine_v+wY*...
288                              Fine_YawRate+wE*Cost_EV+wT*Cost_T;
289    else
290        PI = wu*Fine_u+wv*Fine_v+wY*Fine_YawRate+wb*Fine_beta+wT*Cost_T;
291    end
292
293 return
```

A.3   Reference Function Generation Function: *RefFuncs.m*

```matlab
1  % This script computes coefficients of the reference polynomials
2  %% (x,y) Reference Function (scaled tau_bar=tau/tauf)
3  global a apsi alam Mp Np Ol
4  syms tf x0 xp0 xpp0 xf xpf xppf
5  A= [1 0 0 0  0   0;
6       0 1 0 0  pi  2*pi;
7       0 0 2 0  0   0;
8       1 1 1 1  0   0;
9       0 1 2 3  -pi 2*pi;
10      0 0 2 6  0   0];
11 b= [x0 (xp0*tf) (xpp0*tf^2) xf (xpf*tf) (xppf*tf^2)].';
12 a=A\b;
13 a=collect(a,tf);
14 Mp=length(a);
15
16 %% (psi) Reference Function (scaled tau_bar=tau/tauf)
17 syms psi0 psip0 psipp0 psiff psipf psippff tf
18 C=[1    0    0    0    0    0;
19    0    1    0    0    0    0;
20    0    0    2    0    0    0;
21    1    1    1    1    1    1;
22    0    1    2    3    4    5;
23    0    0    2    6    12   20];
24 d=[psi0 psip0*tf psipp0*tf^2 psiff psipf*tf psippff*tf^2].';
25 apsi=C\d;
26 apsi=collect(apsi,tf);
27 Np=length(apsi);
28
29 %% (speed profile) Reference Function\
```

```
30
31 syms lam0 lamp0 lampp0 lamff lampff lamppff tf
32 LL=[ 1 0 0 0 0 0;
33      0 1 0 0 0 0;
34      0 0 2 0 0 0;
35      1 1 1 1 1 1;
36      0 1 2 3 4 5;
37      0 0 2 6 12 20];
38 l=[lam0 lamp0*tf lampp0*tf^2 lamff lampff*tf lamppff*tf^2].';
39 alam=LL\l;
40 alam=collect(alam,tf);
41 Ol=length(alam);
```

A.4   Sensor Model Generator Function: *SensorModelGeneration.m*

```
1  % Sonar Sensor Model:
2  % This function creates the probability density functions for the sonar
3  % sensor model. It takes the pre—fit curve fits stored in this folder
4  % (created using the cfit Matlab toolbox) and creates the polynomials
5  % representing each sonar sensor probability model pdf for occupied/empty
6  % states. The empirical data input in the sensor probabilities cells (rO
7  % and rE) represents the data used to create the pdf's and resulting curve
8  % fits. This data is not actually used during runtime due to the fact that
9  % it is already incorporated into the saved curve fits PrOmdlfunc and
10 % PrEmdlfunc.
11
12 % The pdf's may be plotted by uncommenting the plotting code in the last
13 % cell.
14
15 global PrOpoly PrEpoly
16
17 % Signal strengths (x—axis)
18 SS=linspace(0,5000);
19 SS=SS';
20
21 %% Sensor Probabilities (Occupied)
22
23 rO = [  0;
24         0.5000;
25         1.0000;
```

```
26        1.5000;
27        2.0000;
28        2.5000;
29        3.0000;
30        3.5000;
31        4.0000;
32        4.5000;
33        5.0000;
34        5.5000;
35        6.0000;
36        6.5000;
37        7.0000;
38        7.5000;
39        8.0000;
40        8.5000;
41        9.0000;
42        9.5000;
43       10.0000;
44       10.5000;
45       11.0000;
46       11.5000;
47       12.0000;
48       12.5000;
49       13.0000;
50       13.5000;
51       14.0000;
52       14.5000;
53       15.0000;
54       15.5000;
55       16.0000;
56       16.5000;
57       17.0000;
58       17.5000;
59       18.0000;
60       18.5000;
61       19.0000;
62       19.5000;
63       20.0000;
64       20.5000;
65       21.1000;
66       21.8000;
```

67	22.6000;
68	23.5000;
69	24.5000;
70	25.6000;
71	26.8000;
72	28.1000;
73	29.5000;
74	31.0000;
75	32.6000;
76	34.3000;
77	36.1000;
78	38.0000;
79	40.0000;
80	42.1000;
81	44.3000;
82	46.6000;
83	49.0000;
84	51.5000;
85	53.9500;
86	56.3500;
87	58.7000;
88	61.0000;
89	63.2500;
90	65.4500;
91	67.6000;
92	69.7000;
93	71.7500;
94	73.7500;
95	75.7000;
96	77.6000;
97	79.4500;
98	81.2500;
99	83.0000;
100	84.7000;
101	86.3500;
102	87.9500;
103	89.5000;
104	91.0000;
105	92.4500;
106	93.8500;
107	95.2000;

```
108          96.5000;
109          97.7500;
110          98.9500;
111         100.1000;
112         101.2000;
113         102.2500;
114         103.2500;
115         104.2000;
116         105.1000;
117         105.9500;
118         106.7500;
119         107.5000;
120         108.2000;
121         108.8500;
122         109.4500];
123
124 % Normalize rO Vector (ensure sum(rO) = 1)
125 rO=rO./sum(rO);
126
127 % determine function representing this pdf
128 load PrOmdlfunc;
129 PrOpdf=PrOmdlfunc;
130 PrOpoly=coeffvalues(PrOpdf);
131 PrO=polyval(PrOpoly,SS);
132 % % Plot pdf against original normalized sensor values
133 % plot(SS,PrO,':b')
134 % hold off
135
136 % % Plot error between rO and PrO
137 % Err=PrO-rO;
138 % plot(SS,Err)
139
140 %% Sensor Probabilities (Empty)
141
142 % rE=normpdf(SS,500,500);
143
144 rE=[25;34;54;62;67;71;73;74;74;73;71;65;57;51;45;40;35;31;27;24;21;18;16;...
145     14;12;11;10;9;8;8;7;7;6;6;5;5;5;5;4;4;4;4;4;4;3;3;3;3;3;3;3;3;3;3;3;...
146     2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;1;1;1;1;1;1;1;1;1;1;1;1;1;...
147     1;1;1;1;1;1;1;1;1;1;1;1];
148
```

```
149  % Normalize rE Vector (ensure sum(rE) = 1)
150  rE=rE./sum(rE);
151
152  % determine function representing this pdf
153  load PrEmdlfunc2;
154  PrEpdf=PrEmdlfunc2;
155  PrEpoly=coeffvalues(PrEpdf);
156  PrE=polyval(PrEpoly,SS);
157
158
159  % PrE=PrE./sum(PrE);
160
161  % % Plot pdf against original normalized sensor values
162  % plot(SS,PrE,':b')
163  % hold off
164
165  %% Plot both pdf's together
166  % z=zeros(100,1);
167  % figure()
168  % plot(SS,PrE,':r',SS,PrO,'--b',SS,z,'k')
169  % title('Sonar Return Signal Probability Density Functions')
170  % xlabel('Return Signal Value')
171  % ylabel('Probability of Receiving Signal Value')
172  % legend('P[r_t_+_1 | s(C)=Emp]','P[r_t_+_1 | s(C)=Occ]')
```

A.5   Sonar Sweep and Imaging Function: *sweep.m*

```
1   function sweep( hdg,xv,yv )
2   %SWEEP Performs a sonar sweep simulation given known input mapping data
3   %    This function takes vehicle state inputs and a known world model of the
4   %    environment to determine signal strength values for occupied/unoccupied
5   %    pixels in the sonar sweep.  It then stores these signal strength values
6   %    in a sonar image model matrix representing a notional qualitative sonar
7   %    image to be used for updating the Occupancy Grid.
8
9   %    Inputs:
10  %    hdg:    Vehicle heading (psi) in NED coordinate frame
11  %    xv:     X coordinate of vehicle centroid
12  %    yv:     Y coordinate of vehicle centroid
13
```

```
14  global SnrImg WorldMdl L H R Rs
15  alpha = 45*pi/180;          % Sonar spread half—width in radians
16
17  %*****Next four values are arbitrarily set. Once method is validated, this
18  %     might need to be changed so that the signal returns are more ...
        randomized
19  %     based on sonar sensor model.*****
20  sigOccMed = 4000;   % Median signal strength to use for occupied cells
21  sigOccDev = 750;    % Max deviation from occupied median signal strength
22  sigEmpMed = 300;    % Median signal strength to use for empty cells
23  sigEmpDev = 250;    % Max deviation from empty median signal strength
24
25
26  %******Change the next line to refine sweep***************
27  ang = 1*pi/180;            % Angle increment in radians *
28  %*********************************************************
29
30  xs = xv + L/2*sin(hdg);    % Sonar transmitter x—coordinate in tank frame
31  ys = yv + L/2*cos(hdg);    % Sonar transmitter y—coordinate in tank frame
32
33  % Note:  following are scaled by 10 to change into the matrix space, vice
34  % the cartesian space
35  ms = round(10*(R+(H—ys)));     % Map matrix row index of ys
36  ns = round(10*(R+xs));          % Map matrix column index of xs
37
38  % This 'for' loop iterates through sonar spread in angular increment
39  % specified by variable 'ang'.  At each step, it generates a Bresenham
40  % line, which it then compares to the known image map ('WorldMdl') to ...
        assign
41  % a value of 1 or 0 to each point on the Bresenham line corresponding to
42  % the value for that point within the image map.
43
44  for a=hdg—alpha:ang:hdg+alpha
45      mr=ms—round(Rs*cos(a));   % row index of Bresenham line endpoint ...
            y—coord
46      nr=ns+round(Rs*sin(a));   % column index of Bresenham line endpoint ...
            x—coord
47      % Bresenham line creation
48          [bn bm] = bresenham(ns,ms,nr,mr);
49          len = length(bn);
50
```

```matlab
51      bv=zeros(1,len);
52      for j=1:len
53          bv(j)=WorldMdl(bm(j),bn(j));    % Assign 1 or 0 value to each ...
                point on
54                                          % Bresenham line corresponding ...
                                              to those
55                                          % points in input binary matrix
56
57          % This next part chooses random values between min empty/occupied
58          % signal and max empty/occupied signal as defined above. This
59          % randomizes the signal return value, adding realism to the
60          % simulation.
61          if bv(j) == 0
62              sigEmpmin=sigEmpMed-sigEmpDev;
63              sigEmpmax=sigEmpMed+sigEmpDev;
64              sigEmp=randi([sigEmpmin sigEmpmax]);
65              SnrImg(bm(j),bn(j)) = sigEmp;
66          elseif bv(j) == 1
67              sigOccmin=sigOccMed-sigOccDev;
68              sigOccmax=sigOccMed+sigOccDev;
69              sigOcc=randi([sigOccmin sigOccmax]);
70              SnrImg(bm(j),bn(j)) = sigOcc;
71              break
72          end
73      end
74
75  end
76  end
```

A.6 Occupancy Grid Updater and Information Gain Calculation Function: *OGupdate.m*

```matlab
1  function [IG1 ]=OGupdate( snrimg,occpdf,emppdf )
2  %OGUPDATE Updates the probabilities in Occupancy Grid for each sonar sweep
3  %and calculates the resulting information gain.
4  %   This function takes the sonar image developed in the sweep function as
5  %   well as the sensor signal probability density functions, and
6  %   updates the Occupancy Grid usign the Bayesian Inference equation. It
7  %   then calculates the information gain acheived during each cell update,
```

```matlab
 8  %    and finally calculates the cumulative information gain over the entire
 9  %    occupancy grid for the given trajectory.
10
11  global OG %HE
12  [r c]=size(snrimg);
13  ig1=zeros(r,c); % Matrix preallocation for speed
14  % ig2=zeros(r,c);
15  for i=1:r
16      for j=1:c
17          if snrimg(i,j) ≠ 0
18              PsO = polyval(occpdf,snrimg(i,j));
19              PsE = polyval(emppdf,snrimg(i,j));
20          elseif snrimg(i,j) == 0
21              PsE=0.5;
22              PsO=0.5;
23          end
24
25          PcOt=OG(i,j);     % Prior OG cell value
26
27          % Bayesian inference formula for posterior OG cell value:
28          PcO=(PsO*PcOt)/(PsE*(1—PcOt)+PsO*PcOt);
29
30          % Simple IG method
31          ig1(i,j)=abs(PcO—PcOt);
32
33          % Alternate IG method: This method was derived from the
34          % Probabilistic Robotics book. However, it proved to be very
35          % computationally intensive, and produced results very close to the
36          % much simpler method actually employed above, thus it is not used
37          % at this time.
38
39              % Update Entropy
40                  % Pi=OG(i,j); % Prior cell occupancy probability
41                  % Ptrue=PcO;  % Probability for correctly measuring ...
                         cell is Occupied
42                  % Hp=—Pi*log2(Pi)—(1—Pi)*log2(1—Pi);    % Prior cell ...
                         entropy
43                  % HE(i,j)=Hp; % Update entropy grid
44
45              % Expected entropy after sensing (Probabilistic Robotics,
46              % Eqn. 17.15, pg. 584)
```

95

```
47  %          EH=−Ptrue*Pi*log2(Ptrue*Pi/(Ptrue*Pi+(1−Ptrue)*(1−Pi)))...
48  %             ...
    −(1−Ptrue)*(1−Pi)*log2((1−Ptrue)*(1−Pi)/(Ptrue*Pi+(1−Ptrue)*(1−Pi)))...
49  %             ...
    −(1−Ptrue)*Pi*log2((1−Ptrue)*Pi/(Ptrue*(1−Pi)+(1−Ptrue)*Pi))...
50  %            −Ptrue*(1−Pi)*log2((1−Ptrue)*Pi/(Ptrue*(1−Pi)+(1−Ptrue)*Pi));
51  %
52  %          ig2(i,j)=Hp−EH;  % Expected Information Gain (Probabilistic ...
    Robotics,
53  %                      % Eqn. 17.4, pg. 572)
54
55         % Update Occupancy Grid
56         OG(i,j)=PcO;
57
58
59
60     end
61  end
62  IG1=sum(sum(ig1)); % Calculate total IG over entire OG
63  % IG2=sum(sum(ig2));
```

A.7   Heuristics Violation Analysis Function: *HeurViolArea.m*

```
1  function [ ViolArea ] = HeurViolArea( xS,yS,xViol,yViol )
2  %HEURVIOLAREA−−calculates heuristics boundary violation areas
3      % This function uses the vectors of stern point coordinates to
4      % calculate the total violation area between the arc swept by the
5      % vehicle stern and the heuristics boundaries.
6
7  global RBx BBy
8
9  if (any(xViol)) && (any(yViol))
10     ViolArea=NaN;
11  elseif (any(xViol)) || (any(yViol))
12
13     if any(xViol)
14         % fprintf('Run %g: Violation of Right Boundary.\n',n)
15         xViolVec=xS(find(xViol));
16         xvl=length(xViolVec);
17         for i=1:xvl
```

```matlab
18              xV(i)=xViolVec(i)-RBx;
19          end
20
21          % ***Calculate Violation Area
22          xVint=trapz(yS(find(xViol)),xV);
23          ViolArea=abs(xVint);
24          % fprintf('Violation of Right Boundary: %6.4g\n',ViolArea)
25          % ***
26
27          % ***Plot right boundary violation area—use this for a visual plot
28          % of the actual boundary violations.
29
30 %          RBxp=ones(1,xvl)*RBx;
31 %          xviolfig=figure();
32 %          plot(yS(find(xViol)),xViolVec,'k',yS(find(xViol)),RBxp,'r*');
33 %          jbfill(yS(find(xViol)),RBxp,xViolVec);
34 %          axis equal
35 %          tstr=sprintf('Run %g Right Boundary Violation Plot',n);
36 %          title(tstr)
37 %          xlabel('y')
38 %          ylabel('x')
39 %          legend('Stern Path','Right Boundary','Location','North')
40 %          set(gca,'YDir','reverse')
41 %          camroll(90)
42          % ***
43      end
44
45      if any(yViol)
46          % fprintf('Run %g: Violation of Bottom Boundary.\n',n)
47          yViolVec=yS(find(yViol));
48          yvl=length(yViolVec);
49          %[myv nyv]=size(yViolVec) % t/s only
50          for i=1:yvl
51              yV(i)=BBy-yViolVec(i);
52          end
53
54          % ***Calculate Violation Area
55          yVint=trapz(xS(find(yViol)),yV);
56          ViolArea=abs(yVint);
57          % fprintf('Violation of Lower Boundary: %6.4g\n',ViolArea)
58          % ***
```

```
59
60          % ***Plot bottom boundary violation area—use this for a visual ...
               plot
61          % of the actual boundary violations.
62
63 %          BByp=ones(1,yvl)*BBy;
64 %          [mby nby]=size(BByp) %t/s only
65 %          yviolfig=figure();
66 %          plot(xS(find(yViol)),yViolVec,'k',xS(find(yViol)),BByp,'r*')
67 %          jbfill(xS(find(yViol)),BByp,yViolVec);
68 %          axis equal
69 %          tstr=sprintf('Run %g Bottom Boundary Violation Plot',n);
70 %          title(tstr)
71 %          xlabel('x')
72 %          ylabel('y')
73 %          legend('Stern Path','Bottom Boundary','Location','Best')
74        % ***
75     end
76
77 else ViolArea = 0;
78 end
79
80 end
```

A.8   Avoidance Penalty Analysis Function: *Avoidance.m*

```
1 % This script generates the World Model matrix and initializes the matrices
2 % for the sonar image, heuristics analysis, occupancy grid, and information
3 % gain value storage.
4
5 % World Model (Input data for simulation)
6 global  HMat OG SnrImg WorldMdl R M N IG1 ObsFlag Rs H xl xr yl yu xl2 ...
    xr2 yl2 yu2 %HE
7 % Tank matrix for Bresenham imaging algorithm
8 Rs=10*R;              % Scales the sonar range value for matrix cell
9                       % index addressing.
10 WorldMdl=zeros(M,N);
11
12 % The following 4 'for' loops place ones in tank matrix space outside ...
    the tank
```

```matlab
13  for i=1:Rs
14      WorldMdl(i,:)=1;
15  end
16  for j=M-Rs:M
17      WorldMdl(j,:)=1;
18  end
19  for k=1:Rs
20      WorldMdl(:,k)=1;
21  end
22  for l=N-Rs:N
23      WorldMdl(:,l)=1;
24  end
25
26  if ObsFlag
27  % Obstacles in tank:
28  m1=10*(R+H-yu);
29  m2=10*(R+H-yl);
30  n1=10*(xl+R);
31  n2=10*(xr+R);
32      WorldMdl(m1:m2,n1:n2)=1; % Assigns value of one to all World Model
33                               % cells corresponding to the obstacle location.
34
35  m12=10*(R+H-yu2);
36  m22=10*(R+H-yl2);
37  n12=10*(xl2+R);
38  n22=10*(xr2+R);
39      WorldMdl(m12:m22,n12:n22)=1;
40
41  end
42
43  SnrImg=zeros(M,N);   % Initialize Sonar Image matrix
44  HMat=WorldMdl;       % Used for Heuristic Violation Analysis
45  OG=.5*ones(M,N);     % Initialize Occupancy Grid
46  IG1=zeros(M,N);      % Initialize Information Gain matrix
```

99

THIS PAGE INTENTIONALLY LEFT BLANK

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Research Associate Professor Douglas Horner
   Center for Autonomous Vehicle Research (CAVR)
   Naval Postgraduate School
   Monterey, California

4. Professor Oleg Yakimenko
   Department of Mechanical and Aeronautical Engineering
   Naval Postgraduate School
   Monterey, California