Autonomous Aerial Localization of Radioactive Point Sources via Recursive Bayesian Estimation and Contour Analysis

Jerry A. Towler

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Electrical Engineering

Kevin B. Kochersberger, Chair

William T. Baumann

Cornel Sultan

June 16, 2011

Blacksburg, Virginia

Keywords: Source Localization, Recursive Bayesian Estimation, Contour Following,

Unmanned Systems, UAV

Autonomous Aerial Localization of Radioactive Point Sources via Recursive Bayesian Estimation and Contour Analysis

Jerry A. Towler

The rapid, accurate determination of the positions and strengths of sources of dangerous radioactivity takes high priority after a catastrophic event to ensure the safety of personnel, civilians, and emergency responders. This thesis presents approaches and algorithms to autonomously investigate radioactive material using an unmanned aerial vehicle.

Performing this autonomous analysis comprises five major steps: ingress from a base of operations to the danger zone, initial detection of radioactive material, measurement of the strength of radioactive emissions, analysis of the data to provide position and intensity estimates, and finally egress from the area of interest back to the launch site. In all five steps, time is of critical importance: faster responses promise potentially saved lives.

A time-optimal ingress and egress path planning method solves the first and last steps. Vehicle capabilities and instrument sensitivity inform the development of an efficient search path within the area of interest. Two algorithms—a grid-based recursive Bayesian estimator and a novel radiation contour analysis method—are presented to estimate the position of radioactive sources using simple gross gamma ray event count data from a nondirectional radiation detector. The latter procedure also correctly estimates the number of sources present and their intensities.

Ultimately, a complete unsupervised mission is developed, requiring minimal initial operator interaction, that provides accurate characterization of the radiation environment of an area of interest as quickly as reasonably possible.

This research was supported wholly by funding from the Defense Threat Reduction Agency, Virginia Tech's Unmanned Systems Laboratory, and Viewers Like You.

Acknowledgments

It is good and right, before beginning the exposition of so much research and hard work, to thank those without whom this document would have remained forever empty.

My parents read and edited the first school paper I ever wrote, and it is with the same critical, loving, but unflinchingly honest eyes that they helped guide me through the refinement of this culmination of so many years of their efforts. Their expectations, always higher than I thought I could reach, and yet within surprisingly easy grasp with their support, have driven me to depths of discovery only they could have envisioned. As my parents pulled me from in front, so my younger brother Will and my younger sister Callie nipped at my heels. With a complete disrespect for my desire to set unattainable goals for them by my example, they have continually surprised and exasperated me by calmly observing, admiring, and then obliterating every high-water mark I left. It is through my family's combined encouragement that I have achieved as I have, and I offer my thanks and my love. And, for Will and Callie, yet another triumph for them to gaze at in wonder until they are ready to stride quickly past it for themselves.

My colleagues at the Unmanned Systems Laboratory have provided endless support, aid, and encouragement. From Donnie Rogers, a lowly undergrad who happily took extensive measurements of the RMAX for me, to Dr. K, the lab's director and my advisor, all have contributed to this document's completion. Their ruthless refusal to let me work in peace led to wildly tangential discussions and tremendously fruitful, and sometimes forceful, changes of perspective, without which some of these algorithms would have languished, incomplete. I offer my thanks to all of them, and to my friends outside the lab who reminded me, when necessary, that life outside of robotics does exist, and how good it can be.

My Methodist youth counselor through high school, and close friend ever since, Chris League has taught me more about what kind of person I want to see myself become than any other. I thank him for his unfailing support and unflagging devotion to my success as a person and friend, as starkly differentiated from my success as a researcher.

Finally, I thank my God, in whose hands alone my entire life and work have been, are, and will remain, and who has seen fit to bring me to this point. His works are mighty beyond understanding, and I thank him for allowing me to reflect a small part of his glory through my relationships and my research.

"Old father, old artificer, stand me now and ever in good stead."

Contents

List of Figures

List of Tables

List of Algorithms

Nomenclature

Acronyms

E-box	electronics box
EKF	extended Kalman filter
GSR	ground sampling robot
PDF	probability density function
PID	proportional-integral-derivative [controller]
RBE	recursive Bayesian estimator
SLAM	simultaneous localization and mapping
SNL	Sandia National Laboratories
UAV	unmanned aerial vehicle
UKF	unscented Kalman filter
USL	Unmanned Systems Laboratory

Chapter 1

Introduction

The Unmanned Systems Laboratory (USL) at Virginia Tech has developed a number of systems to facilitate rapid, safe response to an explosive event, specifically one involving the dispersal of radioactive material. Whether from a nuclear attack involving the detonation of a yielding device; a dirty bomb, in which radioactive material is dispersed via conventional explosives; or a serious accidental occurrence, such as at a nuclear power plant, response teams must quickly assess the situation to present effective safety zones and begin evaluating the causes and possible responses to the disaster. The danger to personnel suggests the use of autonomous robots to perform the initial exploration, driving the development in many labs, including the USL, of unmanned aerial vehicles (UAVs) carrying arrays of sensors to perform these dangerous, time-critical tasks. This thesis presents a complete radiation reconnaissance mission comprising an efficient ingress to the area of interest, a directed search for radioactive point sources, and the estimation of point source location and intensity.

Because of its commonality across detection efforts such as geological surveying, disaster response, and search-and-rescue missions, the problem of point source localization has been explored extensively. However, much radiation sampling technology relies on expensive, heavy, high-volume detectors that are unsuitable for deployment on a lightweight, expendable craft. This thesis explores techniques available for a relatively small single-crystal detector that fits on a small UAV with limited lifting capacity. The following sections present background on the USL's mission and challenges specific to this application.

1.1 Mission Background

The vehicle platform selected for the mission is the Yamaha RMAX UAV, the small-scale helicopter seen in Figure 1.1. With an overall length of 3.63 m and a main rotor diameter of 3.115 m, the RMAX has a maximum takeoff weight of 94 kg and a maximum payload of 28 kg. The USL has modified the helicopter for suitability to disaster response missions. Specifically, a WeControl onboard autopilot system was added, giving high-quality position and velocity data as well as the ability to input high-frequency velocity commands to implement controllers onboard. In addition, two major payloads have been developed: a permanent electronics box (e-box) to coordinate onboard instruments, run flight control software, and communicate with a ground control station; and a set of modular pods that support specific mission operations.

The overall mission has three main components: aerial surveillance and topography mapping,

radiation mapping and localization, and ground sample collection. For each mission section, a specific payload has been developed. The topographical surveillance mission utilizes a carbon-fiber stereo camera boom to generate three-dimensional terrain maps overlaid with color images. Flight trajectories developed by USL engineers maximize vehicle endurance while providing sufficient data for complete topographical imaging. These maps facilitate ground robot path planning as well as providing an initial overview of the situation and potentially-new topography to mission operators.

A radiation detector developed by Sandia National Laboratories (SNL) and provided to Virginia Tech comprises the entirety of the radiation mapping pod, described fully in Subsection 1.2.2.

Finally, ground sample collection entails the deployment of a winch pod containing a ground sampling robot (GSR), also developed at the USL. After the localization of a radioactive sample via the techniques described in this thesis, the helicopter hovers over the sample location and lowers the GSR to the ground. A tension controller allows the GSR to navigate freely while remaining tethered to the stationary UAV. After collecting the sample, the GSR is winched back into the air, flown back toward the ground station, and released over a safe containment vessel. A high-level view of the system payloads is shown in Figure 1.2.

Figure 1.1: Yamaha RMAX UAV (Matt Torok/USL, used with permission)

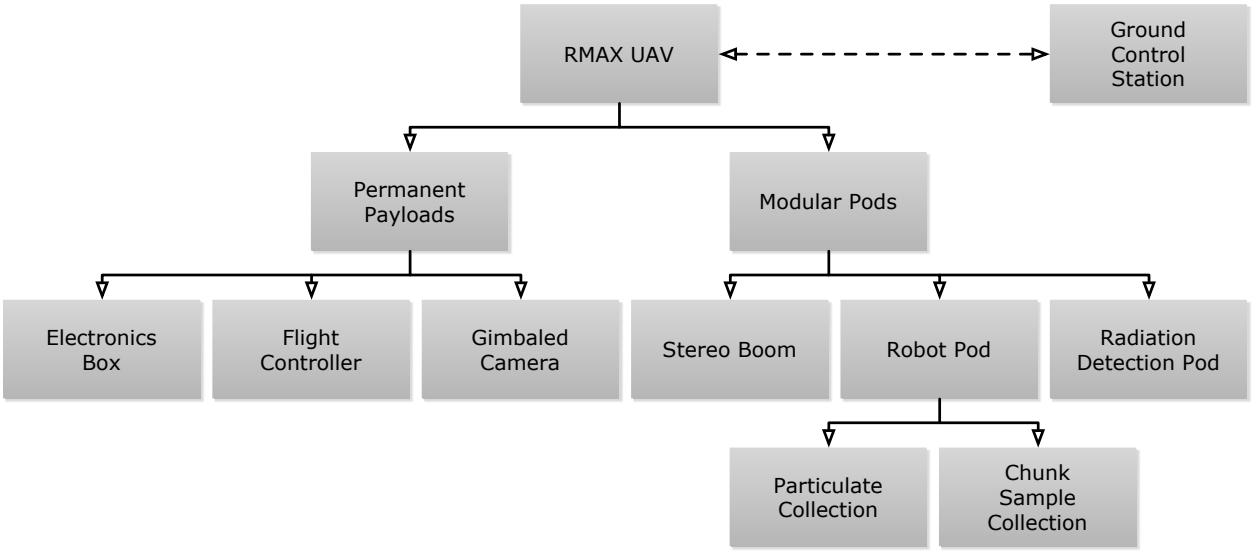


Figure 1.2: System payload architecture

1.2 Mission Constraints

Both the nature of the USL's proposed mission and the chosen vehicle generate constraints within which any of its subsystems must operate. This section explores these constraints and their impact on the techniques presented in this thesis.

1.2.1 Time and Fuel

Because the overall system will function as a disaster response and recovery tool, time is a crucial element of the response. Specifically, in a disaster involving nuclear material, responders must rapidly establish a perimeter outside of which rescue personnel may safely operate. Delays in identifying the danger zone may expose humans to harmful radiation. Fast response time will also enable emergency workers to contain any hazards more easily than after a significant delay. Therefore, a primary consideration in developing radiation localization techniques must be the ability to perform position estimation in a reasonable period of time; in the best case, the estimation should occur in real time as the helicopter flies.

The computational ability of the onboard computer factors into the algorithm's abilities as well. Because it must use onboard power, the e-box computer is a PC/104 stack with a 2 GHz Intel® Core™ 2 Duo processor and 2 GB of RAM. As a result, any localization algorithm must run at the desired speed on that hardware in parallel with other sensor management and communication software.

Although safety concerns instill an urgency in the mission, physical limitations carry their own constraints. The helicopter carries enough fuel for roughly one hour of flight time with the radiation pod mounted, so the total ingress, search, localization, and egress time must be less than that to avoid the unnecessary loss of the vehicle. This thesis presents a technique to minimize ingress and egress time with a goal of extending the time available for search and localization as much as possible. Because of the fuel-imposed time limit, the search path must efficiently cover the area of interest while ensuring that the algorithm does not overlook a radioactive source. Once a source is detected, the localization technique must be time efficient even in the presence of several sources spread over a large area.

1.2.2 Radiation Detector Design

The small size of the unmanned platform limits the payload weight capacity, which influences the complexities and the capabilities of the detector used. The detector provided by SNL comprises a cylindrical sodium-iodide scintillating crystal to convert gamma rays to visible light photons, a massive photomultiplier to amplify flashes in the crystal to voltage levels appropriate for traditional electronics, and an embedded computer to translate the analog signal to spectral information and communicate with the e-box. The crystal in this detector is a cylinder nine inches long and three inches in diameter, for a volume of 1.042 L. The detector returns the integrated number of counts over a one-second period. As a consequence of the nonnegligible integration period, this thesis assumes that all counts returned for a period occurred at the detector's location at the end of that period.

Because the scintillating crystal in the detector is not a sphere, its shape provides a small amount of directionality. A high-energy gamma ray approaching the crystal from the side may penetrate the entire crystal without reflecting off the crystal's molecules to produce the visible photon collected by the photomultiplier tube. The same gamma ray approaching the detector from the end must pass through three times as much material without colliding at all to avoid detection. The much lower probability of the latter event means that the detector will measure a slightly higher event rate when "pointed at" a source than when turned orthogonal to it. This directionality is attenuated, however, with distance from the detector. The plots in Figure 1.3 show the reduced effect of directionality as the distance from the detector to the source increases from twenty meters to sixty meters. Because of the slightness of this effect and its decrease at distance, this thesis assumes that the detector is nondirectional.

1.3 Proposed Approach and Organization

The remainder of this thesis comprises three major topics: planning efficient flight paths to and from an area of interest, detecting the presence of one or more radioactive point sources within the area of interest, and quickly and accurately estimating the position of those point sources once detected. Two different algorithms are proposed to perform localization: a statistical Bayesian method and a more straightforward contour analysis method.

A review of research related to the work presented here is given in Chapter 2. These areas

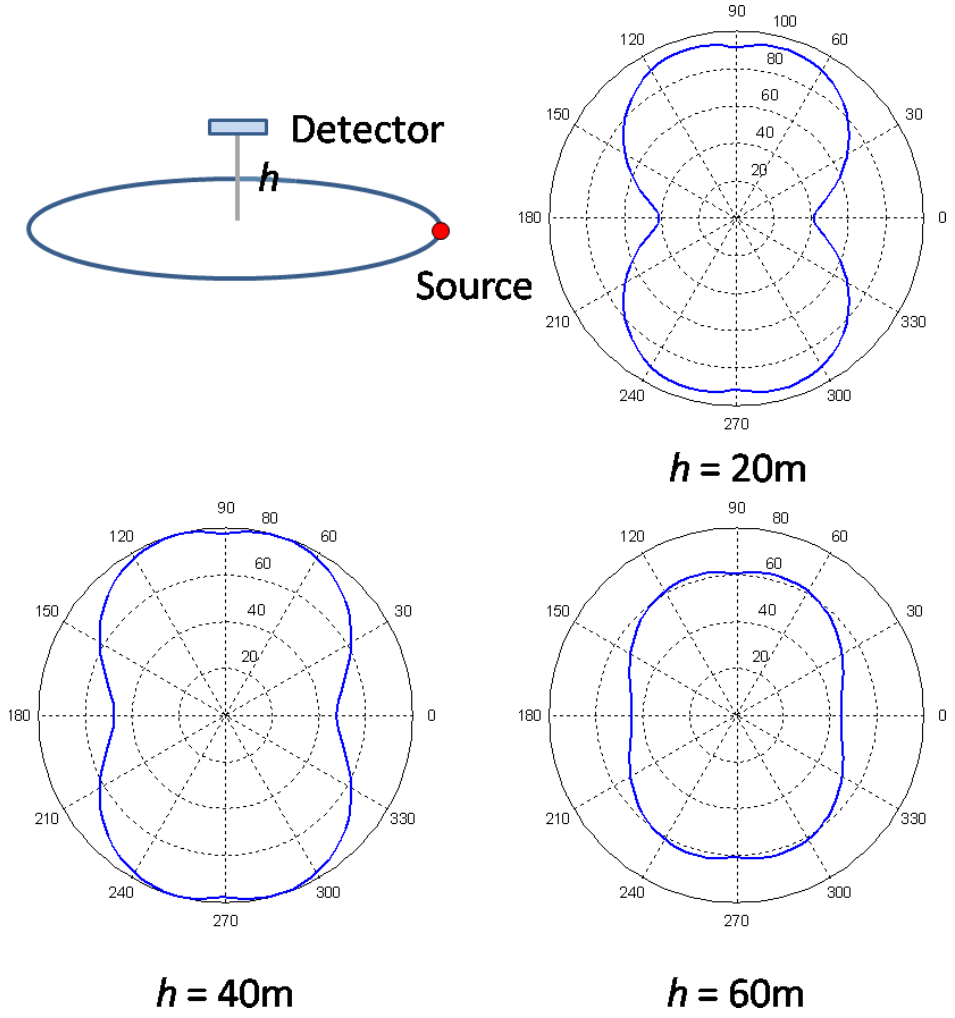


Figure 1.3: Detector directionality at various distances from a source

include radiation mapping, recursive Bayesian estimators (RBEs), contour following, and the Hough transform.

The necessary mathematical and programmatic background on which this work is based is presented in Chapter 3. Specifically, the basic workings of nuclear radiation detection, recursive Bayesian estimators, and the Hough transform are given, as well as their programmatic implementations, where appropriate.

The first localization method, the grid-based RBE method, is presented in Chapter 4. Algorithm development considerations and simulation results are provided in addition to the algorithm itself. The reasoning and methods behind the development of the second localization method, contour analysis, are given in Chapter 5.

Presented in Chapter 6 are the algorithms and flight plans chosen and developed for optimally fuel-efficient travel to and from the area of interest.

Chapter 2

Literature Review

Because of the importance of responding appropriately and quickly to nuclear material dispersion, significant research has been done in the area of point source detection and localization. Although the constraints imposed on the work contained in this thesis yield unique methods and results, reviewing prior research involving robots of greater size and capability helps guide the decisions presented herein. Both of the localization techniques developed in later chapters draw on the work of previous researchers, and the contour analysis method specifically combines elements from disparate fields to perform a fast, general localization. This chapter traces some of the research most relevant in developing these techniques as well as some related research incorporating these approaches.

2.1 Radiation Detection

The field of radiation detection can be roughly divided into two major sub-topics, focusing on either localization or mapping. In general, localization work addresses the problem of finding a point source of radiation using the overlaying radiation distribution as a guide in the search. Radiation mapping, conversely, usually cares little about individual sources and seeks instead to generate a complete map of an area.

2.1.1 Point Source Localization

To localize a radioactive source implies, in practice, estimating its position using received radiation measurements from one or more sensors or sensor arrays, usually gamma-ray detectors. In addition to being useful in disaster response scenarios, everyday work with radioactive sources requires extreme caution and security to ensure the safety of researchers as well as defense against loss and theft.

Approaching the problem from the standpoint of a warehouse operator attempting to track a radioactive source as it moves about the facility, Howse et al. propose several algorithms to follow the source's movement [21]. Using a concept familiar from global positioning system calculations, they evaluate level curves of potential source locations for each of four detectors placed in the corners of a room and take their intersection points as the most likely locations for the sources. They use a recursive nonlinear least squares optimization algorithm to reconcile sensor estimates of the source location into a single position estimate. The researchers

assume that the intensity of their ^{137}Cesium source is known *a priori* and successfully track the source in real time within an error of, on average, one foot, in a roughly 50×30 ft room.

Howse et al. provide, in addition to their research, a very useful analogy to localizing a radioactive source, presented here in its entirety:

> Imagine a building containing only one room and having a flat roof with four skylights cut into the ceiling, each near one corner of the building. You are standing on the roof and someone is walking around in the room holding a candle. You must determine the position of the candle in the room by observing the relative brightness of the light coming through the four skylights. Keep in mind that the flickering of the candle leads to variations in its brightness that are proportional to the square root of the brightness itself.[1]

When the trajectory of a source, but not its intensity, is known, an array of sensors may be used to detect the presence or absence of radioactive material, such as in [6]. Brennan et al. characterize the bounds of material detection using a network of sensors with known but randomized locations and find that real-time Bayesian classification algorithms are feasible and accurate in networks with fewer than ten nodes.

The nonlinear, non-Gaussian nature of radioactive decay suggests that the most appropriate Bayesian algorithm must go beyond the standard Kalman filter. It is tempting simply to attempt to linearize the model in the style of the extended Kalman filter (EKF), but Muske

[1] Howse et al., p. 1738

and Howse found that the nature of the nonlinearities inherent in the model can drive at least the EKF unstable [31]. At least part of this instability results from the highly non-Gaussian shape of the Poisson model that describes radioactive sources at low count rates.

In the previous cases, either the number of sources, the intensity of the sources, or the trajectory of the source is known, and multiple radiation sensors are used to perform estimation. As the proposed mission of this thesis has none of those luxuries, more relevant research is found in Morelande et al., which evaluates two methods of simultaneously estimating the number of sources present and their locations and intensities [29]. In the first case, the maximum likelihood estimator (MLE) is used to find the source positions and intensities, and a generalized maximum likelihood rule chooses the number of sources. The second technique estimates parameter values with a Bayesian algorithm very common in particle filter applications, known as importance sampling with progressive correction. The paper concludes that the MLE technique successfully estimates parameters for zero, one, or two sources, but fails when a third source is introduced, while the Bayesian method performs well in all scenarios.

The preceding work suggests that the Bayesian technique to be used must be at least as powerful in the face of nonlinearity and non-Gaussian distributions as the particle filter, and that consideration is where the work on which this thesis is most directly based began. Focusing again on the case in which the number and intensity of the sources are known, Brewer uses the USL platform and a particle filter to successfully localize a single point source within a $250 \times 250\,\mathrm{m}$ box [7]. The algorithms presented in this thesis relax both assumptions Brewer makes about prior knowledge available to the algorithm; they also extend the search

area, increase estimation accuracy, decrease estimation time, and expand from localization only to an entire mission plan.

2.1.2 Radiation Mapping

Mapping the radiation emissions of an area can help determine the soil and mineral content of the area, evaluate radiation exposure after a catastrophic event like Chernobyl or Fukushima, or help guide localization searches. The first two cases tend to cover extremely large areas, measured in hundreds of square kilometers, and use manned low-flying fixed-wing aircraft on simple predefined trajectories. The final case can employ much smaller robots moving more slowly and more deliberately for a finer-grained map.

Large-scale Radiation Mapping

Aerial gamma-ray surveys have been conducted since as early as the 1940s to aid in fields as disparate as geological mapping, mineral exploration, and nuclear surveillance. Initial surveys collected only gross count measurements, as the instruments were simple scintillometers and Geiger counters that had no spectral decomposition abilities. The received counts, therefore, were the combination of naturally-occurring potassium, uranium, and thorium. By the 1960s, detector technology had advanced sufficiently to allow the separation of the spectrum to evaluate the individual contributions of distinct radioactive isotopes. Potassium content was measured directly by examining ^{40}K, while uranium and thorium were measured by

monitoring returns from their decay series, specifically ^{214}Bi for uranium and ^{208}Tl for thorium. In 1980, Pitkin and Duval determined required detector and trajectory parameters to obtain desired survey uncertainty values [32]. Importantly with regard to the work in this thesis, they investigated the effects of detector velocity on detector capability.

Some aerial radiological surveys are intended to classify the soil and mineral content of the area being monitored [15]. Even in the absence of high-quality geological surveying, Galbraith and Saunders were able to identify specific lithological types and determine uranium source rocks by correlating known features of rock types with the geometry of the radiation data observed.

Despite advancements in technology, the gamma-ray detectors used in the aircraft on these survey missions are extremely large, complex, and expensive. Sanderson mentions that even as recently as 2008, these detectors consisted of several very large sodium iodide (NaI) scintillators. As an example, the detector used by Sanderson had four $10 \times 10 \times 40\,\text{cm}$ crystals, for a total volume of 16 L, as well as an additional germanium (Ge) detector [34]. These surveys also typically encompass extremely large areas; typical flight line spacing as of 1991 was in the range of 1 km [17].

For a much more recent comparison, including applications to disaster response, the United States Environmental Protection Agency (EPA) published an overview of its system in 2010 [9], mentioning its six $2 \times 4 \times 16$ inch NaI detectors (a total of 12.6 L) and two 3×3 inch lanthanum bromide ($LaBr_3$) detectors. The paper focuses on disaster response, specifically referencing the September 11 attack on the World Trade Center and various response pro-

grams developed as part of the National Response Framework. The EPA focuses on the need for time-critical movement, emphasizing that its system can be deployed anywhere within the continental United States within nine hours. Their detector provides more than one million channels of spectral data; for comparison, the SNL detector used for this thesis has only 1024 channels.

Small-Scale Mapping

For radiation mapping applied on a reduced scale for such purposes as localization or situational awareness, much smaller robots are used, necessitating significantly smaller detectors. Because less area is to be monitored, researchers have developed algorithms allowing robots to determine their own paths to increase mapping efficiency.

Cortez et al. approach the mapping problem as a basic uncertainty-minimization problem. They attempt to direct their mobile robot into areas of greatest uncertainty and take more measurements in those locations to derive a map of a given area [12]. They label their techniques "gradient-based Bayesian" and "sequential-based Bayesian," and use as a metric of uncertainty the variance of the measured distribution over the expected radiation level at that point. In the gradient-based Bayesian search, the robot follows not gradients of counts, which make no sense in the stochastic environment of radiation sensing, but rather gradients of uncertainty. The area is divided into cells, and the robot remains within a cell until a neighboring cell has a higher value of the potential function, which Cortez defines as related to the distance from the desired search end point and the variance of neighboring

cells. The sequential-based Bayesian search is time-optimal in the sense that each grid cell is visited exactly once, but the path planning is done *a priori* and the robot does not leave a particular grid cell until it has collected enough measurements to reduce the uncertainty below a desired threshold.

The same authors extend their analysis in a follow-up report to multiple robotic platforms mapping a predefined area. The robots distribute work based on equalizing the area of Voronoi cells with respect to radiation density as they perform the mapping [11]. This work uses algorithms similar to the previous analysis to determine robot movement between grid cells.

To help speed the mapping task, Kumar et al. use similar uncertainty calculations but allow continuous movement of the robot along a one-dimensional path [26]. Instead of stopping in grid cells, however, in this case the robot maintains a constant velocity as long as uncertainty stays low and slows down when the variance increases above a desired threshold.

All three of these investigations derive their division of the search area from the development of occupancy grids, in which an area is divided into cells and each cell may be marked with certain parameters. In simple cases, the marks may be "occupied," "unoccupied," and "unvisited." These methods were developed by Alberto Elfes as part of his doctoral dissertation at Carnegie Mellon University; they were first mentioned in a paper Elfes co-authored in 1985 [28] and first published formally in 1987 [14].

2.2 Recursive Bayesian Estimators

The Kalman filter became the most commonly used tool in the development of control systems, and specifically robotics, because of its simplicity, applicability, and ease of use. Since its original publication in 1960 by R.E. Kalman [23], researchers have extended and applied it to innumerable measurement problems, most notably in the field of autonomous navigation. Certain variations of the Kalman filter have become so popular in their own right that they have acquired specific names: the extended Kalman filter (EKF) and the unscented Kalman filter (UKF) are the most popular of these. The development of these filters made Bayesian estimation techniques available for non-linear, non-Gaussian systems, but as discussed above, the application to radiation monitoring and localization requires at least their more general cousin, the particle filter.

The particle filter, also known as the sequential Monte Carlo method, is, as expected from the name, derived from the Monte Carlo method developed by Metropolis and Ulam in 1949 [27]. The method was later applied to Bayesian localization, notably by Gordon et al. [16], who developed the original particle filter algorithm as it stands today. Their aim was to avoid the computational complexity of evaluating a probability density function (PDF) over static grid nodes, even when those nodes may not be useful for the estimation desired. By randomly choosing the initial particle locations and updating their positions via Bayes' rule, the particles converge on the desired result. The result was a recursive Bayesian estimator (RBE) that could handle any nonlinearity in the underlying model structure, did not require

either the prior or posterior PDF to be Gaussian, and required much less computational power than existing methods. This type of particle filter employs sequential importance resampling; the technique was extended to use stratified sampling by Kitagawa [24]. This extension can help reduce sampling error in certain particle populations. (The particle filter with sequential importance resampling was used by Brewer's localization method.)

Its computational benefits influence the particle filter's popularity for robotic localization, area mapping, target localization, and simultaneous localization and mapping (SLAM). A thorough examination of their use in target tracking is given by Stone [35], who explores target tracking in both single- and multi-vehicle environments and a variety of Bayesian filters and recursive Bayesian estimators. A specific example is given by Wong et al., where a team of autonomous vehicles searches for multiple lost targets, maintaining their target position probability estimates via RBE algorithms [37].

The grid-based method, the basis of one of the algorithms proposed in this thesis, is actually a regression from the particle-filtering method that solves some problems while introducing several drawbacks. Notably, it suffers from "the curse of dimensionality," as it is called by Bourgault [4]. That is, by increasing any dimension of the search, the exponent on the time and memory required by the entire search process increases. This "curse" may dramatically, and sometimes fatally, increase computation time. The problems solved, however, are those of heuristics: where a particle filter must be tuned in terms of collapse rate, degeneracy, and redistribution, the grid-based search avoids moving particles at all in favor of static grid points. Therefore, no tuning is necessary, and the problem may be approached without

reservation or concern for tuning parameters. With even inexpensive computers, such as the one available in the USL e-box, now containing sufficiently powerful processors, and efficient matrix multiplication and convolution algorithms implemented in environments such as MATLAB, a grid-based search is made possible in real time on commodity hardware.

2.3 Contour Following

Although the contour following utilized by the algorithms presented in this thesis involves level curves of radioactive intensity, it is instructive to explore other types of contour following. Two major avenues of research are those of wall following and constrained manipulator movement.

Wall following is useful primarily for small robots to navigate (and potentially map) an enclosed area by maintaining a particular distance from a wall, ensuring that some concept of self-localization is maintained (as opposed to moving into the middle of a room and losing all awareness). Much of this research is modeled after the movement of cockroaches; in fact, many robots directly model the cockroach, which uses the flagella of its antennae to maintain a constant distance from a wall [8]. The cockroach stays close to the wall while running quickly, and farther away when moving more slowly, enabling the insect to feel changes in the wall topography more intensely and adapt more quickly when necessary. The cockroach's maneuverability is significantly greater than most robots, however, able to rotate at a rate of up to twenty-five body turns per second to ensure that it does not run

into projections on the wall.

These principles were extended to robots using, for example, ultrasonic sensors instead of antennae to maintain a constant distance to the wall. In the implementation by Braunstingl et al., the controller uses fuzzy logic to control its position along a smooth trajectory while following the wall contours at high speeds [5].

Another application of contour following is most useful when the trajectory is pre-defined and a robotic manipulator must follow it as closely and as quickly as possible. Huang and McClamroch present an application where a manipulator must describe a known curve on an unknown surface [22]. This control would be most useful in carving or writing. In such scenarios, contour error reduction is obviously very important, but for some contour following tasks, error estimation techniques fast enough to work in real time are difficult to derive. Nonlinearities such as hysteresis may cause serious problems in these estimators, requiring novel techniques, such as one put forward by Cheng, to overcome [10].

2.4 Hough Transform

The Hough transform facilitates finding incomplete instances of shapes in images. Its usual application in robotics is to locate straight lines in computer vision algorithms used for navigation. The transform is based on work by Paul Hough to detect patterns in bubble chambers [19]. (A bubble chamber is a container filled with superheated liquid and is used to detect electrically charged particles.) Hough received a US patent for his technique in

1962 [20]. The utility of his work in the detection of shapes other than lines was discovered more than a decade later by Duda and Hart, who also simplified and thereby codified the technique for line finding [13].

Duda and Hart note that the Hough transform could be used to find arbitrary shapes "in principle," but discuss its application in detail with regard to only circles. That complete extension was performed by Ballard another decade later [3]. Kassim et al. did a survey of Hough transform techniques, including the generalized Hough transform [1]. A much more modern application than the original bubble chamber analysis comes from the analysis in 2008 of radar images of objects with rotating parts. The rotation, not surprisingly, generates noise about the Doppler frequency of a moving object. Zhang et al. applies the Hough transform to detect lines and curves on the spectrogram returns from the radar apparatus [38]. This detection can focus on the actual center Doppler frequency, providing a much cleaner radar image.

Chapter 3

Background

The techniques introduced in this thesis build upon a considerable body of work done by previous researchers in the fields of localization, controls, and image processing. This chapter presents the necessary conceptual and mathematical background for the techniques used or extended in the next three chapters.

3.1 Nuclear Radiation Detection

As a radioactive isotope decays, it emits a variety of types of radiation, including alpha particles ($^4_2$He), beta particles (β^-), X-rays, and gamma rays. Unfortunately for detection purposes, alpha and beta particles and X-rays suffer from self-absorption within the source, resulting in different emission characteristics for different source shapes. Therefore, radiation detection instruments rely on impingement of gamma rays [25].

3.1.1 Gamma Ray Interactions

When a gamma ray intersects with a scintillator, such as the NaI scintillator in the SNL radiation detector, any one of three types of interaction may occur [2]:

Photoelectric interactions

A photoelectron absorbs the energy of the gamma ray and emits a visible-light photon. These photons are the source of most of the counts recorded for lower-energy gamma rays.

Compton interaction(s) followed by escape of a secondary gamma ray

Part of the energy of the gamma ray is transferred to one or more Compton electrons as part of an inelastic collision of particles, and the (now weaker) gamma ray leaves the crystal without further interaction. These interactions appear to the detector as multiple lower-energy gamma rays known as Compton backscatter; they contribute to the gross gamma ray count, but not to the height of the peak for the energy of the initial gamma ray. (This type of interaction is responsible for the apparently everpresent grouping of low-energy gamma rays on a spectrograph even in the presence of only very specific isotopes.)

Compton interaction(s) followed by a photoelectric interaction

These events are similar to the previous case, but instead of the gamma ray escaping the crystal after the collisions, all of its energy is absorbed within the detector. Because all of the Compton interactions and the final photoelectric interaction occur within a

single detection cycle of the instrument, they appear as a single gamma ray of the same energy as the original one. This phenomenon accounts for most of the recorded count rate for high-energy events.

A radiation detector records all of the interactions that take place within some period of time—in the SNL detector, one second—and reports the total number of interactions as well as a spectrum with counts divided into energy ranges; the SNL detector has — such channels.

The rate thus reported by the instrument depends on the intensity of the radiation source and the distance between the instrument and the source. As expected for radiation in a homogenous medium, the received intensity is inversely proportional to the square of that distance. (This assumption arises from a simple observation: as a source radiates energy into a homogenous medium, the boundary of that energy is a sphere of increasing surface area. The surface area increases with the square of the radius, so if the energy is evenly spread over that surface, the energy per area is inversely proportional to the square of the radius.) Extensive research has investigated the accurate localization of one or many radioactive sources acting either as point sources or as fields of radioactivity. In terms of point sources, which are the focus of this thesis, Gunatilaka et al., among other observations, verified the $1/R^2$ relationship and gave a succinct equation for the measured intensity of a given source [18]:

$$\lambda_k(x) = \frac{I}{(x_k - x_0)^2 + (y_k - y_0)^2} + \lambda_b, \tag{3.1}$$

where I is the intensity of the source, (x_k, y_k) is the position of detector k, (x_0, y_0) is the position of the source, and λ_b is the background intensity at the measurement point.

3.1.2 Distribution of Gamma Ray Events

The observation of nuclear decay events is appropriately modeled by the Poisson distribution [25]. This identification is made obvious by the observation that the emission or lack of emission of a gamma ray from a radioactive isotope nucleus is a binary process with a known average rate (for a given isotope) within a given time period. The number of counts received within a certain time period is therefore drawn from a Poisson distribution with argument $\lambda = I$, where I is the average number of counts for that period of time. The discrete probability of measuring exactly k counts from a source with λ average counts is therefore given by the following equation:

$$f(k, \lambda) = \frac{\lambda^k e^{-\lambda}}{k!}.$$

The Poisson distribution has only one argument, λ, which is both the mean rate of occurrence of the rare event being modeled and its variance. It is useful to note that as λ becomes large, the Poisson distribution approximates the Gaussian distribution (with $\mu = \sigma^2 = \lambda$) on which many RBEs rely. Figure 3.1 shows the similarity between a Poisson distribution with $\lambda = 5000$ and a Gaussian distribution with $\mu = 5000$ and $\sigma = \sqrt{5000}$.

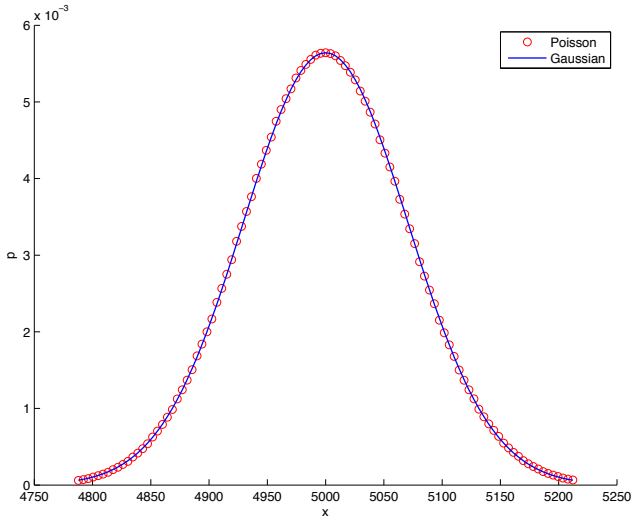


Figure 3.1: The Poisson distribution approximates the Gaussian distribution for high values of λ.

3.2 Recursive Bayesian Estimators

Recursive Bayesian estimators (RBEs), also known as Bayes filters, are, as expected, based on Bayes' rule, which is a tool for calculating the conditional probability of a particular proposition A given a second proposition B. It is expressed simply:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}.$$

The statement of the rule makes clear its primary purpose: the calculation of $P(A|B)$ from $P(B|A)$, $P(A)$, and $P(B)$. An RBE employs this relationship to calculate an estimate of the current state of a system given the previous (estimated) state, the most recent control input, and the most recent measurement of the state. A full, rigorous definition of RBEs can be found in many sources, notably Thrun [36], so this section offers only a basic introduction

with simplified notation to build a foundation on which to expand.

It is useful to note that Bayes' rule may be conditioned on an additional variable: in the case of an RBE, the state estimate is conditioned on the control input and measurement. In this case, Bayes' rule may be rewritten as follows:

$$p(x|z,u) = \frac{p(z|x,u)p(x|u)}{p(z|u)}.$$

Every RBE comprises two basic steps:

Prediction step

The current state x_t is predicted from the previous state x_{t-1} (known as the prior distribution, or prior) and the current control input u_t using the theorem of total probability:

$$\tilde{\hat{p}}(x_t) = p(x_t|u_t)$$
$$\tilde{\hat{p}}(x_t) = \int p(x_t|x_{t-1},u_t)\hat{p}(x_{t-1})dx_{t-1}.$$

In words, the predicted state is the integral of the product of the probabilities associated with the previous state estimate and the probabilities of transitioning from each of those previous states to the next state.

Correction step

The prediction $\tilde{\hat{p}}(x_t)$ is modified by the current measurement z_t to form the posterior

distribution $\hat{p}(x_t)$ based on Bayes' rule:

$$\hat{p}(x_t) = \frac{p(z_t|x_t)\tilde{\hat{p}}(x_t)}{p(z_t|u_t)}.$$

It is important to note that $p(z_t)$ in the denominator is conditionally independent of both u_t and x_t, so the entire denominator can be replaced with a normalizing factor that ensures that $\hat{p}(x_t)$ has a unit integral, satisfying the requirement that it be a probability density function (PDF).

These two steps are performed for each hypothetical state x_t; that is, for every state for which it is desired to calculate a probability. The result is a complete posterior distribution. The entire computation takes place during every iteration of the filter; this algorithm is the recursive Bayesian estimator.

If previous iterations are examined recursively, it becomes clear that at some point a first prior, $\hat{p}(x_0)$, must be chosen. If some information about the system is known *a priori*, an informative prior may be derived; however, choice of the initial prior is frequently drawn from the uniform distribution to indicate a complete lack of knowledge.

It is important to note that the above steps rely on the simplifying Markov assumption. That is, it is assumed that the current state estimate encodes all of the previous state estimates, control inputs, and measurements such that

$$p(x_t|z_{1:t}, u_{1:t}) = \frac{p(z_t|x_t)\tilde{\hat{p}}(x_t)}{p(z_t|u_t)}.$$

The crucial consequence of this assumption is that the filter needs only the previous state estimate, the current input, and the current measurement to perform its computation, instead of retaining all data for each past iteration. The reduction in memory and computational requirements is significant in most cases.

When the system estimated by an RBE is linear and the distributions are Gaussian, this process is the famous Kalman filter. Common RBEs such as the Kalman filter and its siblings the extended Kalman filter (EKF) and unscented Kalman filter (UKF) are utile for non-linear or non-Gaussian distributions to a greater or lesser extent. Specifically, the standard Kalman filter represents the state space as only means and covariances, so it relies on the Gaussian nature of the underlying distribution to function properly. The EKF linearizes the underlying process model about the operating point via the Jacobian and thereby allows the estimation of nonlinear systems, while the UKF relaxes some of the non-Gaussian requirement. Both expansions, however, struggle to fully model a highly nonlinear and non-Gaussian system such as the estimation of the position of a radioactive source. To do so requires employing one of the more general RBEs: the particle filter and the grid-based method.

The particle filter, as discussed in Section 2.2, processes generalized models and distributions by defining a set of particles which take on randomized prior states and whose positions (and therefore the associated states) and weights are updated via a set of rules based on the above process. Thus controlled, the particles cluster around the true state regardless of the nature of the underlying process. The grid-based method defines a lattice of potential states (for example, a grid of (x, y) positions) and calculates a prior and posterior probability for each;

instead of particles clustering around the true state, the posterior probability distribution forms a peak around that state. In both cases, the above prediction and update rules are modified to apply to discrete particles or lattice points instead of a continuum of states. In particular, the prediction step becomes

$$\tilde{\hat{p}}(x_t) = \sum_{x_{t-1}} p(x_t|u_t, x_{t-1})\hat{p}(x_{t-1}).$$

More information about the particle filter may be found in Thrun [36] and Brewer [7]. This thesis' treatment of the grid-based method is found in Chapter 4.

3.3 Hough Transform

The Hough transform is conceptually simple. It will be helpful to begin with the transform for finding lines. Consider a plane with the simplest possible line on it: three points. For each of those points, a number of lines (parameterized by an angle θ and distance from the origin r) is drawn through it, at different angles. If any of these lines intersects one or more of the points, the value associated with that particular line is incremented. (This implies that each line will have a value of at least one, as it must pass through the point under consideration.) After all of the points have been treated in this way, the value associated with each line gives some measure of likelihood that the line exists in the image. In this case, the line that passes through all three points has a value of three, while all the others have a value of only one, making the line that actually exists the most likely candidate. The

entire process is shown in Figure 3.2, and the results are shown in Figure 3.3, where the larger circle indicates a higher weight for the line at $(0, 135°)$.

Obviously, in a photograph or video frame, the number of pixels in question would be significantly higher and the measurements would be noisier, giving rise to a need to eliminate spurious lines by analyzing the distribution of result values. In such a scenario, an edge finding algorithm would simplify the image to aid in the interpretation of the Hough transform results. In the work presented in this thesis, the points considered are generated individually, so no such manipulation is required.

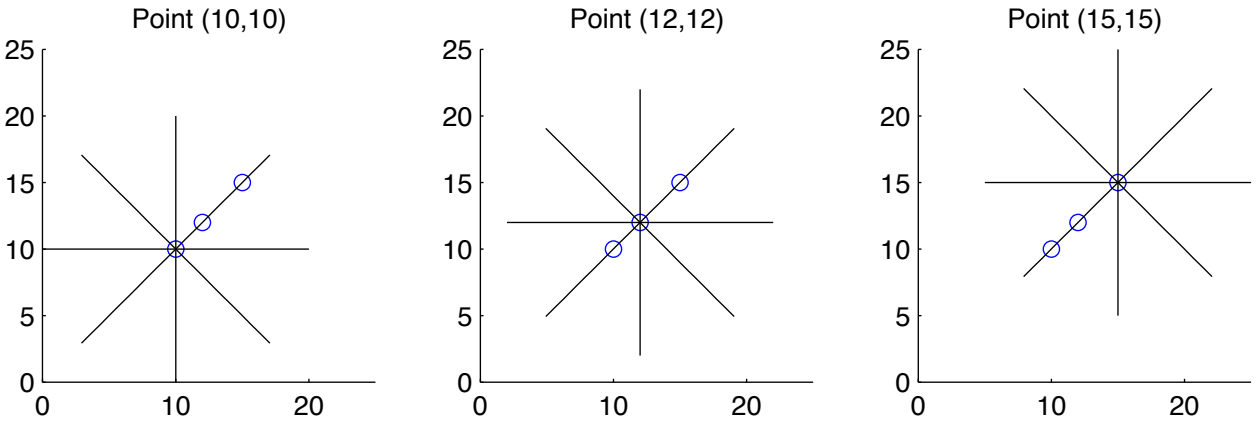


Figure 3.2: Lines drawn through each of three points at 45° intervals

A similar process is followed for localizing circles, as presented in Chapter 5. Circles are parameterized by (x, y, r) tuplets instead of (r, θ) tuplets, but the principle holds: for each point in an image under consideration, some number of circles is generated at certain radii about that point. Each pixel intersected by a circle has its value incremented at the current radius—that is, if a 10-pixel circle is drawn about $(0, 0)$, then the all of the tuplets $(x, y, 10)$

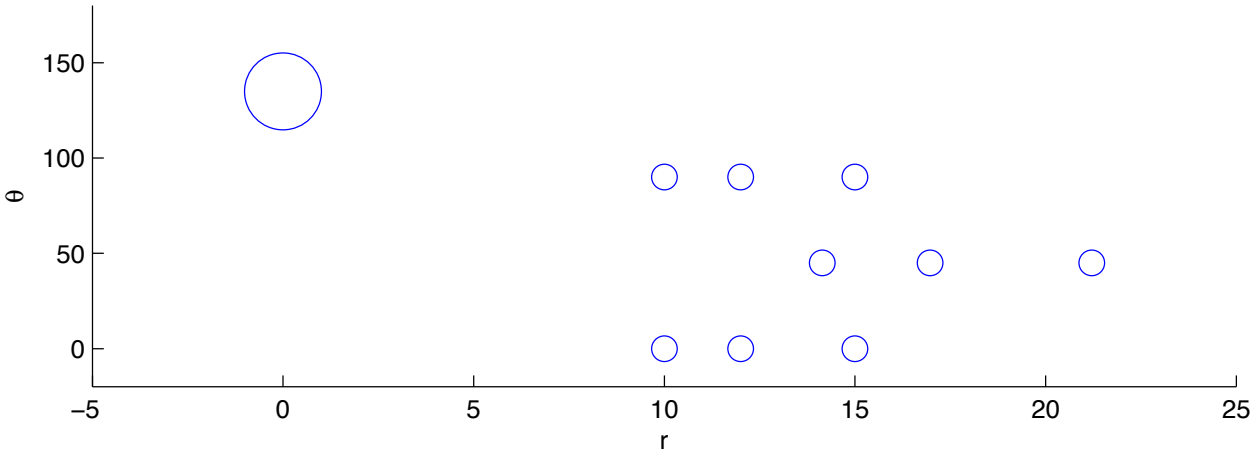


Figure 3.3: Relative weights of (r, θ) line tuplets

such that $\sqrt{x^2 + y^2} = 10$ would be incremented by one. This method of incrementing tuplets is the reverse of the example with lines; the change significantly reduces the computational time required when the number of pixels to be considered as potentially part of a circle is much less than the total number of pixels in the image. This algorithm is demonstrated in Figure 3.4, in which it is clear that the center point $(10, 10)$ is intersected by every circle of radius 10, implying that the tuplet $(10, 10, 10)$ has the highest value. It should also be clear that even if half of the points were removed, the algorithm would still accurately locate the true center point.

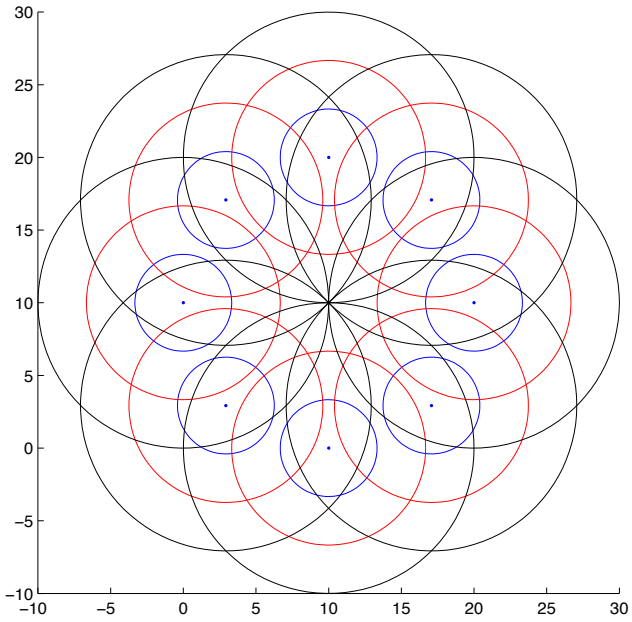


Figure 3.4: Eight points with circumscribed circles of radius 3.33, 6.67, and 10; the tuplet representing the true center point and radius, $(10, 10, 10)$, clearly has the highest value.

Chapter 4

Grid-Based RBE Localization

The immediate evolution of the particle filter to eliminate the drawbacks of its heuristic nature is the grid-based recursive Bayesian method. Instead of using a random distribution of particles and updating their states according to a set of tuning parameters, this procedure uses a static set of evenly-spaced grid points and updates only their probabilities at each time step. This chapter discusses the function of this algorithm (Section 4.1), gives an explanatory example (Section 4.2), and presents the results of implementing it to localize a single point source of known intensity (Section 4.3). The MATLAB code that implements the algorithms developed in this chapter is shown in Appendix A.

4.1 Grid-Based RBE Algorithm

The basic operation of a recursive Bayesian estimator (RBE) is discussed in Section 3.2. This section highlights only the specific operation of this type of RBE in the context of localizing a radioactive source.

4.1.1 Grid Size Selection

To employ a grid-based RBE, a search field must be selected and discretized into a number of grid cells. The grid resolution should be chosen according to the needs of the search (for example, if the source is thought to be on a vehicle, centimeter accuracy is unnecessary, as an estimate within two meters will effectively localize the vehicle) and the available computational power. The relationship of the number of grid points to the computation time per iteration is shown in Figure 4.1. The relationship of the logarithms of the number of grid cells to the computation time per iteration is clearly linear; the least-squares linear regression is shown as the solid line in the figure.

On a dual-core 2.4GHz Intel® Core™ 2 Duo processor with 4GB of RAM, the relationship is expressed as follows:

$$timePerIteration = e^{(0.9324 \log(numCells) - 12.2653)}$$

with linear correlation coefficient $R^2 = 0.9944$ (raw) and $R^2_{adj} = 0.9938$ (adjusted for degrees

of freedom). These values indicate extremely good fit, as expected. Obviously, for extremely large grids, the computer will need to have enough RAM to avoid swapping during the computation, effectively limiting the grid size to $\frac{\text{available RAM}}{8B \cdot \text{number of cells}}$; violating this rule will result in extremely large computation times as the processor reads and writes from its hard drive.

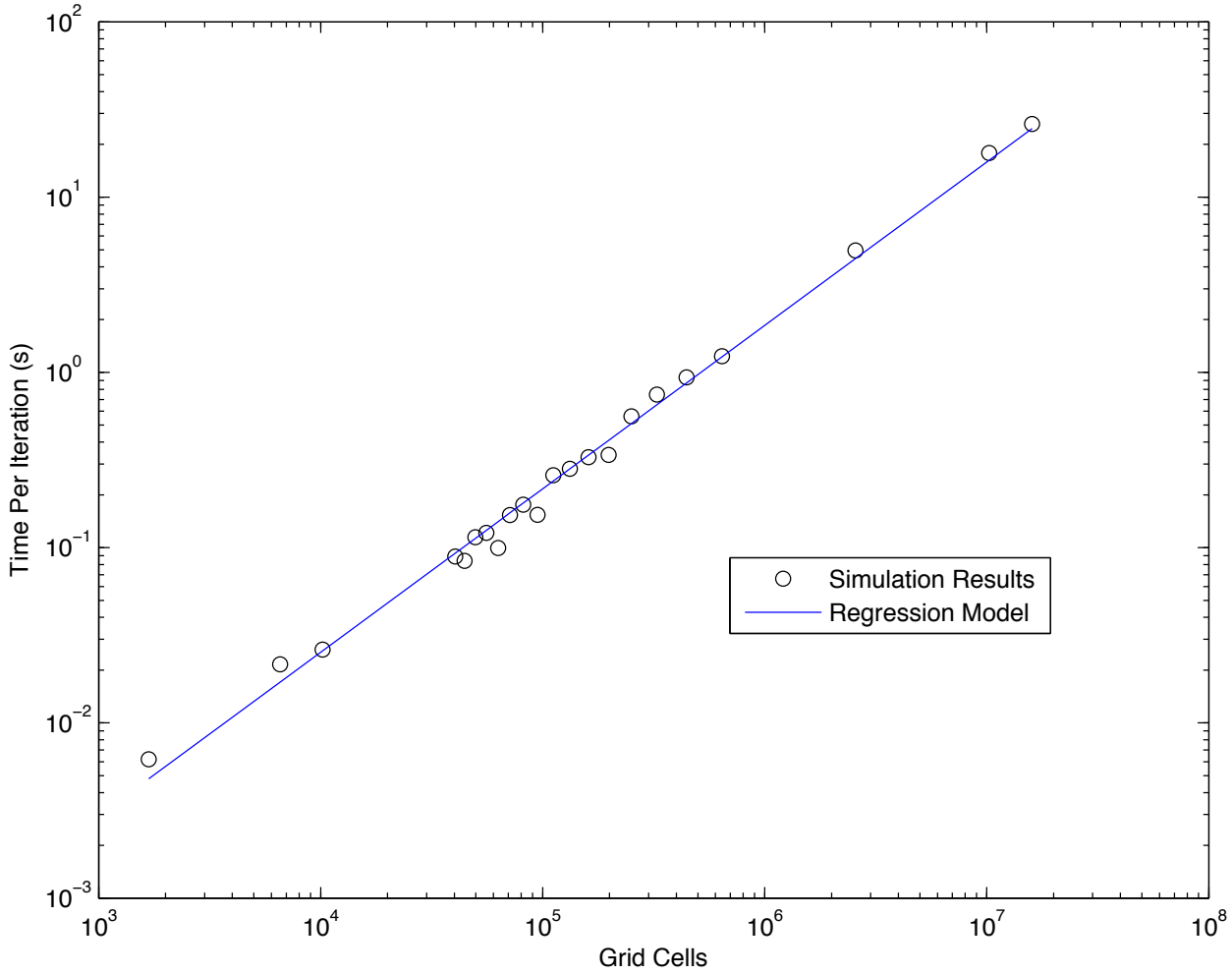


Figure 4.1: Iteration time as a function of the number of grid cells in 2D grid-based RBE localization

4.1.2 Localization Algorithm

The prior is drawn from the uniform distribution such that the sum is 1; that is,

$$prior_{i,j} = \frac{1}{i_{max} j_{max}} \; \forall \; i, j.$$

With the prior grid established, Algorithm 1 shows the process of localizing the source. Note that the detection model used for all simulations in this chapter, GETSOURCECOUNTS, is a procedure provided by SNL to estimate the mean number of counts expected to be observed given a sensor location and the location, isotope, and activity of the source. The procedures POISSPDF and POISSCDF return the PDF and CDF, respectively, of the Poisson distribution given a value x and Poisson parameter λ.

At each iteration (which occurs each time a new observation is available from the radiation detector, or when the previous iteration concludes, whichever comes second), the observation is checked against a known background count level for the area of interest and altitude of the sensor (line 3). If the observation is higher than the background, it is assumed the source has been detected. In this case, the expected number of counts for each grid cell, *assuming the source is in that cell*, is calculated (line 5). The likelihood that the source is in that cell is then the probability that the true observation was drawn from a Poisson distribution with a mean of the expected counts for that cell (line 6).

If the observed count level is less than or equal to the background count level, the source is assumed not to have been observed. The expected count level for each cell is again computed,

but the next calculation is different. Because the source was not detected, the likelihood is computed by the information provided by that fact. Simply, if the sensor did not detect the source, the source is most likely not in a cell within the maximum detection range of the sensor. The likelihood for those cells is therefore set very low, while likelihood for all other cells is set at or close to one to avoid changing the prior value when no new information is available. This calculation is accomplished by computing the probability of detection for each cell, which is the probability that the observation would be less than or equal to the background count level if the source were in fact in that cell (line 11).

The likelihood is one minus that value (line 12), as shown by examining the extreme values. If the probability of detection for a cell is 1, the sensor definitely would have detected a source in that cell; because it did not, the source is definitely not in that cell, so the likelihood is 0. If the probability of detection for a cell is 0, the sensor could never have detected a source in that cell; because it did not detect the source, it has no information about the presence or absence of a source in that cell, so the likelihood is 1.

The prior is updated by the likelihood grid and re-normalized so it remains a PDF. Because the value for each cell in the prior is the probability that the source is in that cell, the cell with the highest value is the most likely location for the source given the information collected so far. That cell's probability is also the algorithm's confidence in its estimate: because the sum of all the cells is 1, a low value indicates either a few other cells with similarly high probabilities or many cells with lower, but non-zero, probabilities. The former scenario suggests several likely locations for the source; the latter indicates a lack of precision, or

Algorithm 1 2-D Grid-Based RBE Localization

```
repeat
  observedCounts ← current observation
  if observedCounts > background then                    {the source is detected}
    for all cell ∈ grid do
      expectedCounts ← GETSOURCECOUNTS(sensor, cell)
      cellLikelihood ← POISSPDF(x = observedCounts, λ = expectedCounts)
    end for
  else                                                   {the source is not detected}
    for all cell ∈ grid do
      expectedCounts ← GETSOURCECOUNTS(sensor, cell)
      probabilityOfDetection ← 1 − POISSCDF(x = background, λ = expectedCounts)

      cellLikelihood ← 1 − probabilityOfDetection
    end for
  end if
  prior ← prior · gridLikelihood                         {update prior}
  prior ← prior / SUM(prior)                             {normalize prior to PDF}
  confidence ← MAX(prior)
  sensor ← MOVESENSOR(sensor, target)
until confidence ≥ 90%
```

"fuzziness," in the estimate. Further observations will correct both issues and increase the confidence calculated by the algorithm.

4.1.3 Sensor Control Algorithm

With a mobile sensor, it is desirable to move the sensor after each iteration in a manner that maximizes the information gained in the next iteration. It is intuitive, especially after observing several simulations of grid-based localization, that moving orthogonally, in some sense, to the source estimate should accomplish this improvement. In addition, it is easily observed that a sensor moving along a straight line can create "ghost" estimates, as a nondi-

rectional sensor cannot distinguish between its left side and its right. Moving orthogonally to the vector between the sensor and the source estimate maximizes the difference between the likelihoods at the positions of the true and "ghost" estimates; this motion is equivalent to moving along a path tangent to the circle centered at the source estimate and passing through the sensor position. Brewer clearly demonstrates this effect with regard to a particle filter; the extension to the grid-based method is obvious [7]. The procedure to produce such a path is Algorithm 2.

The radius correction term *radiusCorrection* (line 4) adjusts the control velocity for the sensor to move it toward the circle of radius *radius* while still moving tangentially to $centerVector$. There are two special cases: *a*) When the sensor is directly above the source position estimate, "tangentially" has no meaning, so the algorithm arbitrarily moves the sensor to the right so the next iteration can apply tangential control; and *b*) When the sensor is directly to the right or left of the source estimate, the tangent is infinite, so the algorithm moves the sensor directly forward or backward, whichever continues a counter-clockwise tangential path. (The counter-clockwise direction is arbitrary; moving clockwise would be as effective.)

4.2 One-Dimensional Example

The principles and algorithms in the previous section apply to a grid-based localization search of arbitrary dimensionality. Simplifying the scenario to a one-dimensional search along the

Algorithm 2 Tangential Circling: moveSensor

Require: $centerVector$ is the vector from the sensor's current position to the desired center of the circling pattern

Require: $maxVelocity$ is the maximum velocity of the sensor

Ensure: $control$ is a vector of (x, y) velocities that will move the sensor counterclockwise along, or toward, the desired circling pattern at radius $radius$

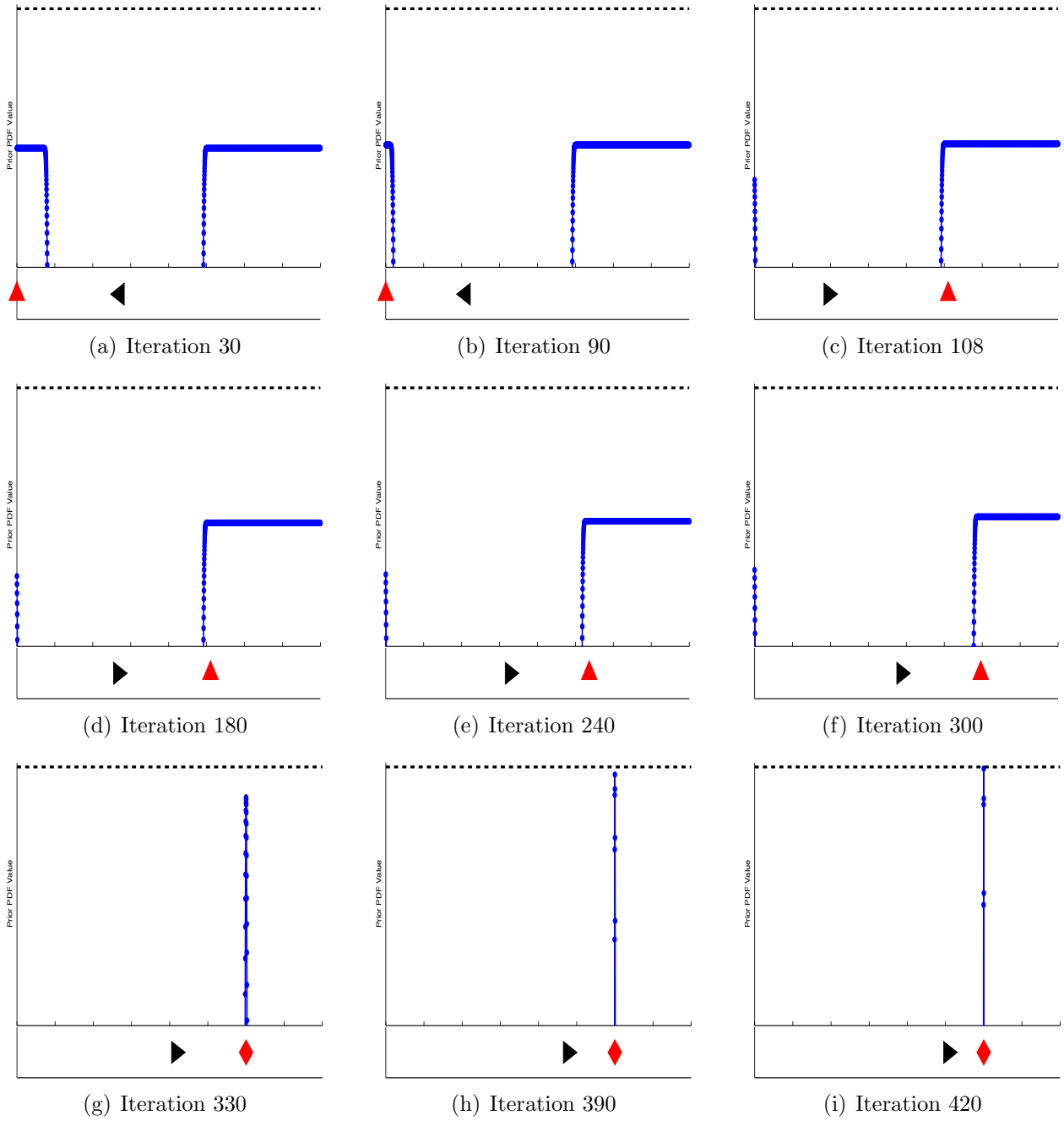
Ensure: $||control|| = maxVelocity$

if $||centerVector|| == 0$ **then** {sensor is on top of the source}
  $control \leftarrow (maxVelocity, 0)$ {arbitrarily move right}
else {we need to circle the target}
  $radiusCorrection \leftarrow 0.1(||centerVector|| - radius)$
  if $centerVector.y \neq 0$ **then**
    $t \leftarrow -\operatorname{sgn}(centerVector.y) \cdot \begin{bmatrix} -1 \\ \frac{centerVector.x}{centerVector.y} \end{bmatrix}$ {CCW vector tangent to $centerVector$}
    $control \leftarrow maxVelocity \cdot \frac{t}{||t||}$ {set $||control|| = maxVelocity$}
    $control \leftarrow control + radiusCorrection \cdot \frac{centerVector}{||centerVector||}$
  else {sensor is exactly horizontal to the source}
    $control \leftarrow (0, -\operatorname{sgn}(centerVector.x) \cdot maxVelocity)$
  end if
end if

x-axis allows an easy-to-follow example.

The source in this scenario is a 1.7 Ci block of ^{75}Se located at +100 on the x-axis of a field that spans $[-200, 200]$ with a resolution of 0.1 m. The sensor starts at -50 (Figure 4.2(a)) and, because of the lack of information, begins by moving left, as indicated by the left-facing triangle. When multiple "maximum" prior values exist, the algorithm arbitrarily chooses the first, which is located at -200 as indicated by the vertical triangle. In this example, the control value is simply 0.5 m/s toward the estimated source location, as "orthogonal" has no meaning in one dimension. Note that the y-axis of each plot in Figure 4.2 is the value of the prior, spaced logarithmically to clearly show the slow increase at the bottom end of the scale as well as the jump to 80% at the end of the simulation.

As soon as the sensor's detection range (defined by the maximum distance at which the received count value exceeds the background count level) reaches the edge of the field (Figure 4.2(b)), the sensor turns around (Figure 4.2(c)) to explore the right-hand side of the field. At that point, the entire left-hand side of the field has been explored without detecting the source, so the source position estimate is moved to the new most likely location. As the sensor explores the field and eliminates possibilities, the estimate continues to move until the source is actually detected (Figure 4.2(g)) as indicated by the symbol changing to a diamond. Note that, before the source was detected, the value of most of the cells in the prior steadily increased to maintain a sum of 1. Then in Figure 4.2(g), the source is detected and all of the cells outside the detection range are immediately eliminated as possibilities, leading to a sharp jump in the probabilities associated with the non-zero cells of the prior. The sen-



(a) Iteration 30 (b) Iteration 90 (c) Iteration 108

(d) Iteration 180 (e) Iteration 240 (f) Iteration 300

(g) Iteration 330 (h) Iteration 390 (i) Iteration 420

Figure 4.2: One-Dimensional Grid-Based RBE Localization Example

sor continues to move toward the estimated source location until the calculated confidence reaches 80%, as indicated by the dashed line (Figure 4.2(i)).

The evolution of the source position estimate and its error and confidence are shown in Figure 4.3. As expected, the confidence stays negligibly low, but increasing, until the source is detected for the first time, at which point it jumps from 0.1% to 6% and rapidly increases as the estimate becomes more sure. The final estimate error is exactly 0 m, as expected when the source lies exactly on a grid cell. This exactness does not occur when the source lies between grid cells; in that case, the two surrounding cells "split" the confidence (which remains significantly lower than 100%), and the error is the distance from the closest cell to the true source position.

When the source is first detected, the source position estimate and error plots appear to indicate that the correct source location had already been rejected. This apparent discrepancy occurs because, immediately before the source is detected, the cells on the very edge of the detectable area become less likely to contain the source. Because there is still no knowledge of the cells on the other side of the source, the prior has a greater value at those points until they are explicitly rejected by the filter. This phenomenon is a feature of the grid-based method: as long as no information is available about a cell and the source has not been detected, any cell that has not yet been scanned is as likely as any other to contain the source. As a result, no cell is prematurely rejected.

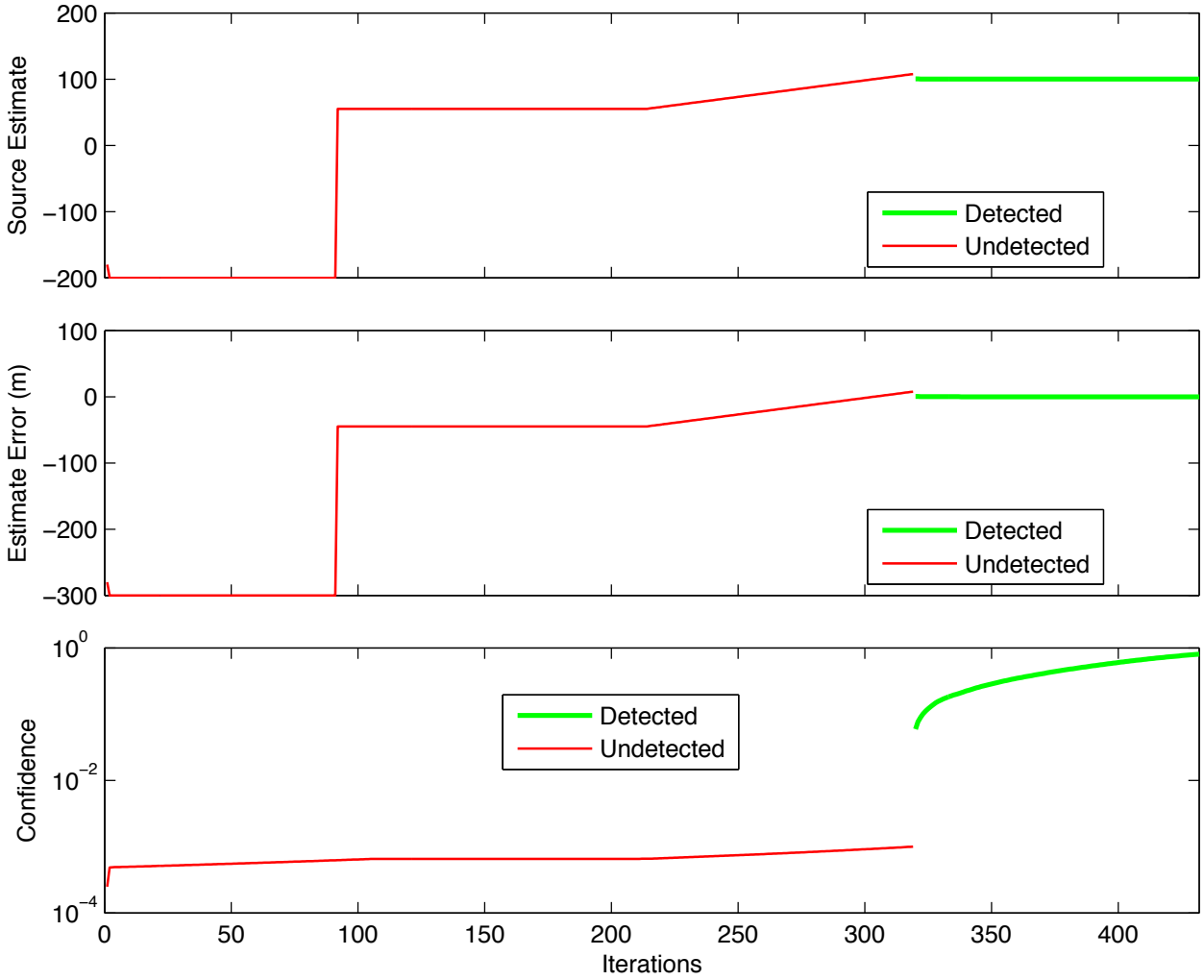


Figure 4.3: One-Dimensional Grid-Based RBE Localization Results

4.3 Two-Dimensional Simulation Results

In the absence of availability of unshielded radioactive sources against which to test this algorithm, a stochastic simulation was developed to test the algorithm against randomized source locations and initial sensor positions. In addition, the same trials were carried out for several different isotopes: ^{75}Se at 1.7 Ci, ^{60}Co at 0.3 Ci (about half as strong as the selenium source), and ^{192}Ir at 6.7 Ci (about twice as strong as the selenium source). The background radiation detected by the sensor was set at a constant 1000 counts, a reasonable average value for an open field in southwest Virginia. For all trials, the field was a 160 000-square-meter area with a resolution of 1 m for a total of 160 801 grid cells. The source was randomly placed exactly on a grid cell within a 100 m radius of the center to ensure detection during the localization flight, and the sensor was initially located randomly without restriction within the search area. The sensor velocity was limited to 4 m/s, which is the maximum horizontal velocity of the Yamaha RMAX. The helicopter can move at 10 m/s straight forward, but in these simulations, it was allowed to move in any direction, necessitating the constraint. The sensor altitude was maintained at 60 m for all simulations.

Of interest in these simulations is the number of iterations required to completely localize the source after the source is initially detected. "Completely localize" in this case is defined as having greater than 90% confidence in the solution, a situation guaranteed in these simulations by placing the source exactly in a grid cell. A real localization flight would not have knowledge of the source's true location, so it could not check its error and would have to

rely on the confidence measure; however, simulation allows error calculation, so those results are reported as well. The number of iterations between the beginning of the search and the detection of the source is irrelevant to the analysis of the operation of the filter itself.

The simulation was run 500 times for each isotope, and analysis of the results appears in Table 4.1 for iterations from detection to high confidence ($> 90\%$) and Table 4.2 for iterations from detection to zero error. Because the initial sensor and source locations in the simulation were completely randomized, occasionally the sensor started within detection range of the source. The data for these cases is not reported in the tables because initial detection provides a significant advantage over searching for the source. In these trials, the advantage has a mean of 4.66 iterations. The trials analyzed here therefore provide the most "fair" comparisons between trials. The results are also presented graphically in standard box plots: Figure 4.4(a) compares the isotopes with respect to iterations between detection and high confidence ($> 90\%$); Figure 4.4(b) compares the isotopes with respect to iterations between detection and zero error; and Figure 4.4(c) compares the number of iterations taken to achieve high confidence to the number taken to achieve zero error among all three isotopes.

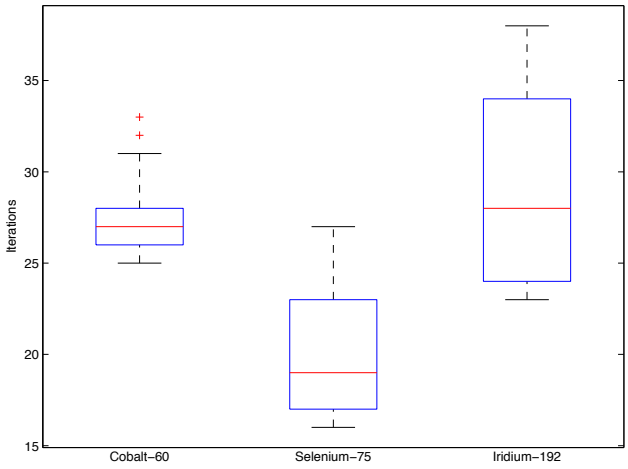
The results reveal several interesting phenomena. In almost 1200 simulations, none took more than 38 iterations between initially detecting a source and localizing it to a very high level of confidence, and most (75%) took 28 or fewer iterations. In a real simulation, these iterations occur at 1 Hz, so most localization would occur within 30 s of detection. The ^{75}Se required significantly fewer iterations to localize than either other isotope, with its worst case near the median case for the other two isotopes. At this time, that result is unexplained:

Table 4.1: 2-D grid-based RBE localization of ^{60}Co, ^{75}Se, and ^{192}Ir at 4 m/s; iterations from detection to confidence > 90%

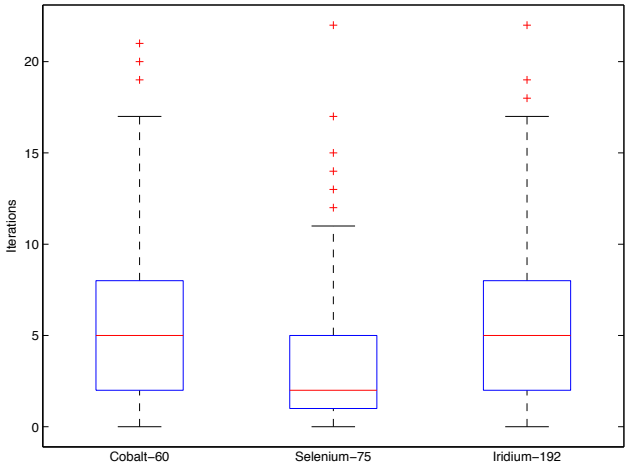
	All Results	^{60}Co	^{75}Se	^{192}Ir
n	1195	478	419	298
Mean	25.07	27.16	19.94	28.94
Std. Dev.	4.90	1.25	3.00	4.68
Median	26	27	19	28
Mode	26	26	19	23
Min.	16	25	16	23
Max.	38	33	27	38

Table 4.2: 2-D grid-based RBE localization of ^{60}Co, ^{75}Se, and ^{192}Ir at 4 m/s; iterations from detection to zero error

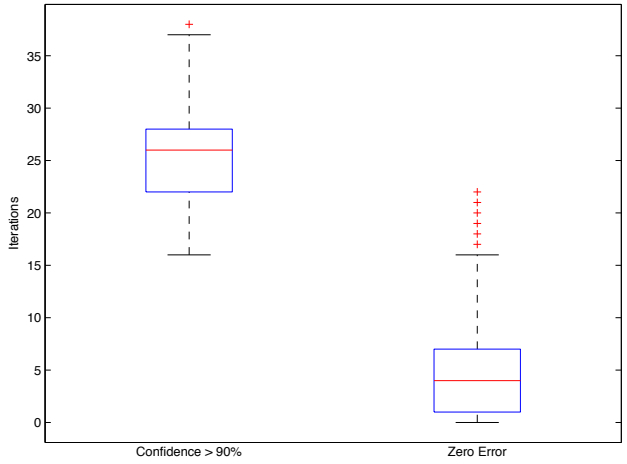
	All Results	^{60}Co	^{75}Se	^{192}Ir
n	1195	478	419	298
Mean	4.74	5.71	3.18	5.36
Std. Dev.	3.93	4.25	3.00	3.88
Median	4	5	2	5
Mode	1	1	1	1
Min.	0	0	0	0
Max.	22	21	22	22



(a) Confidence > 90%



(b) Zero Error



(c) All Isotopes

Figure 4.4: Two-Dimensional Grid-Based RBE Localization Simulation Results

Table 4.3: Maximum Detection Distances for Isotopes Used in Simulation

Isotope	Range (m)
0.3 Ci ^{60}Co	48.67
1.7 Ci ^{75}Se	97.31
6.7 Ci ^{192}Ir	158.06

it may be a result of the relative maximum detection distances for each isotope, which are shown in Table 4.3. The maximum detection distance is the maximum horizontal distance at which a sensor at 60 m altitude should detect the source above a 1000-count background.

The tendency of ^{75}Se to be localized faster than the other two isotopes is also evident, but not as pronounced, when comparing the time to attain zero error. Importantly, all three isotopes had at least one trial in which the algorithm correctly estimated the source position on the same iteration in which the source was detected. More trials attained zero error on the iteration after detection than at any other time, and the mean time to localization is less than 5 iterations. It is interesting to note that the maximum iterations to zero error are nearly equal for each isotope; some trials took up to 22 iterations to reduce error to zero.

The most important observation from the statistics related to achieving zero error appears in Figure 4.4(c): attaining zero error takes many fewer iterations than achieving 90% confidence in the position estimate. In fact, zero error is always attained before 90% confidence, as shown in Figure 4.5. The solid line separates the regions where 90% confidence takes longer (above) and where zero error takes longer (below). This relationship is intuitively correct: if the correct position has $p > 0.5$, it is guaranteed to be the reported estimate (because $\sum_{i,j} p_{i,j} = 1$), but raising that confidence to 90% may take many more iterations. As a

Table 4.4: 2D Grid-Based RBE vs. Particle Filter

	Time (s)		Error (m)	
	Grid-Based RBE	Particle Filter	Grid-Based RBE	Particle Filter
Mean	25.07	45	0	3.7
Std. Dev.	4.90	10	0	1.3
Min.	16	26	0	1.3
Max.	38	69	0	7.8

result, the two events may occur simultaneously, but it is guaranteed that, once the algorithm declares itself finished, the error is zero.

Another insight gained from that plot is that, while it appears there exists a general trend of positive correlation between the iterations taken to achieve zero error and the iterations taken to achieve 90% confidence, the correlation is very low. In fact, it is 0.6822 for ^{60}Co, 0.1301 for ^{75}Se, and 0.1253 for ^{192}Ir, for an overall correlation of just 0.3455. That is, it is nearly impossible to predict, given how long the filter took to converge on a confidence higher than 90%, how much earlier it achieved zero error. All that can be said is that zero error occurred some time before the algorithm could raise confidence to the required level.

The results of these simulations of the two-dimensional grid-based RBE demonstrate that the algorithm is both very fast and very accurate. It represents a significant improvement in both time and error over the particle filter algorithm presented by Brewer [7]. The fastest result he presented took 26 s (at 0.01 s/iteration), and the lowest error was 1.3 m, both of which are superseded by the results presented in this chapter. Complete comparisons are shown in Table 4.4.

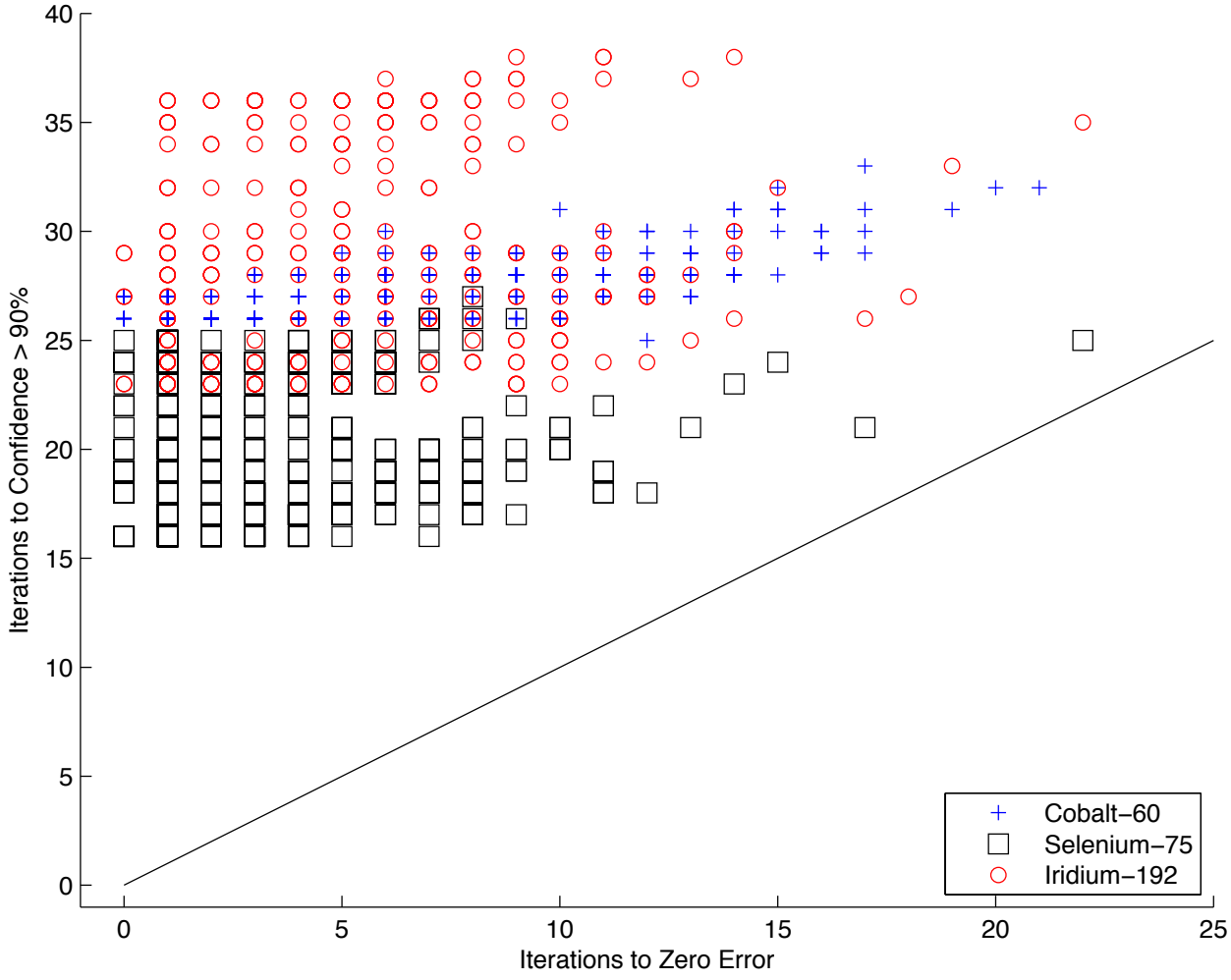


Figure 4.5: Two-Dimensional Grid-Based RBE Localization: Comparison of Iterations to 90% Confidence and Iterations to Zero Error

Important differences exist between the work presented in this chapter and the results presented in Brewer. Because Brewer reported results in seconds, the results above have been converted to seconds. This represents no change in the numbers themselves, because the filter presented here operates at exactly 1 Hz, the update rate of the SNL radiation detector. In addition, the particle filter was tested on a 250 m $\times$ 250 m field, about 40% of the size of the field in the simulations presented above. Brewer's results include the entire period of the trial, while the results above include only the time after first detecting the source. This discrepancy is acceptable because Brewer assumed a zero-count background, so the detection range was effectively infinite and detection can be assumed to have occurred immediately.

Recall that these simulations require prior knowledge of the isotope and activity of the source and that only one source exists for each trial. The technique presented in Chapter 5 removes these constraints.

Chapter 5

Contour Analysis Localization

Removing the prior knowledge constraints imposed by the computation requirements of recursive Bayesian estimation algorithms requires completely re-posing the problem. Instead of considering the localization of a point source via estimating its effects and calculating probabilities, the techniques described in this chapter utilize a mapping approach. Assuming no prior knowledge of the situation dictates an algorithm that intelligently observes the area and draws conclusions about the presence or absence of radioactive sources based on the resulting map. This approach requires three basic steps, as detailed in this chapter: Section 5.2 describes initially locating a particular level set of radiation contours; following the contour to obtain a useful map of the area is described in Section 5.3; and finally, the procedures in Section 5.4 translate the resulting map into source position estimates. Finally, the results of applying the contour analysis of Section 5.4 to the contours generated in Section 5.3 are presented in Section 5.5. The MATLAB code that implements the algorithms

developed in this chapter is shown in Appendix B.

5.1 Algorithm Overview

Before exploring the steps of this algorithm individually, it will be helpful to summarize its operation. The contour analysis localization algorithm, as indicated by its name, operates by mapping level sets of radiation counts and analyzing them to localize radioactive sources.

Consider the field of radioactive intensity created by the presence of one or more radioactive sources in an area. Just as a topographical map presents elevation contours measured in meters, this radiation field exhibits similar level sets in radiation intensity measured in counts per second. Under the assumption that air is a homogeneous medium[1], level contours of radioactive intensity form perfect concentric spheres centered on the center of mass of the source. The intersection of that sphere with a plane, in this case the plane of constant altitude in which the sensor travels, is therefore a perfect circle. Accurately tracing that circle would reveal the location of the source at the circle's horizontal center. Because the relationship between distance and received intensity is known (Equation 3.1), if the received intensity at each point on the circle is known and constant, the intensity of the source may also be estimated.

Adding a second, third, or additional sources to the first changes the contour shape predictably. Specifically, because the contribution from each source is perfectly spherical, the

[1]Changes in the density of air can invalidate this assumption, especially in the presence of changing humidity, differing air pressure as altitude increases, or thermoclines.

Table 5.1: Radioactive source combinations used for analysis

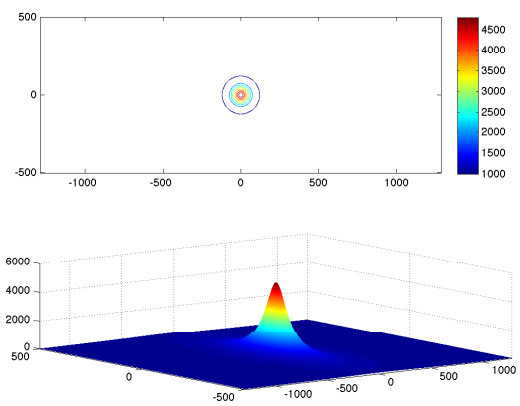
Combination	Intensity (Counts)	Separation (m)	Figure
One Weak Source	5000	—	5.1(a)
One Strong Source	50 000	—	5.1(b)
Two Equal Sources	(20 000, 20 000)	500	5.1(c)
Two Unequal Sources	(50 000, 5000)	500	5.1(d)
Three Unequal Sources	(5000, 10 000, 5000)	(250, 250)	5.1(e)

combined contour can be considered as the sum of those circles with some distortion of each circle by the contribution of the other source. The amount of distortion varies from zero for relatively weak sources far away from each other to intractably extreme for sources of similar positions or extremely disparate magnitudes. (The more intense source will "shadow" the smaller source entirely, making detection extremely difficult or impossible. Changing the altitude of the search will change the shape of the contour and, if the sensor is brought sufficiently close to one source, localization may become possible again.)

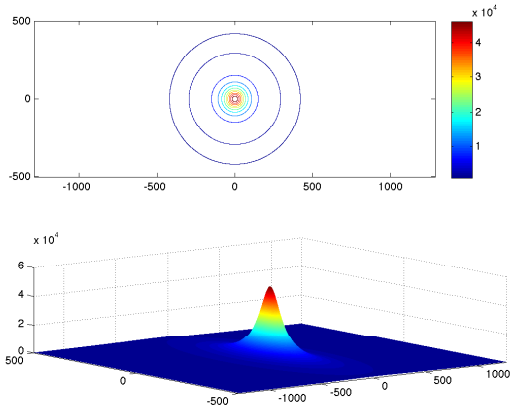
For the purposes of this chapter, five combinations of radioactive sources were simulated using generic sources with strength measured in received counts at 60 m. The combinations are listed in Table 5.1 and their contours are shown in Figure 5.1.

All the contours presented in Figure 5.1 are as seen by a sensor at a constant altitude of 60 m. For comparison among the source combinations, the outermost contour in each image is 1000 counts, and the next inner contour is 2000 counts, but the remainder (all interior contours) represent different count levels for each combination.

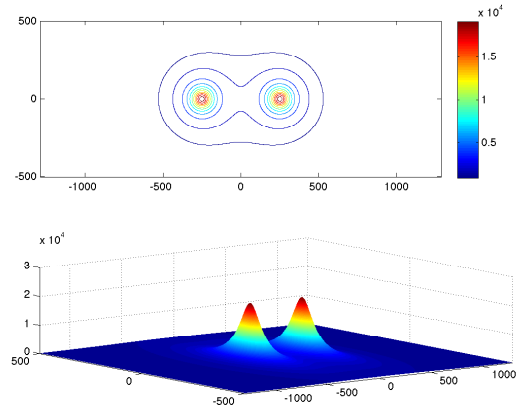
Though this analysis technique removes the major constraints on source localization [the limit



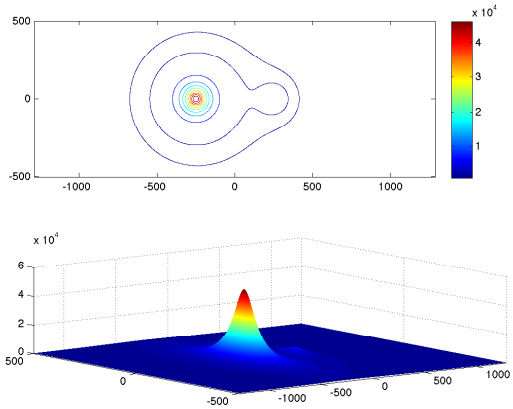
(a) One Weak Source



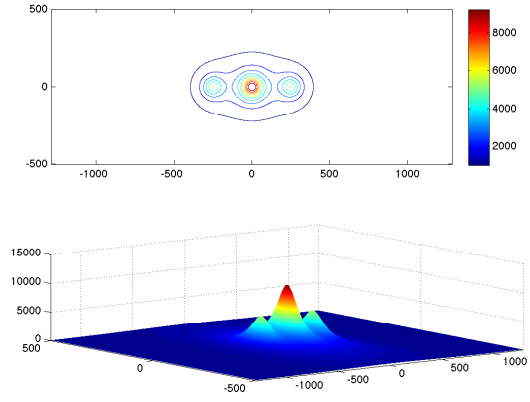
(b) One Strong Source



(c) Two Equal Sources



(d) Two Unequal Sources



(e) Three Unequal Sources

Figure 5.1: Contour and surface plots of radioactive source combinations used for analysis

of one source and the need for prior knowledge of the source's strength (and by extension, its isotope)], one (weak) requirement for prior knowledge remains. Because the algorithm maps a particular contour level, that contour level must be chosen somehow. The choice of contour level has a great effect on the shape of the contour in multi-source situations, as easily seen in Figures 5.1(c), 5.1(d), and 5.1(e).

5.2 Contour Detection

Because of the time-limited nature of the mission prescribed for this system, rapidly detecting the designated contour level is a necessity. This task comprises searching the area of interest using the most efficient flight path in terms of both fuel and time. Both requirements suggest using the vehicle's maximum speed, which is possible only when moving forward. Maintaining this speed requires moving in large, smooth curves instead of more traditional patterns such as the lawnmower coverage pattern and the ladder search pattern. Flight plans explicitly designed by the USL for high-speed coverage of an area for stereo imaging, known as the zamboni pattern, are very inefficient in coverage for a sensor with as large a range as a radiation detector. The obvious choice of path resulting from these considerations is a spiral; specifically, an Archimedean spiral. Parameterized by

$$r = a\theta + b, \tag{5.1}$$

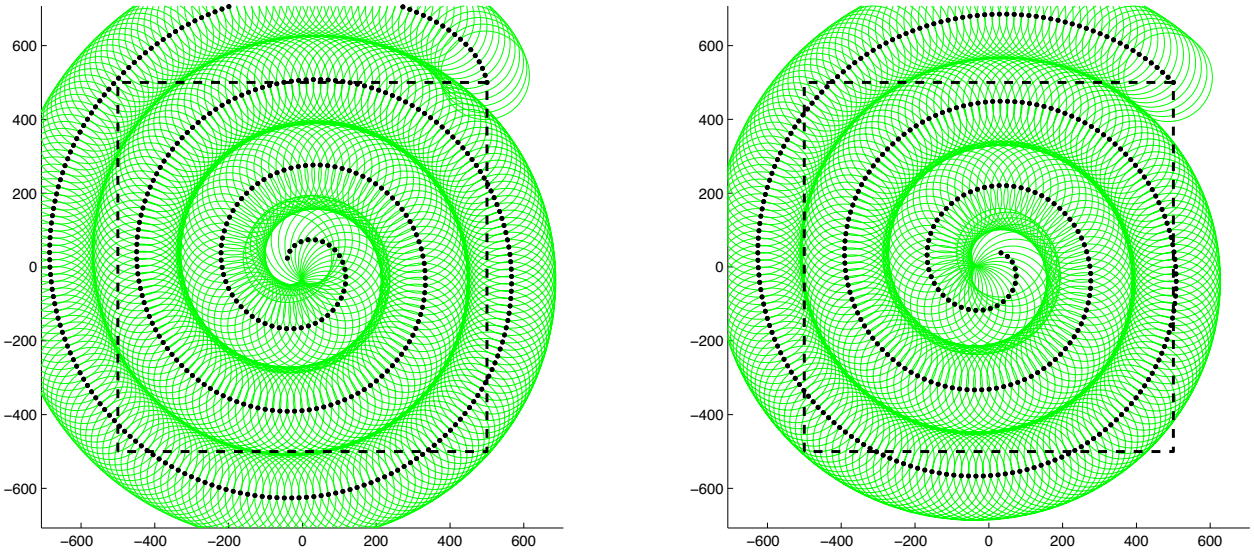
the Archimedean spiral maintains a constant separation between its rings, ideal for a search pattern. The spiral shape also allows the helicopter to move in large, sweeping curves and thereby maintain a high forward velocity.

To implement this search path, the maximum detection range of the sensor for the desired contour intensity is first calculated. The distance between the arms of the spiral is slightly less than twice this distance—uncertainty grows at the edge of the detection range, as seen in Chapter 4, so the swaths will overlap at the edges. Control inputs to the helicopter are developed with the procedure given in Algorithm 2 to make the helicopter circle the center of the area of interest. The radius of the circle is calculated from the angle of the helicopter's position with respect to field, but not in as straightforward a manner as given in (5.1). Instead, a procedure was developed (Algorithm 2) to employ the relationship of the initial location, the maximum necessary radius of the spiral, and the maximum detection range to calculate the appropriate radius at each iteration. The full MATLAB code that implements the algorithm is given in Appendix B.

As an example, Figure 5.2(a) shows a complete spiral path generated with this algorithm with the helicopter moving at 10 m/s attempting to detect a 5000-count source against 1000-count background radiation on a one-square-kilometer field. The black dots show the path of the helicopter, which starts at the top-right corner of the square; the green circles show the detection range of the sensor at each position; and the dashed square denotes the boundaries of the field of interest. Note that the entire field of interest has been observed at some point, and almost all of it has been observed for several seconds. As mentioned above, the edges of

the circles overlap along the curves of the spiral to make up for the lack of certainty at the edges of the detection range. The entire spiral requires 812 seconds, or about 13.5 minutes, to search the entire area; if a source existed in the area, the spiral would have terminated when the source was first detected, and the contour would then be followed as described in Section 5.3.

It is clear that the path in Figure 5.2(a) is not perfectly efficient. In fact, the first several locations do not observe the area of interest at all; this shortcoming is a consequence of requiring that 100% of the field be observed at some point. Relaxing that restriction very slightly permits the path shown in Figure 5.2(b), which is the same scenario without the absolute coverage requirement. Note the small areas on the bottom-right and top-right of the square that are never observed. The benefit of allowing the algorithm to ignore those small areas is that the incomplete coverage spiral can be completed in only 694 seconds, or about 11.5 minutes, nearly two minutes faster than the first path above. A contour would have to be extremely small to exist within the area of interest and avoid detection; such a weak source would not likely be detected above background radiation. Note, however, that the entire contour need not exist within the prescribed boundaries. If any part of it does, the contour mapping algorithm will follow the entire contour irrespective of its size or position.



(a) Exhaustive spiral search, 5000 counts

(b) Quick spiral search, 5000 counts

Figure 5.2: Archimedean spiral search patterns

5.3 Contour Following

Once the specified contour level has been detected, the helicopter must map the entire contour for analysis. Detection is defined as measuring at least $\lambda_d + \sqrt{\lambda_d}$ counts, where λ_d is the desired contour count level. The adjustment is necessary to ensure the algorithm begins on the inside of the contour, and it serves as a safeguard against receiving an anomalously-high count and beginning the contour mapping prematurely.

The contour following algorithm operates in a very simple manner: if a measured count is greater than λ_d, the helicopter turns to the right; if a measured count is less than λ_d, the helicopter turns to the left. The result is that the helicopter follows the contour in a counterclockwise direction. The forward speed of the helicopter is set at the beginning of the test, and should be adjusted by the operator for the size of the search area, the desired

speed of the search, and the desired accuracy and completeness of the finished contour. A faster search will result in less accuracy in the analysis step, but some loss of accuracy may be acceptable for the time saved. A large search area or physically large contours (a low λ_d for a strong source, for example) allows much higher forward speeds, while tight contours (relatively high λ_d) require a slower scan to follow the contour properly. The limiting factor from the helicopter for these considerations is the maximum turn rate. The Yamaha RMAX has a maximum yaw rate of 30°/s, but such a high yaw rate coupled with high forward speeds may result in sideslip, which would draw the vehicle off the contour and result in a lower turn rate than the commanded yaw rate. For this reason, yaw rates in this chapter are limited to 15°/s, which the Yamaha Attitude Control System (YACS) can translate directly into turn rate.

5.3.1 Control Algorithm

The helicopter's turn rate is determined by a proportional-integral-derivative (PID) controller that takes as input the latest received count rate c_k and the contour level λ_d and

returns the desired helicopter yaw rate ψ_d:

$$\begin{aligned}
\sigma_\lambda &= \sqrt{\lambda_d} && \text{(standard deviation)} \\
e_{pk} &= \frac{c_k - \lambda_d}{\sigma_\lambda} && \text{(proportional error)} \\
e_{dk} &= c_k - c_{k-1} && \text{(differential error)} \\
e_{ik} &= \sum_{j=1}^{k-1} e_{ij} + e_{pk} && \text{(integrated error)} \\
\psi_d &= \frac{k_p}{\sigma_\lambda} e_{pk} + \frac{k_d}{\sigma_\lambda} e_{dk} + \frac{k_i}{\sigma_\lambda} e_{ik}.
\end{aligned}$$

Proportional error differs from the simple error from λ_d to adjust for the $1/R^2$ relationship of radiation emission intensity to distance. For extremely high contour levels, an error of as little as 1 m may result in hundreds or thousands of counts of error; an extreme turn will take the vehicle off the contour. Conversely, for very low contour levels, a displacement of several meters may result in tens of counts of error, so the helicopter must react quickly.

The PID constants k_p, k_d, and k_i are modified by σ_λ so that the same constants may be used for any desired contour level. An unconstrained nonlinear optimization method (MATLAB's FMINSEARCH function) was used to generate error-optimal PID constants using the 2000-count contour of Figure 5.1(e) with an update rate of 1 Hz at 1 m/s. The resulting values are presented in Table 5.2.

The controller output ψ_d undergoes several checks before it is applied to the helicopter. It

Table 5.2: Contour following PID controller constants

Constant	Value
k_p	-23.6648
k_d	-11.9835
k_i	-0.0037

is first limited to the maximum yaw rate ψ_{max}:

$$\psi_d = \min\left(\left|\psi_d\right|, \left|\psi_{max}\right|\right) \cdot \operatorname{sgn}\left(\psi_d\right).$$

In addition to constraining the yaw for any individual iteration, the total amount of consecutive yaw allowed is also limited to 180°. This restriction prevents the helicopter, in extreme circumstances, from turning in circles either inside or outside the contour. The vehicle may not execute more than 180° of yaw without crossing the contour; as a consequence, the helicopter must always re-intersect the contour.

To determine when the contour has been completed, the helicopter waits until it has completed at least 180° of yaw from its heading when it first encountered the contour, and then stops when it has achieved 360°, a full rotation. Because all radiation contours are known to be closed, these conditions must be met for every contour eventually.

The complete code to implement this contour following algorithm is presented in Appendix B.

5.3.2 Ideal Simulation

To demonstrate the efficacy of the contour following algorithm, this section presents the results of simulations where the measured counts were calculated exactly according to Equation 3.1. The intensity calculation is reprinted here for convenience:

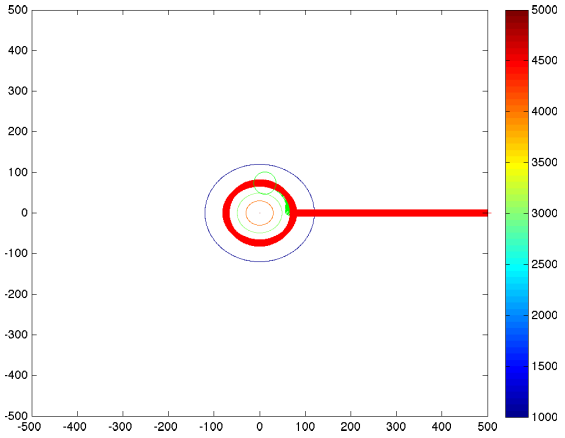
$$\lambda_k(x) = \frac{I}{(x_k - x_0)^2 + (y_k - y_0)^2} + \lambda_b. \tag{5.2}$$

Given an initial ingress path of a straight line, the contour following paths for each source combination introduced in Figure 5.1 are shown in Figure 5.3. Red '+' symbols indicate locations where the sensor received a count level lower than ψ_d, and green '×' symbols indicate locations where the sensor received a count level greater than or equal to ψ_d. Note that in this figure, and all similar subsequent figures, the red and green marks are so close together they appear as solid lines. Six contour lines, representing one, two, and three standard deviations on either side of ψ_d, were added to these figures in addition to those from Figure 5.1. These additional contours are most easily visible in Figures 5.3(c) and 5.3(d) and serve as more precise indications of the slope of the count gradient along the contour being mapped. The crucial observation is that the $1/R^2$ nature of radiation intensity falloff with distance results in extremely steep slopes, making contour following difficult even at low contour levels. That is, a small position error can result high count errors. For each combination, the helicopter started at $(500, 0)$ and the 2000-count contour was flown at 1 m/s with radiation count updates at 1 Hz. For these simulations, the constraint that the

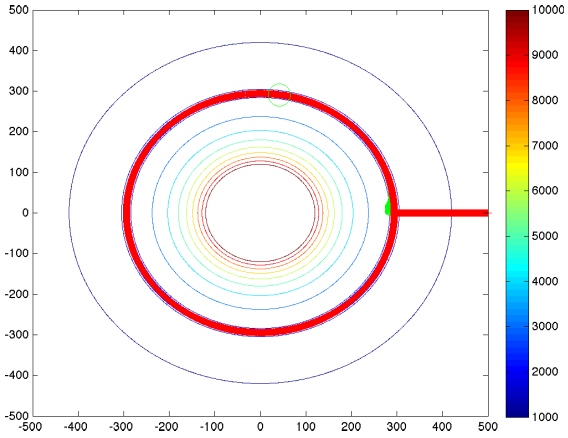
applied turn rate remain below 6°/s was imposed to maintain precise flight paths even at high speeds.

Although the SNL radiation detector provides new count rates at only 1 Hz, the simulations were run at four update rates and ten different forward speeds for comparison purposes. The median error for each simulation is shown in Figure 5.4. It is clear from the plots that faster updates reduce error; this result is expected because updating the helicopter control yaw more often permits more precise following of any flight path, including these contours. In the same way, higher speeds introduce greater error as the helicopter moves farther before corrections occur. Tracking contours with sharp curves at high speeds can induce much greater error in some situations. For example, comparing Figure 5.4(b) with Figure 5.4(d) at 10 m/s and a 10 Hz update rate shows an extreme anomaly associated with the very sharp curve around the smaller source. The slow update rate for the other three cases actually proved a benefit, as the 10 Hz rate compensates too well, resulting in a significantly worse trace as shown in Figure 5.5.

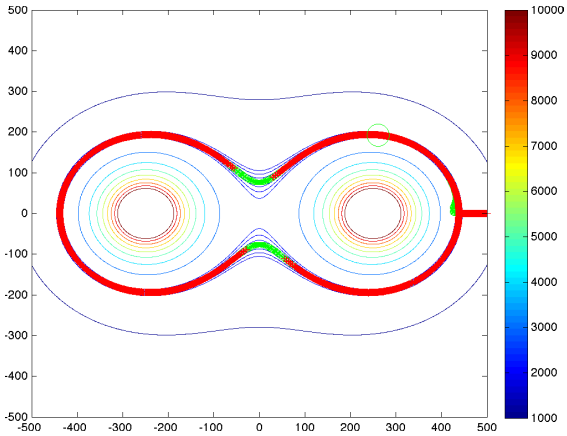
These observations indicate that fast updates and slow speeds result in the most error-free results. However, slower travel also increases the time required to trace the contour, thereby increasing the time before localization can occur. The trace times are roughly inversely proportional to helicopter speeds, as expected. The variations occur when the speed makes following a particular curve impossible, so the paths taken are not identical. Representative times are given in Table 5.3 for each source combination; the values presented are the means over the four update rates simulated. The extreme anomaly in the 10 m/s simulation for



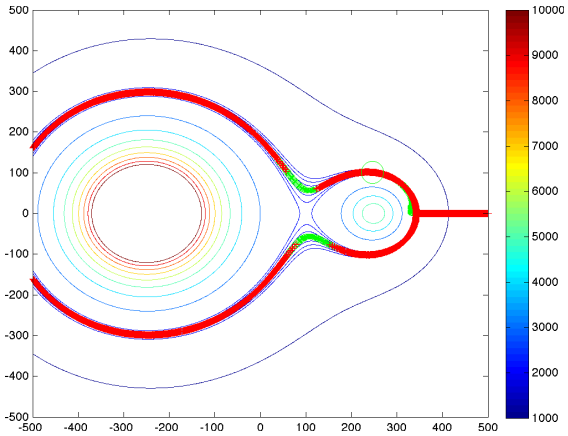
(a) One Weak Source



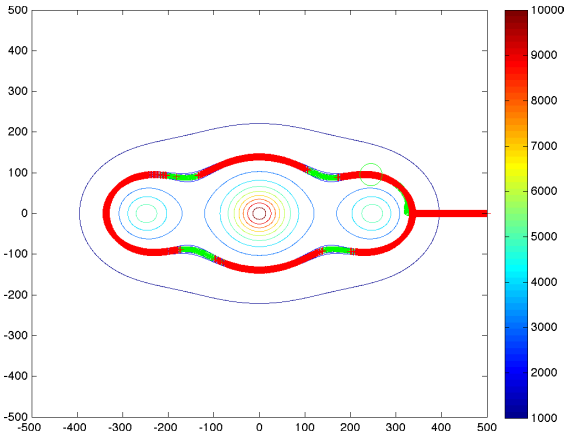
(b) One Strong Source



(c) Two Equal Sources

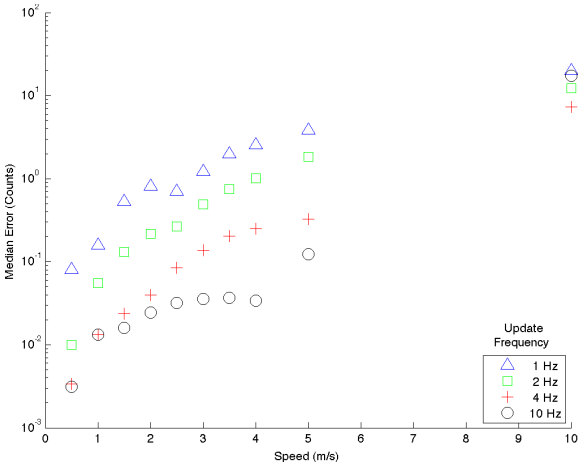


(d) Two Unequal Sources

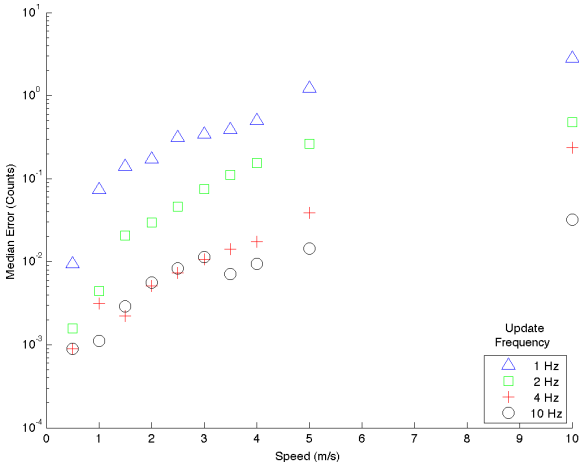


(e) Three Unequal Sources

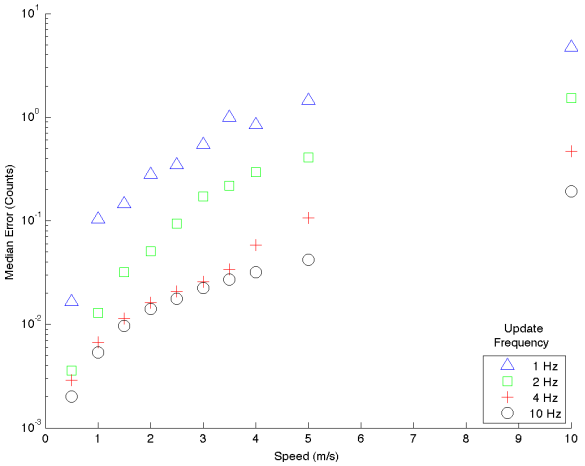
Figure 5.3: Following 2000-count contours of various source combinations. Red + marks indicate $c_k <$ $_d$ and green marks indicate c_k $_d$



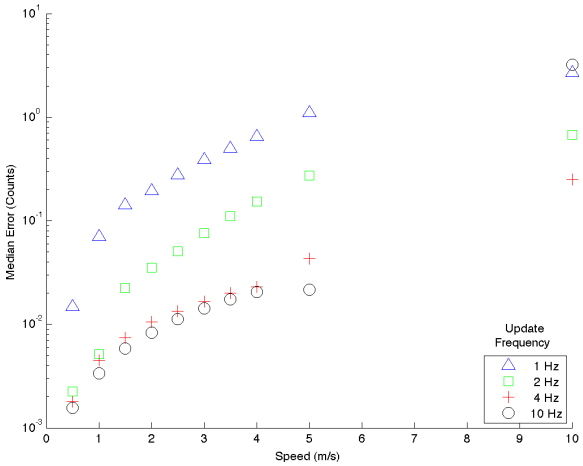
(a) One Weak Source



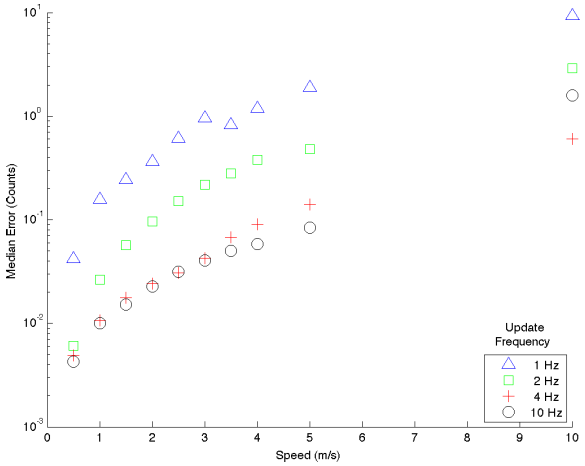
(b) One Strong Source



(c) Two Equal Sources

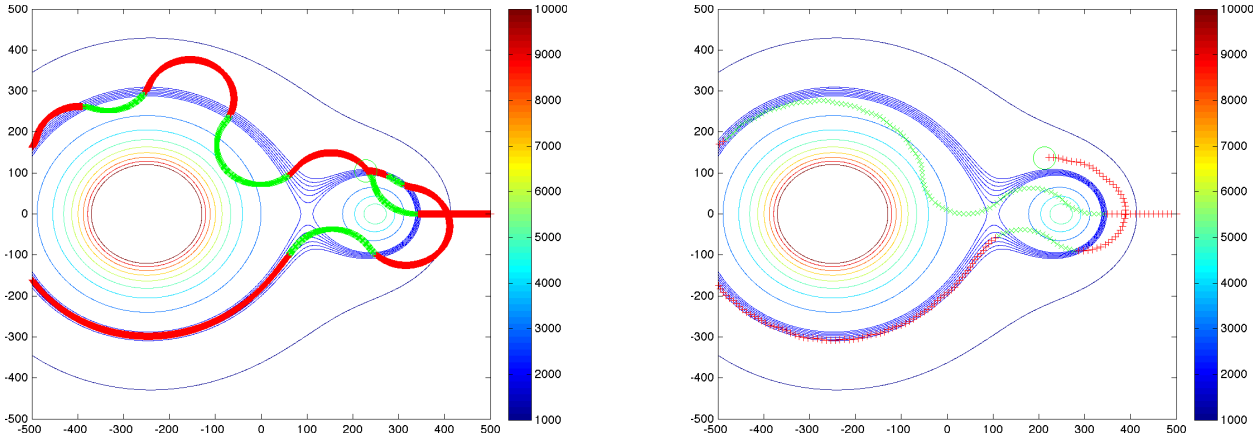


(d) Two Unequal Sources



(e) Three Unequal Sources

Figure 5.4: Median count rate errors during ideal contour following



(a) 10 Hz Update (b) 1 Hz Update

Figure 5.5: Ideal contour following median error anomaly

Table 5.3: Mean times required to follow ideal contours

Combination	Mean Time (s)	
	1 m/s	10 m/s
One Weak Source	584.20	86.64
One Strong Source	2316.50	235.34
Two Equal Sources	2645.58	243.63
Two Unequal Sources	2614.21	280.05
Three Unequal Sources	1763.68	179.88

a single weak source is a consequence of the high speed: the helicopter does not trace the contour well at any update rate.

Recall that for all of the above simulations, the helicopter turn rate was limited to 6 s. Increasing the maximum turn rate solves problems at low speeds and update rates, allowing the helicopter to achieve zero error more quickly and maintain it more closely at low speeds. At high speeds, however, increasing the limit results in overcompensation for error with the PID parameters used. Because errors are higher at high speed, the desired yaw correction is higher; raising the limit on the applied rate allows more extreme correction, which frequently

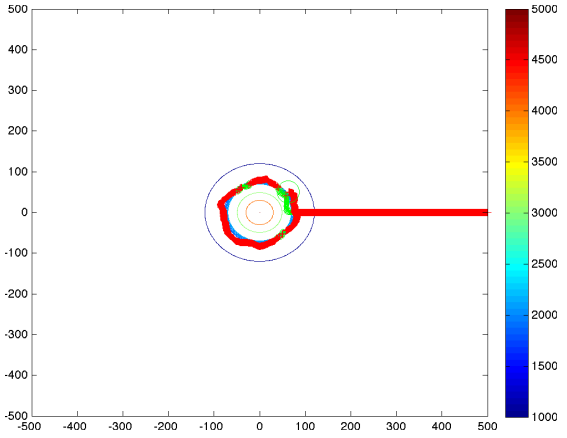
results in overcompensation.

5.3.3 Stochastic Simulation

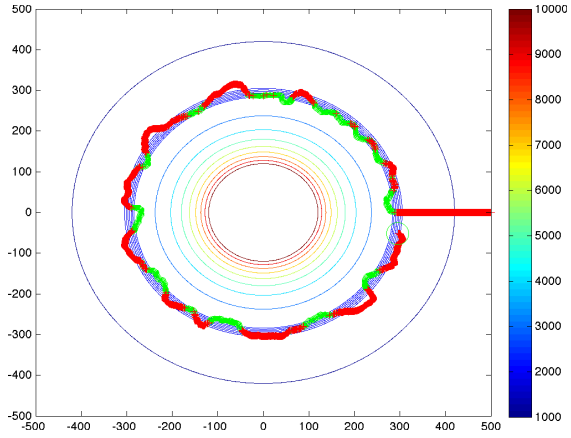
The simulations in the previous section were run again with ideal received count rates replaced by values drawn from the Poisson distribution. If the ideal count rate for a location is λ_r, the count reported for these simulations was $\mathcal{P}(\lambda_r)$. The significantly greater variation in the received counts resulted in much higher error rates for the contours. To give the controller more freedom to correct for the widely-varying responses, the maximum turn rate was increased to 15°/s for these simulations.

The same conditions as the above simulations were maintained: for each simulation, the helicopter started at $(500, 0)$ and the 2000-count contour was flown at 1 m/s with radiation count updates at 1 Hz. Representative examples of the resulting maps are shown in Figure 5.6. Because the received counts are drawn from the Poisson distribution, the map appears different on each run; those shown in the figure are merely samples.

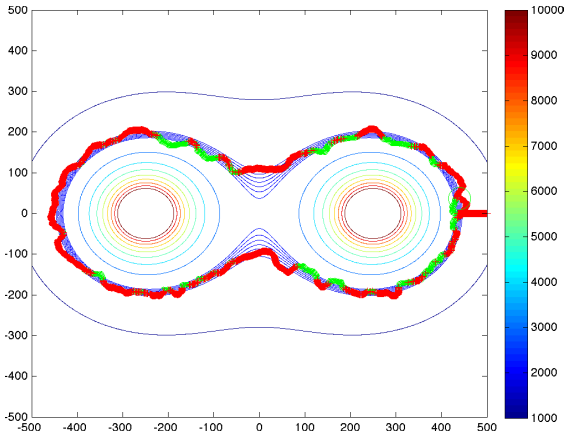
The most prominent observations from these plots are the high displacement error from the true contour and the lack of smoothness in the traveled points. In the ideal simulations presented in Section 5.3.2, the received count at a point is deterministic, so count errors translate directly to distance errors. The same is not true of these simulations, in which the count rate is drawn from a Poisson distribution, so the count rate and position are only weakly correlated. The median displacement errors for the various configurations simulated,



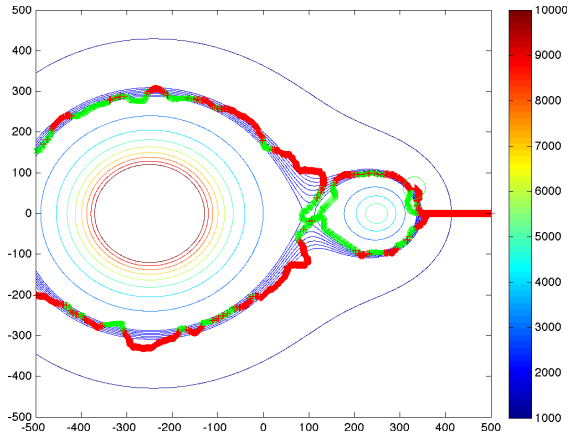
(a) One Weak Source



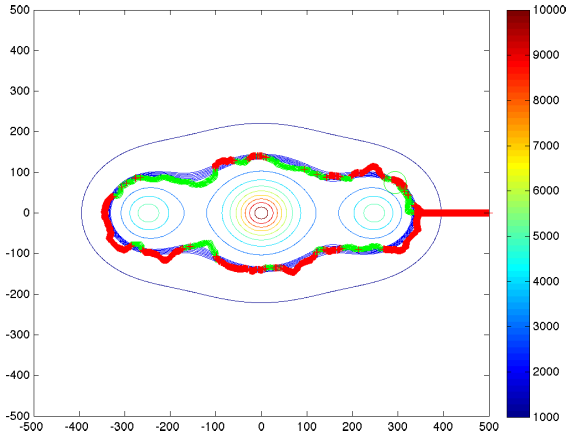
(b) One Strong Source



(c) Two Equal Sources



(d) Two Unequal Sources



(e) Three Unequal Sources

Figure 5.6: Following 2000-count contours of various source combinations with sensor readings drawn from the Poisson distribution. Red + marks indicate $c_k <$ $_d$ and green marks indicate c_k $_d$

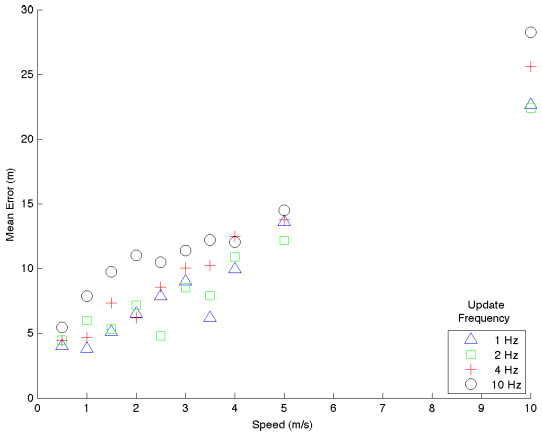
as measured by the difference between the ideal contour and the Poisson-modified contour, are shown in Figure 5.7. Although the data are scattered, a general trend emerges: higher update rates lead to more error. A simple solution exists in the aforementioned lack of smoothness in the maps. The maps in Figure 5.6 all used an update rate of 1 Hz and still described extremely non-smooth paths. Whereas in the ideal simulations, a higher update rate smoothed the path, in this case higher update rate reduces smoothing because physically-close points need not have numerically-close count rates.

The mean distance error is very high in all cases, though some data points exist on both extremes. For example, (1 Hz, 1 m/s) in Figure 5.7(a) has an error below 5 m; (10 Hz, 10 m/s) in Figure 5.7(d) has a mean error of nearly 50 m, rendering the map nearly unusable for determining the source location and entirely useless for determining the source intensity.

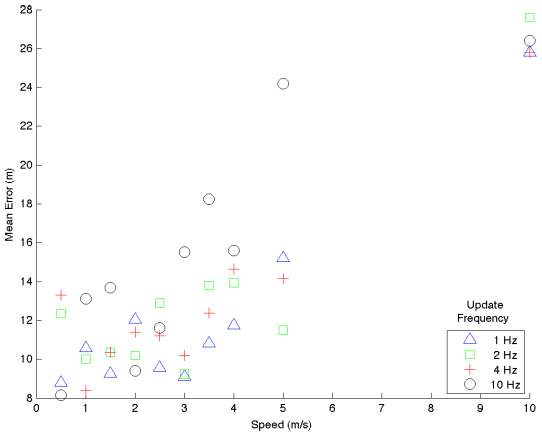
The time taken to map a complete contour does not change significantly when the received counts are made realistic. In many cases, the Poisson-distributed count rates resulted in faster scans because the path is less precise and therefore may be shorter. Complete timing results for both the ideal and stochastic simulations are presented in Table 5.4.

5.4 Source Localization

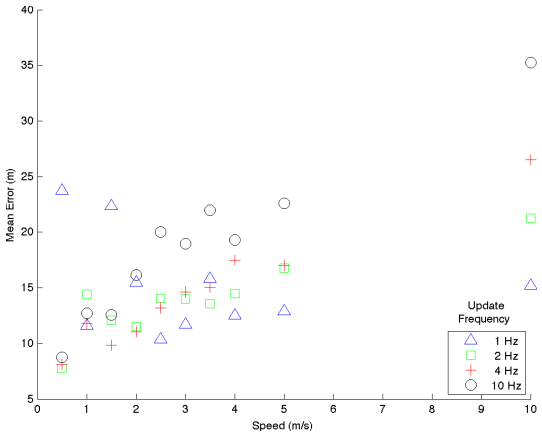
Once a level contour of radiation counts has been mapped, the positions and intensities of the radioactive sources that generated the contour may be estimated. The Hough transform, introduced in Section 3.3, is the central algorithm in this step of the process. The procedure



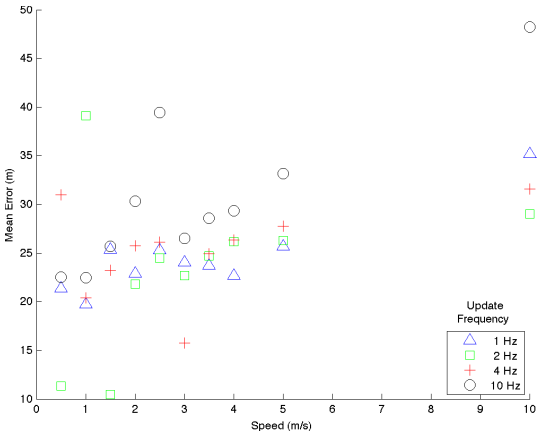
(a) One Weak Source



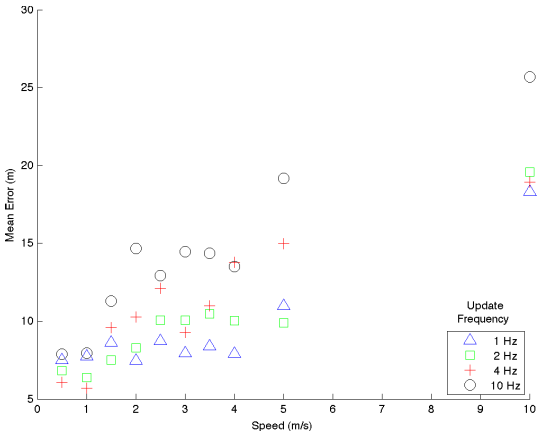
(b) One Strong Source



(c) Two Equal Sources



(d) Two Unequal Sources



(e) Three Unequal Sources

Figure 5.7: Median displacement errors for Poisson contour following of various source configurations

Table 5.4: Mean times required to follow ideal and Poisson contours

Speed (m/s)	Mean Time (s)									
	Weak Source		Strong Source		Three Sources		Equal Sources		Unequal Sources	
	Ideal	Poisson	Ideal	Poisson	Ideal	Poisson	Ideal	Poisson	Ideal	Poisson
0.5	1163	1184	4623	3810	3520	3447	5281	5635	5218	4087
1.0	584	665	2316	2376	1763	1944	2646	3340	2614	3155
1.5	392	438	1547	1603	1178	1331	1767	1961	1744	1779
2.0	295	325	1161	1202	885	991	1327	1504	1309	1638
2.5	236	255	932	971	709	788	1064	1218	1050	2019
3.0	198	216	777	812	592	650	888	1006	876	830
3.5	170	185	667	706	506	553	872	778	753	815
4.0	150	167	585	600	443	484	666	755	660	743
5.0	122	134	470	489	358	392	536	601	531	607
10.0	87	73	235	355	180	194	244	305	235	255

outline is presented as Algorithm 3; the complete code is included in Appendix B.

The contour points are first converted into a binary image to simplify later calculations (line 1). The image size in pixels is exactly the size of the contour in meters. Because the Hough transform can be extremely computationally intensive, the image may be downsampled to halve or quarter the resolution and correspondingly decrease computation time. The Hough transform algorithm requires a search space (a set of $(x, y, radius)$ tuplets to analyze), which provides another opportunity to speed up the algorithm. Because this analysis assumes that the contour mapping has been carried out as in Section 5.3, the image will contain an entire contour. The maximum radius of any circle contained in the image must be no larger than half the size of the smaller dimension of the image, providing an upper bound on the radii to test. The minimum radius is set arbitrarily; a circle with radius less than about one-eighth the smaller dimension of the image represents a source that is almost always too weak to localize accurately. This consideration provides a useful lower bound on

Algorithm 3 Hough Transform-Based Contour Analysis

$cImage \leftarrow$ GETIMAGE($contourPoints$)
$cImage \leftarrow$ DOWNSAMPLE($cImage$)
$maxRadius \leftarrow \frac{\text{MIN}(\text{DIM}(cImage))}{2}$
$minRadius \leftarrow \frac{maxRadius}{8}$
$cand \leftarrow$ HOUGHCIRCLES($cImage, minRadius, maxRadius$)
$cand \leftarrow$ DELETEINVALIDCANDIDATES($cand, cImage$)
$cand \leftarrow$ UPSAMPLE($cand$)
while LENGTH($cand$) > 0 **do**
  $bestCand \leftarrow cand(cand.c == \text{MAX}(cand.c))$
  for all $cand$ s.t. $cand.center$ is within $bestCand.radius$ of $bestCand.center$ **do**
    delete from $cand$
  end for
end while
repeat
  for all $est \in estimates$ **do**
    $est.intensity \leftarrow$ ESTIMATEINTENSITY(est)
    $estimates \leftarrow$ ADJUSTCOUNTS($estimates$)
  end for
  $maxChange \leftarrow$ MAX($estimates.adjustment$)
until $maxChange == 0$

the radii.

Given these constraints, the image is analyzed with the Hough transform, returning a set of candidate estimates consisting of a center (x, y), a radius r, and a confidence c. When the traced contour is not strictly convex (that is, in almost every possible combination of two or more interacting sources), some of the estimated centers will be outside the contour. Because the source locations must obviously be inside the contour, these centers are invalid and need to be eliminated. The procedure DELETEINVALIDCANDIDATES (line 6) takes the contour points in the image as a polygon and uses MATLAB's INPOLYGON function to eliminate all estimates with centers outside the contour. Following this computation, if the original image was downsampled, the candidates are restored to the original coordinate system of the contour points for further processing.

Because the Hough transform returns all possible circles within the constraints given to it, the list of candidates is usually extremely long and contains many circles that only intersect the contour at a few points. The Hough transform algorithm developed for this thesis eliminates many candidates on each internal iteration prior to returning them. This elimination uses an analysis of the algorithm's confidence in each estimate relative to other estimates for that iteration. Regardless of this first thinning effort, the final candidate list still includes many circles that could not be excluded without comparison to the complete list of candidates.

This algorithm assumes that two sources that fall so close as to be within the contour radius of each other are indistinguishable with the given contour. This assumption does not hold true in all cases; there are scenarios in which two sources of similar strength may lie very close

to each other but still be separable, but the vast majority of source combinations validate the assumption. Holding this assumption true suggests the process by which spurious source estimates are eliminated. The most-likely candidate (that with the highest confidence) is assumed to be a correct source position. All other candidates whose centers fall within one radius of the chosen source position are eliminated from consideration. The loop beginning on line 8 performs this elimination until all candidates have been either chosen or eliminated.

Given a list of source estimates, the intensity may be estimated as suggested previously. The equation for received intensity,

$$\lambda_k(x) = \frac{I}{(x_k - x_0)^2 + (y_k - y_0)^2} + \lambda_b,$$

changes form to accommodate the different definition of source intensity in this chapter. Specifically, because source intensity is defined as the received counts at 60 m, the equation becomes

$$I_{received} = I_{source} \frac{60^2}{r_{source}^2},$$

where r_{source} is the distance from the source to the sensor. This equation may be rearranged to solve for I_{source} if r_{source} is redefined to represent the radius associated with a source estimate:

$$\hat{I}_{source} = I_{contour} \frac{r_{source}^2}{60^2}. \tag{5.3}$$

Unfortunately, this calculation is too simple. In the presence of more than once source, not

all of the counts measured to create the contour can be attributed to any one source. Each source impacts the contour at all points, requiring the compensation of each source intensity estimate for each other source intensity. This algorithm, hereafter referred to as iterative intensity correction, is outlined by the loop starting at line 14 in Algorithm 3. For each estimate, the intensity is initially estimated according to Equation 5.3 using the original contour level for $I_{contour}$. Once an intensity estimate is derived for a source, its effect on each other sources is calculated as the theoretical received counts at the contour on the far side of the second source. That is, the adjustment on source A for the contribution of source B is equal to the theoretical received counts from source B at a distance $(dist_{AB} + radius_B)$. The adjustment is always downward because count contributions are always nonnegative. On the next iteration of the loop, the adjusted count level replaces $I_{contour}$ in the estimation of each source intensity. This process is repeated until the adjustment for each source is reduced to zero. At the end of the adjustment loop, the estimates are final.

5.5 Simulation Results

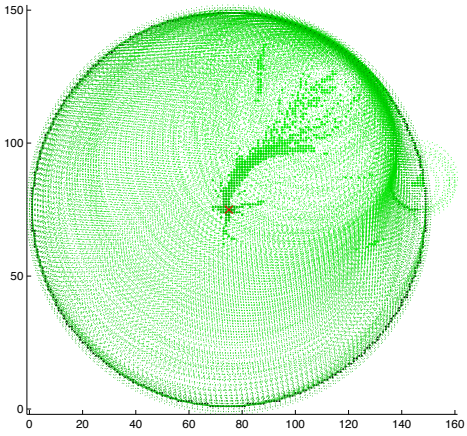
The Hough transform-based source localization algorithm presented in Section 5.4 was applied to both the idealized and stochastic contours generated in Section 5.3. This section presents those results as well as further analysis of the localization algorithm's performance.

5.5.1 Ideal Contour Results

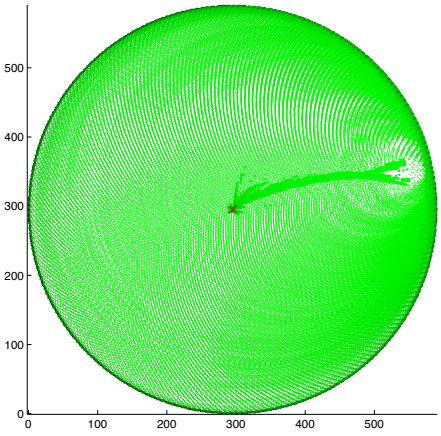
The output of the Hough transform is an extremely long list of candidate source locations that must be culled to the best possible source estimates. The initial results for each source configuration are shown in Figure 5.8. The thick black line is the traced contour from Section 5.3. The green dots are the centers of all of the estimates returned, while the dotted green circles show the radii associated with each estimate. Darker center dots and circles indicate higher confidence associated with an estimate. Note that numerous circles of improper size, indicating sources of improper strength, were returned. The center estimates tend to cluster together near the correct source locations, represented by small red "x" marks.

These plots reveal that even with idealized contours, the signal-to-noise ratio of the returned candidate source estimates is extremely low. Evident in every image is the extreme noise generated by the initial acquisition of the contour. As the helicopter temporarily enters the contour, it flies a curve that the Hough transform correctly interprets as a possible circle. This effect is particularly clear in Figure 5.8(a).

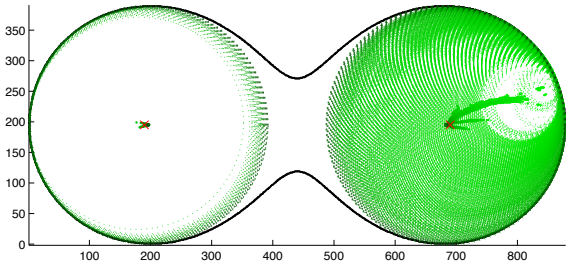
The effect of the choice of minimum radius is very clear in Figure 5.8(b). Because of the initial incursion into the contour (not visible in the figure due to the tight grouping of circles), potential source locations extend from the edge of the contour into the center. However, in Figure 5.8(d), the small radius is necessary to acquire the smaller source. Finally, note that despite the distortion of the center circle in Figure 5.8(e), the transform returns several possibilities near the correct source location.



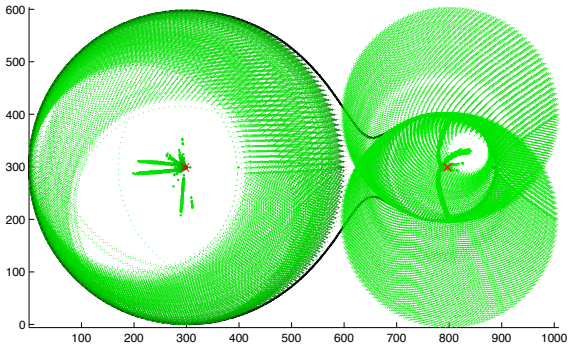
(a) One Weak Source



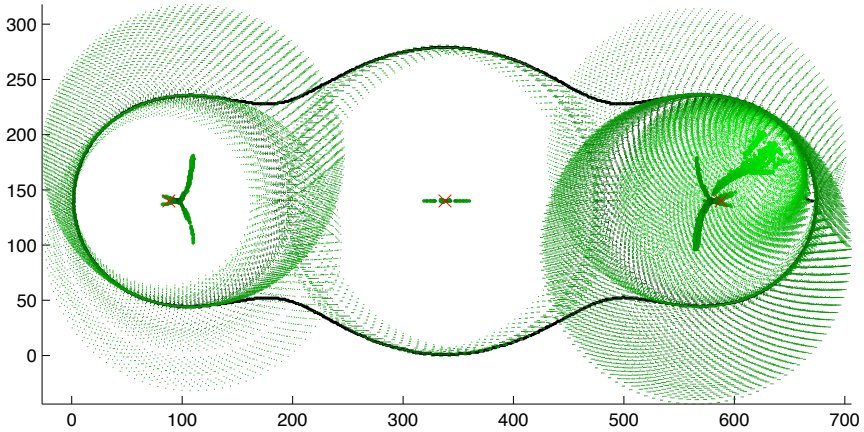
(b) One Strong Source



(c) Two Equal Sources



(d) Two Unequal Sources



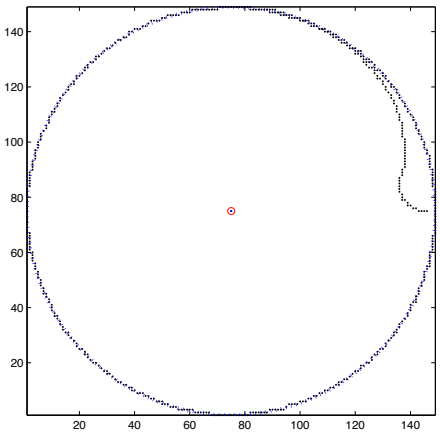
(e) Three Unequal Sources

Figure 5.8: Initial Hough transform results for ideal contours. The black line is the contour; green dots and circles represent source position estimates and associated radii; and red '×' marks are the true target positions.

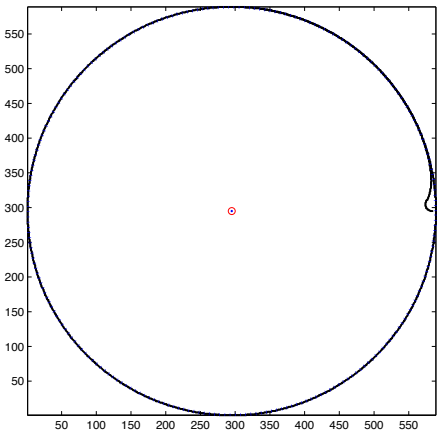
Clearly, nearly all of the returned candidates must be eliminated. The remaining best-choice estimates for each of these source combinations are shown in Figure 5.9. The black line is again the contour; blue dots and circles indicate the best estimated source locations and radii; and the red circles show the true location of the targets.

The source position estimates for the single source configurations are perfect. The contour was tracked closely enough that the estimate could be made exact. When the number of sources increases to two or three, the estimates become less precise because of the distortion of each source's contour by the other sources. However, the errors are very low, especially for the center source in Figure 5.9(e). Errors on the side sources in that plot resemble the errors in Figures 5.9(c) and 5.9(d) in that the true source positions are farther to the outside of the plot than the estimates. It appears that the errors are consistent and related to the strengths of the other source. Specifically, in Figure 5.9(d), the estimate for the large source is nearly correct, but the estimate for the smaller source is significantly worse.

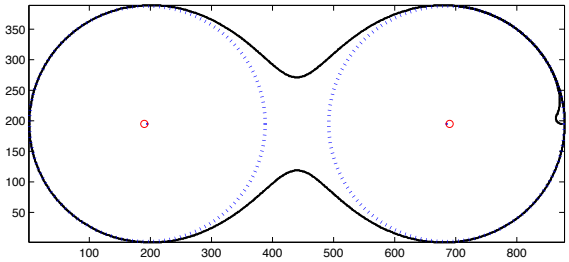
The unequal source plot highlights a shortcoming of this method of localizing sources. Because the contours are post-processed, the helicopter has no opportunity to actively eliminate the spurious result (the large circle on the right of the plot) as it does in the grid-based RBE method. However, the error also emphasizes why the confidence rating returned by the Hough transform is crucial to analyzing the results. (Recall that the confidence is simply the incremented counter value associated with a given source estimate.) The numerical results for this analysis, including the source position and intensity estimates, their errors, and the confidence levels, are shown in Table 5.5, Table 5.6, Table 5.7, Table 5.8, and Table 5.9.



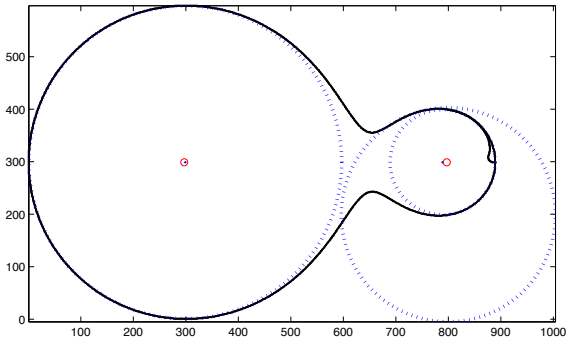
(a) One Weak Source



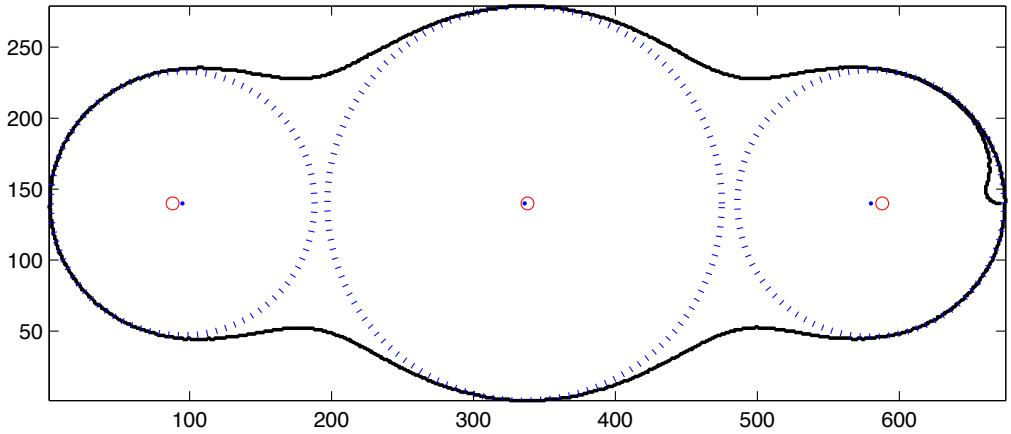
(b) One Strong Source



(c) Two Equal Sources



(d) Two Unequal Sources



(e) Three Unequal Sources

Figure 5.9: Best-choice source estimates for ideal contours. The black line is the contour; blue dots and circles represent source position estimates and associated radii; and red circles are the true target positions.

Table 5.5: Ideal contour analysis results for one small source

DR	Position (m)		Conf.	Raw Intensities			Adj. Intensities		
	Est.	Err.		I	Err.	Rel. Err.	I	Err.	Rel. Err.
1:1	(0,0)	0.00	1.00	5042	42	0.0084	5042	42	0.0084
2:1	(1,1)	1.41	1.00	5042	42	0.0084	5042	42	0.0084
4:1	(1,1)	1.41	1.00	4880	−120	−0.0240	4871	−129	−0.0259
	(73,9)	73.55	0.10	2036	−2964	−0.5929	264	−4736	−0.9473

Table 5.6: Ideal contour analysis results for one large source

DR	Position (m)		Conf.	Raw Intensities			Adj. Intensities		
	Est.	Err.		I	Err.	Rel. Err.	I	Err.	Rel. Err.
1:1	(0,0)	0.00	1.00	50020	20	0.0004	50020	20	0.0004
2:1	(1,1)	1.41	1.00	50020	20	0.0004	50020	20	0.0004
4:1	(1,1)	1.41	1.00	49369	−631	−0.0126	49369	−631	−0.0126

Each table includes information for one source configuration with three downsampling ratios (DR): 1:1, 2:1, and 4:1.

It is clear from the tables that downsampling almost always introduces position and intensity estimate error. It is therefore recommended only when time is extremely critical. The computation time for each source configuration and downsampling ratio is shown in

Table 5.7: Ideal contour analysis results for two equal sources

DR	Position (m)		Conf.	Raw Intensities			Adj. Intensities		
	Est.	Err.		I	Err.	Rel. Err.	I	Err.	Rel. Err.
1:1	(245,0)	5.00	1.00	22694	2694	0.1347	20963	963	0.0481
	(−245,0)	5.00	0.95	22694	2694	0.1347	20872	872	0.0436
2:1	(244,1)	6.08	1.00	22909	2909	0.1454	21179	1179	0.0590
	(−244,1)	6.08	0.93	22909	2909	0.1454	21046	1046	0.0523
4:1	(248,1)	2.24	1.00	22480	2480	0.1240	20755	755	0.0378
	(−244,1)	6.08	0.97	22480	2480	0.1240	20699	699	0.0349

Table 5.8: Ideal contour analysis results for two unequal sources

DR	Position (m)		Conf.	Raw Intensities			Adj. Intensities		
	Est.	Err.		I	Err.	Rel. Err.	I	Err.	Rel. Err.
	(−248, 0)	2.00	1.00	51336	1336	0.0267	50857	857	0.0171
1:1	(242, 0)	8.00	0.28	7556	2556	0.5111	4991	−9	−0.0018
	(253,−100)	100.04	0.10	25120	20120	4.0240	19992	14992	2.9985
	(−247, 1)	3.16	1.00	51336	1336	0.0267	50800	800	0.0160
2:1	(243, 1)	7.07	0.30	7556	2556	0.5111	4887	−113	−0.0226
	(255, −99)	99.13	0.11	25120	20120	4.0240	19981	14981	2.9962
	(−247, 1)	3.16	1.00	50676	676	0.0135	49909	−91	−0.0018
4:1	(245, 1)	5.10	0.39	7556	2556	0.5111	4493	−507	−0.1014
	(253, −99)	99.05	0.18	25120	20120	4.0240	19910	14910	2.9821

Table 5.9: Ideal contour analysis results for three unequal sources

DR	Position (m)		Conf.	Raw Intensities			Adj. Intensities		
	Est.	Err.		I	Err.	Rel. Err.	I	Err.	Rel. Err.
	(242,0)	8.00	1.00	6909	1909	0.3818	6035	1035	0.2071
1:1	(−243,0)	7.00	0.98	6805	1805	0.3610	5920	920	0.1840
	(−2,0)	2.00	0.57	12734	2734	0.2734	10915	915	0.0915
	(242,0)	8.00	1.00	6909	1909	0.3818	5992	992	0.1984
2:1	(−242,0)	8.00	0.98	6909	1909	0.3818	5988	988	0.1976
	(0,0)	0.00	0.59	12889	2889	0.2889	11045	1045	0.1045
	(242,4)	8.94	1.00	7120	2120	0.4240	6139	1139	0.2278
4:1	(−238,4)	12.65	0.95	7120	2120	0.4240	6049	1049	0.2099
	(−6,4)	7.21	0.66	12889	2889	0.2889	11031	1031	0.1031

Figure 5.10 to demonstrate the time advantage of downsampling. Note, however, that the worst-case analysis took less than 2.5 minutes, and the difference for the smallest plot is less than five seconds, reinforcing that only extremely time-critical analyses should downsample the contours.

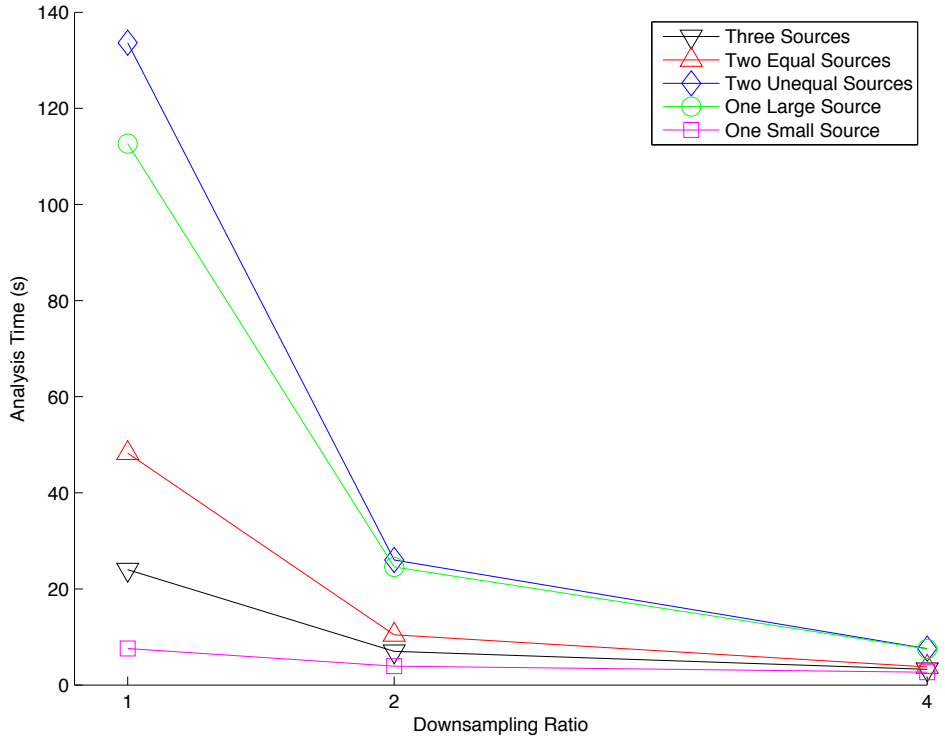


Figure 5.10: Ideal contour analysis times by downsampling ratio; 1 indicates no downsampling

The data also show that the iterative intensity correction routine (Algorithm 3, line 14) radically reduces error in almost all cases. The only case in which error is increased is the localization of a single small source with 4:1 downsampling, where a spurious source is not eliminated and it disrupts the estimate for the correct source. Even in that case, the change was only 9 counts per second in the wrong direction, a negligible value. (Note also in that case that the confidence in the spurious source was extremely low, only one-tenth as likely

to be correct as the correct source. Any operator to whom these results were returned could immediately discard that source as unimportant.)

When the single-source configurations are excluded (because intensity correction has no effect on them), the correction algorithm improves relative error by a mean of 0.1868, 0.0888, and 0.2399 for three sources, two equal sources, and two unequal sources respectively. Overall, the algorithm improves relative error by a mean of 0.1718 across all trials. This improvement represents a significant difference in the precision of the intensity estimate. For example, the smaller of the two unequal sources, which has a true intensity of 5000 counts, was estimated at 7556 counts by the raw algorithm, but was corrected to 4887 counts. The source estimate was initially 51.11% high and was corrected to 2.26% low. These results are summarized in Table 5.10.

Table 5.10: Improvement in mean ideal contour intensity estimates via iterative intensity correction

Source Combination	Raw Rel. Err.	Adj. Rel. Err.	Δ Rel. Err.
Two Equal Sources	0.3563	0.1693	0.1868
Two Unequal Sources	0.1347	0.0459	0.0888
Three Unequal Sources	0.2667	0.0268	0.2399
Overall Means	0.2525	0.0807	0.1718

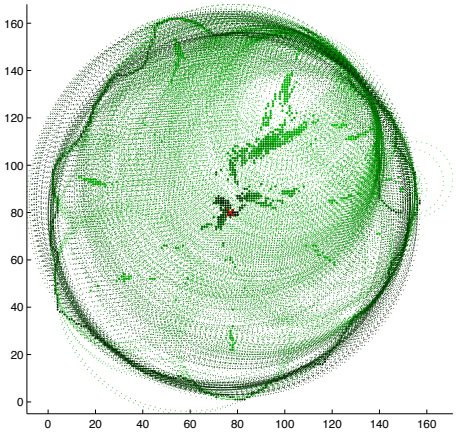
5.5.2 Poisson Contour Results

The algorithm applied to the ideal contour results above was also applied to the simulations in which received radiation count values were chosen from the Poisson distribution. Hereafter, these contours will be referred to as "Poisson contours" for simplicity of nomenclature. The

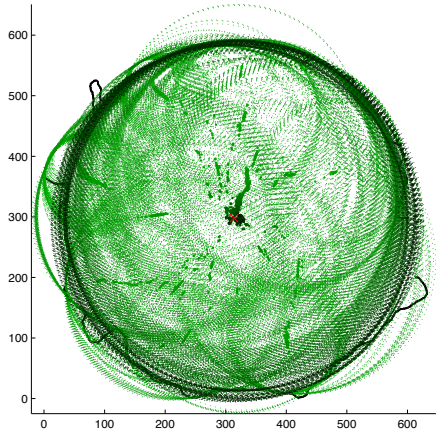
initial results for each source configuration, using the contours shown in Figure 5.6, are shown in Figure 5.11 with the same symbols as Figure 5.8. The green dots are the centers of all the estimates returned, the dotted green circles show the radii associated with each estimate, and darker green symbols indicate higher confidence associated with the estimate. The correct source locations are again represented by red '×' marks. The contours shown were processed with no downsampling.

Following a realistic contour using Poisson-distributed received count rates resulted in extremely variable contours. The path oscillates randomly and only roughly follows the true contour, as shown most clearly in Figure 5.11(c). In that figure, an entire portion of the contour is a perfectly straight line resulting from a violation of the total yaw restriction imposed in Section 5.3 to ensure completion of the contour. Despite the variation, however, after removing the invalid source location estimates (those lying outside the polygon whose vertices are the contour points), the returned circles look very similar to those for the ideal contours. The minimum radius prevented the suggestion of sources inside the small oscillations of the traced contour, and the Hough transform successfully located sufficiently-circular sections of each contour to permit successful analysis. The worst cases are clearly the smaller source in Figure 5.11(d) and the left-hand source in Figure 5.11(e).

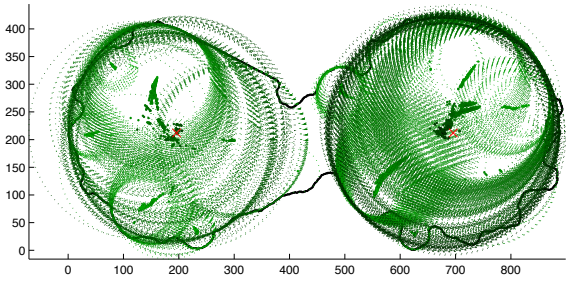
It is important to note at this point that the nature of the Hough transform algorithm prioritizes larger circles. Because the confidence associated with each estimated location is computed by the total number of contour points intersected, a larger circle has a larger maximum possible confidence. This property of the transform is particularly useful when



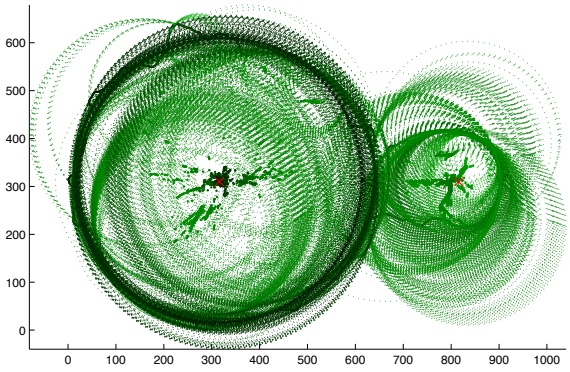
(a) One Weak Source



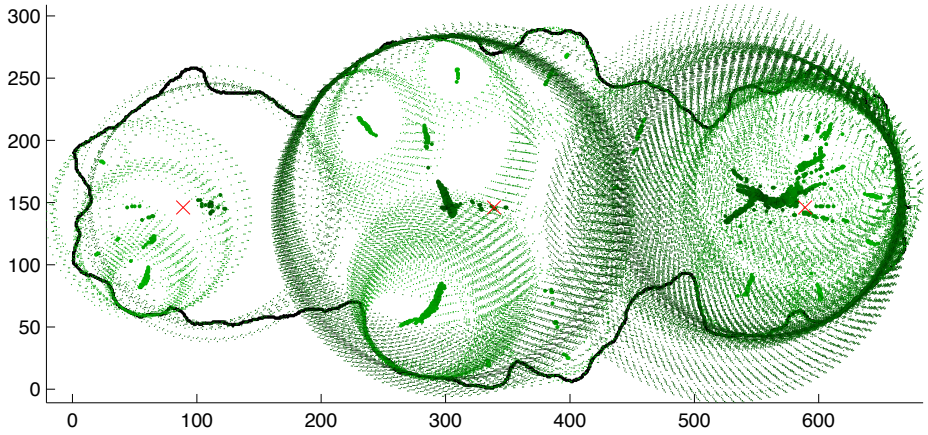
(b) One Strong Source



(c) Two Equal Sources



(d) Two Unequal Sources



(e) Three Unequal Sources

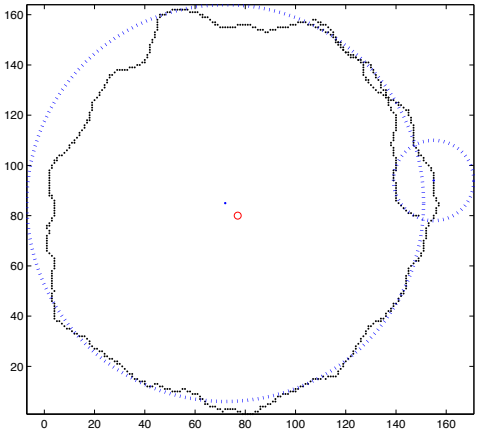
Figure 5.11: Initial Hough transform results for Poisson contours. The black line is the contour; green dots and circles represent source position estimates and associated radii; and red '×' marks are the true target positions.

analyzing such variable contours as these because the smaller circles returned already have decreased confidence, even when the segment of the contour they intersect is far more circular than the larger "correct" circles. This feature is particularly evident in Figure 5.11(e), in which many more estimates with small radii were returned than estimates with the larger radii that correspond to the correct sources. Despite the prevalence of smaller circles, however, the larger circles have much more confidence simply due to their size and capability of intersecting more contour points.

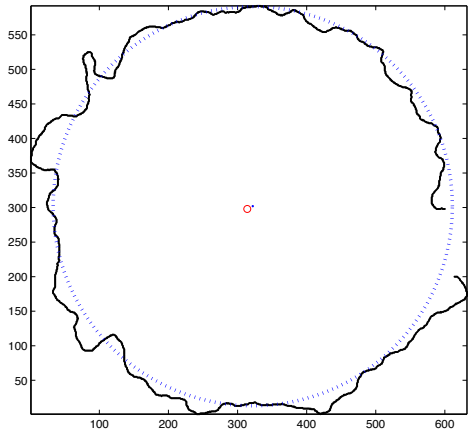
The result of eliminating spurious estimates is not as clean in these simulations as it was when analyzing the ideal contours. As seen in Figure 5.12, the algorithm found many more likely source position candidates than with the ideal contours. However, the confidence associated with the still-incorrect estimates is significantly lower than that of the most-accurate estimates. As earlier, the black line in each plot is the traced contour, the blue points and dashed blue circles are final source estimates and their associated radii, and the small red circles are the true source locations.

Despite the relatively low quality of the traced contours, the algorithms developed earlier in this thesis successfully localized each of the sources in each configuration. In some cases, the position estimate is almost as accurate as for the ideal contours, such as in Figure 5.12(b); in others, the estimate is significantly more erroneous, as for all three sources in Figure 5.12(e).

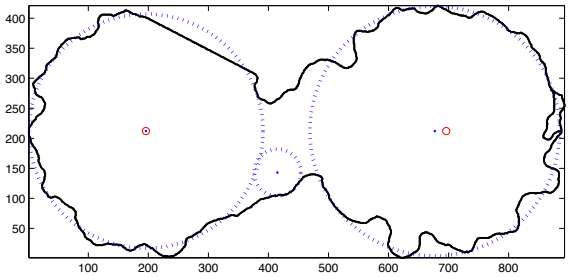
With the single exception of the single strong source, all source configurations resulted in incorrect source position estimates in the final output. In some cases, such as the single weak source, the spurious source is trivially eliminated by noting that most of its circle



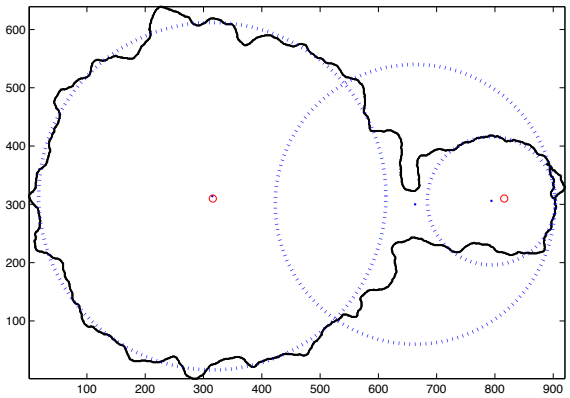
(a) One Weak Source



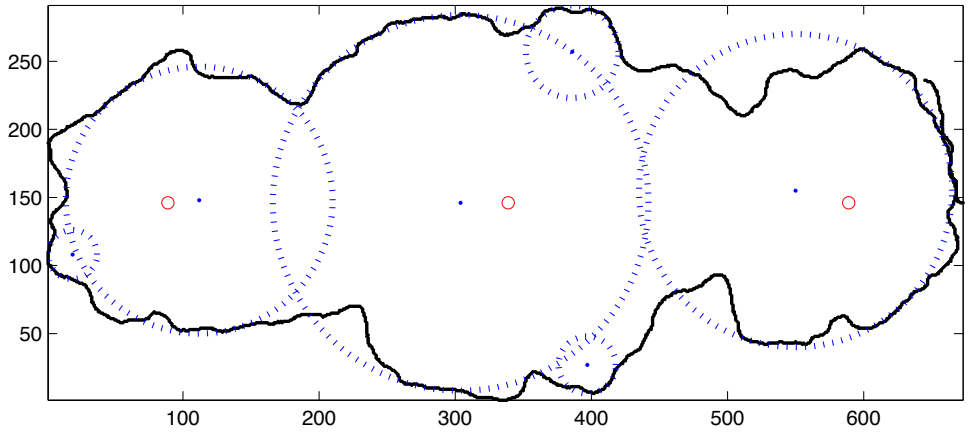
(b) One Strong Source



(c) Two Equal Sources



(d) Two Unequal Sources



(e) Three Unequal Sources

Figure 5.12: Best-choice source estimates for Poisson contours. The black line is the contour; blue dots and circles represent source position estimates and associated radii; and red circles are the true target positions.

Table 5.11: Poisson contour analysis results for one small source

DR	Position (m)		Conf.	Raw Intensities			Adj. Intensities		
	Est.	Err.		I	Err.	Rel. Err.	I	Err.	Rel. Err.
1:1	(−5, 5)	7.07	1.00	5467	467	0.0934	5419	419	0.0838
	(78,14)	79.25	0.25	2142	−2858	−0.5716	594	−4406	−0.8812
2:1	(1, 0)	1.00	1.00	5209	209	0.0418	5194	194	0.0388
	(−25,74)	78.11	0.14	2036	−2964	−0.5929	317	−4683	−0.9367
4:1	(3, 4)	5.00	1.00	5209	209	0.0418	5209	209	0.0418

lies outside the contour. Unfortunately, this observation cannot form the basis of a robust decision algorithm, as nearly half of the circle of the correct source is also outside the contour in that case. The same is true of the incorrect middle source in Figure 5.12(d).

Some invalid results are more difficult to obviously eliminate, however. The central source estimate in Figure 5.12(c) may in fact exist according to the traced contour; visually, it cannot be ruled out. Fortunately, the confidence metric returned by the Hough transform algorithm seems to indicate that circle is invalid: its confidence is only one-third as high as that of the right-hand source and less than half as high as that of the left-hand source. The complete numerical results of the analysis of the Poisson contours are presented in Table 5.11, Table 5.12, Table 5.13, Table 5.14, and Table 5.15. Each table includes information for one source configuration with three downsampling ratios (DR): 1:1, 2:1, and 4:1.

The relationship of downsampling to increased error is no longer as clear with the Poisson contours as it was when analyzing the ideal contours. In fact, in many cases, downsampling served as a trivial smoothing function on the contour, permitting more-accurate estimates during the Hough transform step. The most evident change from the ideal contours is the

Table 5.12: Poisson contour analysis results for one large source

DR	Position (m)		Conf.	Raw Intensities			Adj. Intensities		
	Est.	Err.		I	Err.	Rel. Err.	I	Err.	Rel. Err.
1:1	(8, 4)	8.94	1.00	48401	−1599	−0.0320	48401	−1599	−0.0320
2:1	(6,−6)	8.49	1.00	50676	676	0.0135	50676	676	0.0135
4:1	(10,−2)	10.20	1.00	50676	676	0.0135	50676	676	0.0135

Table 5.13: Poisson contour analysis results for two equal sources

DR	Position (m)		Conf.	Raw Intensities			Adj. Intensities		
	Est.	Err.		I	Err.	Rel. Err.	I	Err.	Rel. Err.
	(231, 0)	19.00	1.00	26036	6036	0.3018	24297	4297	0.2148
1:1	(−250, 0)	0.00	0.84	23125	3125	0.1562	20900	900	0.0450
	(−31,−69)	229.61	0.31	2845	−17155	−0.8578	414	−19586	−0.9793
	(232, 0)	18.00	1.00	26036	6036	0.3018	24238	4238	0.2119
2:1	(−248, 8)	8.25	0.85	23342	3342	0.1671	21095	1095	0.0548
	(−34,−62)	224.72	0.30	3176	−16824	−0.8412	529	−19471	−0.9735
	(234, 4)	16.49	1.00	26036	6036	0.3018	24646	4646	0.2323
4:1	(−266, −8)	17.89	0.75	21636	1636	0.0818	19598	−402	−0.0201
	(−26,−68)	234.09	0.20	2720	−17280	−0.8640	620	−19380	−0.9690

Table 5.14: Poisson contour analysis results for two unequal sources

DR	Position (m)		Conf.	Raw Intensities			Adj. Intensities		
	Est.	Err.		I	Err.	Rel. Err.	I	Err.	Rel. Err.
	(−251, 4)	4.12	1.00	51336	1336	0.0267	48486	−1514	−0.0303
1:1	(228, −4)	22.36	0.88	8722	3722	0.7444	3847	−1153	−0.2305
	(97, −10)	153.33	0.44	34000	29000	5.8000	24040	19040	3.8081
	(−250, 4)	4.00	1.00	51336	1336	0.0267	50587	587	0.0117
2:1	(232, 0)	18.00	0.76	8480	3480	0.6960	6271	1271	0.2541
	(24,−116)	254.03	0.28	2980	−2020	−0.4040	407	−4593	−0.9186
	(−250, 10)	10.00	1.00	52000	2000	0.0400	49948	−52	−0.0010
4:1	(234, 2)	16.12	0.58	8480	3480	0.6960	4327	−673	−0.1345
	(106, 2)	144.01	0.34	31902	26902	5.3804	22621	17621	3.5241

Table 5.15: Poisson contour analysis results for three unequal sources

DR	Position (m)		Conf.	Raw Intensities			Adj. Intensities		
	Est.	Err.		I	Err.	Rel. Err.	I	Err.	Rel. Err.
1:1	(−35, 0)	35.00	1.00	12580	2580	0.2580	10546	546	0.0546
	(211, 9)	40.02	0.95	9347	4347	0.8694	7757	2757	0.5514
	(−227, 2)	23.09	0.80	7336	2336	0.4671	5334	334	0.0667
	(47, 111)	120.54	0.33	2642	−7358	−0.7358	303	−9697	−0.9697
	(−320, −38)	79.65	0.30	2180	−2820	−0.5640	720	−4280	−0.8559
	(58,−119)	132.38	0.28	2245	−7755	−0.7755	225	−9775	−0.9775
2:1	(213, 8)	37.85	1.00	9220	4220	0.8440	7697	2697	0.5393
	(−35, 0)	35.00	0.97	12580	2580	0.2580	10521	521	0.0521
	(−227, 2)	23.09	0.77	7336	2336	0.4671	5320	320	0.0639
	(47, 112)	121.46	0.34	2642	−7358	−0.7358	352	−9648	−0.9648
	(−319, −38)	78.77	0.28	2180	−2820	−0.5640	694	−4306	−0.8613
	(61,−120)	134.61	0.27	2180	−7820	−0.7820	211	−9789	−0.9789
	(−319, 38)	78.77	0.26	2180	−2820	−0.5640	666	−4334	−0.8668
4:1	(217, 10)	34.48	1.00	8969	3969	0.7938	7457	2457	0.4914
	(−35, 2)	35.06	0.80	12276	2276	0.2276	10096	96	0.0096
	(−227, 6)	23.77	0.66	7120	2120	0.4240	5396	396	0.0792
	(105, 18)	106.53	0.48	5920	−4080	−0.4080	2137	−7863	−0.7863
	(49, 110)	120.42	0.34	2720	−7280	−0.7280	518	−9482	−0.9482
	(57,−106)	120.35	0.25	2569	−7431	−0.7431	535	−9465	−0.9465
	(−319, 42)	80.78	0.21	2142	−2858	−0.5716	830	−4170	−0.8340
	(−319, −38)	78.77	0.20	2142	−2858	−0.5716	863	−4137	−0.8274

profusion of spurious estimates when analyzing the three sources. Fully three, four, and five spurious results were returned for 1:1, 2:1, and 4:1 downsampling, respectively, for that configuration. Of greater concern, the confidence values associated with some of the incorrect values is significantly higher than desired. For example, the most-likely incorrect estimate at 4:1 downsampling is more than 100 m away from the closest true source location, but carries a confidence of 48%, while the least-likely correct estimate has a confidence of only 66%.

Another large deviation from the ideal contour analysis results, the position errors for the Poisson contour source estimates are substantially higher, reaching past 40 m for the right-hand source of three when using no downsampling. This extremely high error renders the result nearly unusable for the task of collection; however, the field is a kilometer long and the contour itself is almost 700 m long. This algorithm has reduced the search area from $1\,000\,000\,\mathrm{m}^2$ ($193\,910\,\mathrm{m}^2$ of contour) to just over $5000\,\mathrm{m}^2$, a reduction of 99.5% (97.4% from the contour). The result is even more useful when determining safety perimeters: slightly larger inaccuracies in source positions do not affect perimeter assignment nearly as much as search efforts because the safety zone will include a buffer that will easily correct for this worst-case inaccuracy of less than 50 m.

The estimate position errors for the ideal and Poisson contours are compared visually in Figure 5.13 and numerically in Table 5.16. The mean position estimate error for Poisson contour analysis is nearly four times that for the ideal contour analysis, although the standard deviation also nearly quadrupled. The variation in contour quality resulted in both one perfect estimate (the left-hand source of the two equal sources) and the 40-meter error

already discussed.

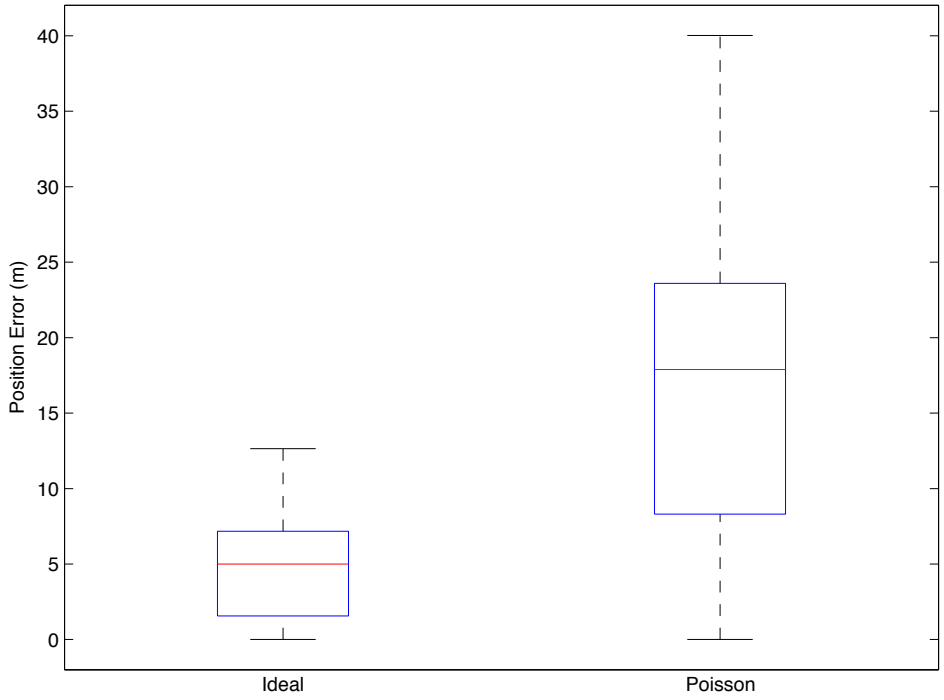


Figure 5.13: Visual comparison of source position estimate errors between ideal and Poisson contours

As the source position estimate errors increased, so did the intensity estimate errors. The iterative intensity correction again successfully decreased the errors. The mean relative error for the three-source combination was 51% after the initial analysis, but was reduced to 21% through the compensation technique presented in Section 5.4. The complete intensity estimate errors and error reductions are presented in Table 5.17 alongside the ideal contour results for comparison.

The significantly less-smooth Poisson contour results in a very large number of candidate source locations. However, the efficiency of this implementation of the Hough transform processes the stochastically-derived contours nearly as quickly as the smooth contours of the

Table 5.16: Numerical comparison of Euclidean source position estimate errors between ideal and Poisson contours

	Ideal	Poisson
n	27	27
Mean	4.68	17.86
Std. Dev.	3.30	12.11
Min	0.00	0.00
Median	5.00	17.89
Max	12.65	40.02

Table 5.17: Improvement in mean source intensity estimates via iterative intensity correction

Source Combination	Raw Rel. Err.		Adj. Rel. Err.		Δ Rel. Err.	
	Ideal	Poisson	Ideal	Poisson	Ideal	Poisson
Two Equal Sources	0.3563	0.5121	0.1693	0.2120	0.1868	0.3001
Two Unequal Sources	0.1347	0.2184	0.0459	0.1298	0.0888	0.0886
Three Unequal Sources	0.2667	0.3716	0.0268	0.1104	0.2399	0.2613
Overall Means	0.2525	0.3674	0.0807	0.1507	0.1718	0.2167

previous section. Figure 5.14 shows the required computation time for the various source configurations at different downsampling ratios.

The results discussed in this section clearly demonstrate that the techniques presented in this chapter rapidly, effectively, and accurately localize one, two, or three sources of unknown intensity without prior knowledge of the number of sources or their strengths. The mean position error for tracing contours using realistic Poisson distribution-based simulation parameters was 17.86 m. Furthermore, the algorithms presented here estimate the source intensity with reasonable accuracy. The mean relative intensity error in the analysis of the Poisson distribution-based simulation was 15.07%, as improved from a raw estimated intensity relative error of 36.74%. These algorithms give accurate, useful information regarding

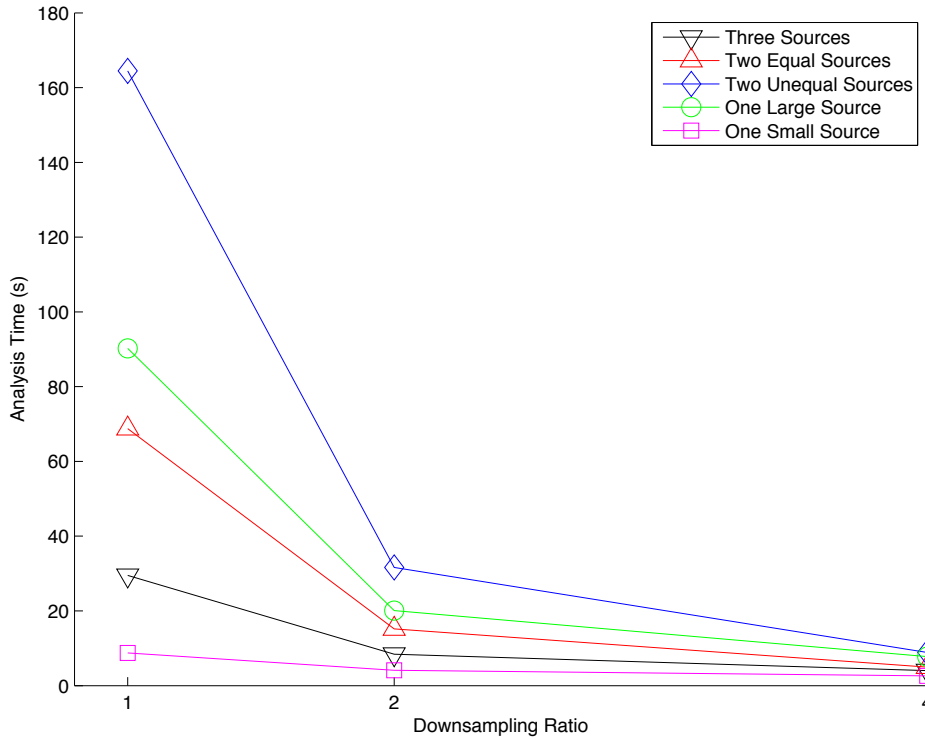


Figure 5.14: Ideal contour analysis times by downsampling ratio; 1 indicates no downsampling

the number of radioactive sources present in an area, their positions, and their intensities, without the unacceptably high computation requirements or the "curse of dimensionality" suffered by both the particle filter of Brewer and the grid-based RBE of Chapter 4.

Chapter 6

Efficient Ingress Paths

The contour-following algorithms presented in Section 5.3 can require extensive time to complete. Especially when the source has a high intensity, completing a single contour may take more than thirty minutes. Because the Yamaha RMAX used by the Unmanned Systems Laboratory has an endurance of only one hour, minimizing the time between takeoff and beginning the contour (ingress) and between finishing the contour and returning to the launch site (egress) is essential. Section 5.2 introduced a contour search method to minimize the time required to find a contour once the search area had been reached. This chapter presents a simple method to optimize fuel usage during the ingress and egress stages, completing the development of the mission considered in this thesis. The work comprises a calculation of drag on the Yamaha RMAX helicopter (Section 6.1) and a nonlinear optimization of energy usage over a given terrain (Section 6.2). The MATLAB code that implements the calculations and optimization problem in this chapter is presented in Appendix C.

6.1 RMAX Power Requirements

Analytical computation of the aerodynamic drag on any non-trivial body is extremely complex. Except when a sufficiently large wind tunnel is available to provide exact measurements, drag models employ calculations based on the component shapes of the body. The approximate shape of each component is compared to tables of shapes with known drag coefficients, and a model of the entire body is built up sequentially. This section presents the approximation of the aerodynamic drag on the USL's Yamaha RMAX helicopter.

In the configuration analyzed, the helicopter has an instrumentation pod carrying a radiation detector provided by Sandia National Laboratories (SNL) and a stereo boom used for topography mapping. The equations and approximations in this section are found in Prouty [33] (especially pp. 288,298-308) and Munson, Young, and Okiishi [30] (pp. 526,537). Where easy to express, the sources of constants and calculations are given. A complete, rigorous treatment of these calculations is far beyond the scope of this thesis, so only the calculations and basic information are presented here. Prouty discusses these calculations in detail.

6.1.1 Measurements and Constants

The calculation of drag begins with measurements of the body to be analyzed. These measurements are presented in Table 6.1. Note that, because imperial measurements are the most common measurements in the literature, the calculations in this chapter are carried out in imperial units. Some measurements were taken in metric units for ease of manipulation; all

were converted before applying the following equations.

Dimensional measurements of the RMAX were taken with measuring tape. Areal measurements were taken by photographing each side of the component and applying manual image analysis. Precise measurements of machined parts in each picture were taken with digital calipers to give the horizontal and vertical dimension of each pixel. Areas were then calculated by totaling pixel coverage of a component and multiplying by the area of a single pixel. These calculations were possible only because of the careful orthogonal positioning of the helicopter in each image.

The takeoff weight of the RMAX was calculated by adding the stock weight reported by Yamaha to the measured weight of each added component. The angular velocity of the main rotor was reported by Yamaha.

Drag calculations depend heavily on the density of the fluid medium. In this case, the fluid is air at $T = 70^\circ\,\mathrm{F}$ ($21.11^\circ\,\mathrm{C}$) and $p = 1\,\mathrm{atm}$ ($14.504\,\mathrm{psi}$). The density of the air is

$$\rho = 144\frac{p}{R_{specific}T} = 1.2233\frac{lb_m}{ft^3},$$

[2]The rotor mask area is the area of the top of the helicopter swept out by the arc of the blades.
[3]Prouty, p. 23
[4]The drag coefficient is based on the fineness ratio, $\frac{l_{rmax}}{\max(w_{rmax},h_{rmax})}$.
[5]Prouty, p. 298, 0% RPM
[6]Prouty, p. 299, 100% RPM and 0 shaft angle
[7]Prouty, p. 301
[8]Prouty, p. 303
[9]Prouty, p. 302
[10]Munson, p. 526
[11]Munson, p. 537

Table 6.1: Yamaha RMAX component measurements

Component	Symbol	Value
RMAX Length	l_{rmax}	$9.3373\,ft$
RMAX Width	w_{rmax}	$1.7060\,ft$
RMAX Height	h_{rmax}	$2.3294\,ft$
RMAX Weight	m_{rmax}	$127.868\,lbs$
Blade Radius	r_{blade}	$5.1099\,ft$
Blade Area	A_{blade}	$2.3190\,ft^2$
Blade Number	n_{blade}	2
Rotor Mask Area[2]	A_{rm}	$6.8597\,ft^2$
Rotor Angular Velocity	ω_r	$89.0118\,rad/s$ $(850\,rpm)$
Fuselage Frontal Area	$A_{fus,x}$	$0.1900\,m^2$
Main Rotor Hub Frontal Area	$A_{mh,x}$	$0.3298\,ft^2$
Main Rotor Shaft Frontal Area	$A_{ms,x}$	$0.0473\,ft^2$
Main Rotor Shaft Diameter	d_{ms}	$0.1183\,ft$
Ratio of Hub Gap to Pylon Width	r_{hp}	0.8194
Tail Rotor Assembly Frontal Area	$A_{tail,x}$	$0.0835\,ft^2$
Landing Gear Crossbar Frontal Area	$A_{lg_{cross},x}$	$0.1840\,ft^2$
Landing Gear Skids Frontal Area	$A_{lg_{skids},x}$	$0.0292\,ft^2$
Landing Gear Arms Frontal Area	$A_{lg_{arms},x}$	$0.0632\,ft^2$
Stereo Boom Frontal Area	$A_{sb,x}$	$1.3238\,ft^2$
Stereo Boom Diameter	d_{sb}	$0.2583\,ft$
Radiation Pod Frontal Area	$A_{rad,x}$	$1.2604\,ft^2$

Table 6.2: Yamaha RMAX component constants

Constant	Symbol	Value
Blade Profile Drag Coefficient[3]	C_{Dblade}	0.02
Fuselage Drag Coefficient[4]	C_{Dfus}	0.08
Main Rotor Hub Drag Coefficient[4]	C_{Dmh}	0.15
Main Rotor Hub Drag Ratio[6]	r_{Dmh}	1.0526
Hub Pylon Interference Drag Factor[7]	k_{Dhp}	0
Tail Rotor Hub Drag Coefficient[5]	C_{Dtail}	0.04
Tail Rotor Hub Drag Ratio[6]	r_{Dtail}	1.0526
Landing Gear Drag Coefficient[8]	C_{Dlg}	0.4
Rotor-Fuselage Interference Drag[9]	ΔC_D	0.018
Stereo Boom Drag Coefficient[10]	C_{Dsb}	1.2
Radiation Pod Drag Coefficient[11]	C_{Drad}	0.80
Pound-Force Conversion	g_c	$32.174049\,\frac{lb_m ft}{lb_f s^2}$

where $R_{specific} = 53.3533\frac{ft-lb_f}{lb_m R}$ (the R in the denominator here indicates temperature in Rankine).

6.1.2 Power Calculations

The total power required by the engine is the sum of four quantities: 1) induced power, $P_{induced}$, the power required to create the rotor-induced velocity V_i; 2) profile power, $P_{profile}$, the power necessary to overcome the profile drag of the rotor blades; 3) parasite power, P_{para}, the power required to overcome the drag of the fuselage (and other components) in forward flight; and 4) climb power, P_{climb}, the rate of change of gravitational potential energy due to climbing (or descending). Each of these calculations will be considered in turn.

Induced Power

The induced power is the power required to generate the rotor-induced velocity. Rotor-induced velocity is the velocity change from the air above the main rotor to the air under the main rotor. The induced power, therefore, is the power added to the air in this manner. The necessary thrust for lifting the helicopter is based on the area swept out by the blades and the mass of the helicopter. The induced power may then be calculated using the forward speed v_x of the helicopter:

$$P_{ind} = \frac{T^2}{2\frac{\rho}{g_c}\pi r_{blade}^2 v_x}, \text{ where}$$
$$T = \frac{1 + 0.3A_{rm}}{\pi r_{blade}^2} m_{rmax}.$$

Profile Power

The profile power is used to overcome the drag on the profile of the rotor blades as they turn. The rotor advance ratio is the relationship of the forward speed of the helicopter to the tip speed of the blades:

$$\mu = \frac{v_x}{\omega_r r_{blade}}.$$

The profile power is then

$$P_{profile} = \frac{\rho}{g_c} A_{blade} n_{blades} \left(\omega_r r_{blade}\right)^3 \frac{C_{Dblade}}{8} \left(1 + 3\mu^2\right).$$

Parasite Power

Parasite power overcomes the parasitic drag of the helicopter in forward flight. The simplest method of calculating the power required begins by calculating the equivalent flat plate area of the entire helicopter. By breaking down the body into simple components, an equivalent flat plate area for each component may be calculated from known basic shapes; the equivalent flat plate areas are additive to give a value for the whole vehicle. All the equivalent flat plate calculations are presented here.

$$fp_{fus} = A_{fus,x} C_{D\,fus} \qquad {R_{sb}}^{12} = 6400 v_x d_{sb}$$

$$fp_{rad} = A_{rad,x} C_{D\,rad} \qquad fp_{sb} = A_{sb,x} C_{D\,sb}$$

$$C_{D_{mh}}^{*} = C_{D_{mh}} r_{D_{mh}} \qquad C_{D_{tail}}^{*} = C_{D_{tail}} r_{D_{tail}}$$

$$fp_{mh} = C_{D_{mh}}^{*} A_{mh,x} \qquad fp_{tail} = C_{D_{tail}}^{*} A_{tail,x}$$

$${R_{ms}}^{12} = 6400 v_x d_{ms} \qquad A_{lg} = A_{lg_{cross},x} + A_{lg_{skids},x} + A_{lg_{arms},x}$$

$$fp_{ms} = C_{D_{ms}} A_{ms,x} \qquad fp_{lg} = C_{D\,lg} A_{lg}$$

$$fp_{rotor} = fp_{mh} + fp_{ms} \qquad fp_{if} = \Delta C_D A_{fus,x}$$

$$fp_{rotor}^{*} = (1 + k_{Dhp}) fp_{rotor}$$

[12]The Reynolds number affects the drag coefficient, thus its calculation here.

The parasite power is a function of the total equivalent flat plate area:

$$fp_{total} = fp_{fus} + fp_{mh} + fp_{ms} + fp^*_{rotor} + fp_{tail} + fp_{lg} + fp_{if} + fp_{sb} + fp_{rad}$$

$$P_{para} = 0.5\frac{\rho}{g_c}v_x^3 fp_{total}.$$

Climb Power

The climb power is the simplest power, both conceptually and mathematically. It is exactly equal to the rate of change in gravitational potential energy as the helicopter moves vertically with velocity v_z:

$$P_{climb} = m_{rmax}v_z. \tag{6.1}$$

This equation appears to suggest $P = mv$ instead of the expected $P = Fv$; however, the following equation provides units to explicate the confusion, which involves the pound-force–pound-mass conversion constant g_c:

$$(P_{climb})_{lb_f} = (m_{rmax})_{lb_m} \frac{(g)_{\frac{ft}{s^2}}}{(g_c)_{\frac{lb_m s}{lb_f s^2}}} (v_z)_{\frac{ft}{s}}.$$

Because the values of g and g_c are approximately equal, this equation simplifies to (6.1).

Total Power

Finally, the total power required for a given forward and vertical velocity is the sum of the component power requirements:

$$P_{total} = P_{induced} + P_{profile} + P_{para} + P_{climb}.$$

6.1.3 Power Curves

Given the equations for total power expended in the previous section, two points of efficiency may be calculated. The first, the optimal forward flight speed for climbing, is the forward speed at which the required power is lowest. At this speed, the maximum power is available for climbing as opposed to forward flight or hovering. Because flight at this speed minimizes power, it is also this speed that gives maximum chronological endurance; that is, the helicopter can fly longer at this speed than at any other. It may not fly the maximum distance, however. The second efficiency point is the optimal forward flight speed for distance endurance. This is the speed at which the energy expended per meter of forward flight (calculated by dividing power by forward speed) is lowest. The relationship of power to forward and vertical speeds is shown in Figure 6.1. On this plot, the minimum point at 14.76 ft/s (4.50 m/s) is the first point of efficiency discussed above.

Note that the plot does not extend to 0 m/s forward speed, or hover. Hovering is a very power-intensive state for a helicopter, and the above calculations do not apply. The basic

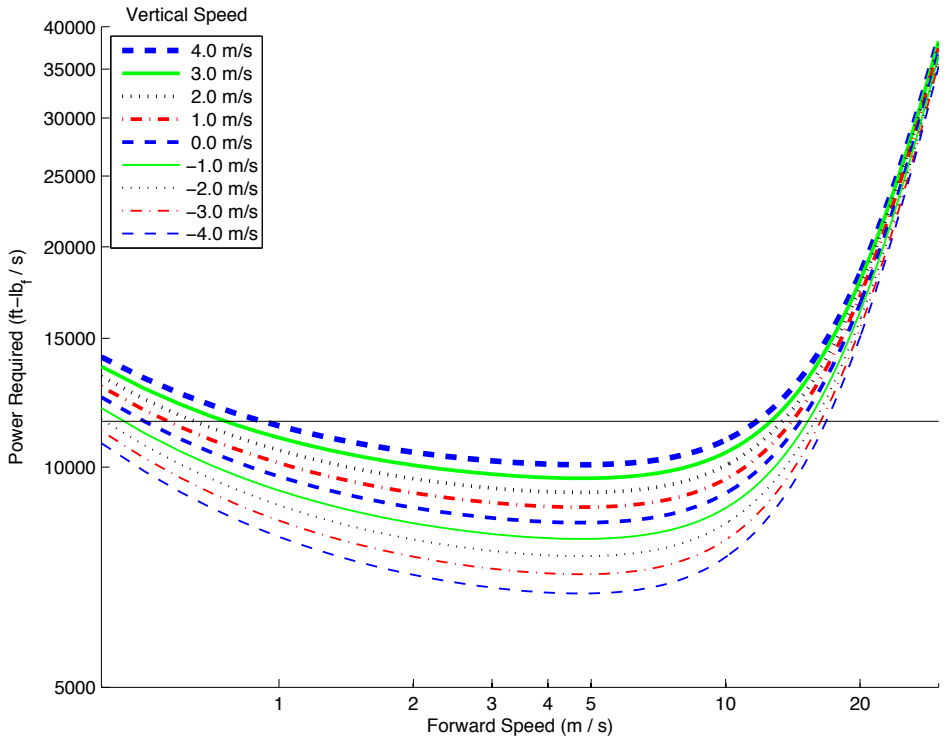


Figure 6.1: RMAX power requirements by forward and vertical speeds

reason for this phenomenon is that, in forward flight, the disc swept out by the main rotor blades provides lift in a manner similar to fixed-wing aircraft; a complete discussion is found in Prouty. This effect can be seen in Figure 6.1 as the concave shape of the curve: as the helicopter moves forward more quickly, the lift provided by the false wing increases and allows the engine to use less power for lift to attain the same forward speed. As the speed increases, however, the power requirements of moving forward rapidly begin to supersede the benefit of the wing effect.

The contribution of each type of power (at $v_z = 0\,\mathrm{m/s}$) is shown in Figure 6.2. The plot makes clear that the parasitic power, which is directly proportional to v_x^3, is negligible at low speeds but grows exponentially and dominates the sum at higher speeds. Conversely,

the induced power, associated with accelerating air as it passes through the rotor blades, decreases as the false wing effect provides lift and the rotor does not need to move as much air as quickly for the same effect. The climb power for a 4.0 m/s climb is shown for comparison. In level flight, the climb power is obviously zero and does not appear on a logarithmic scale; similarly, descent power is negative and exactly equal in magnitude to the corresponding positive value.

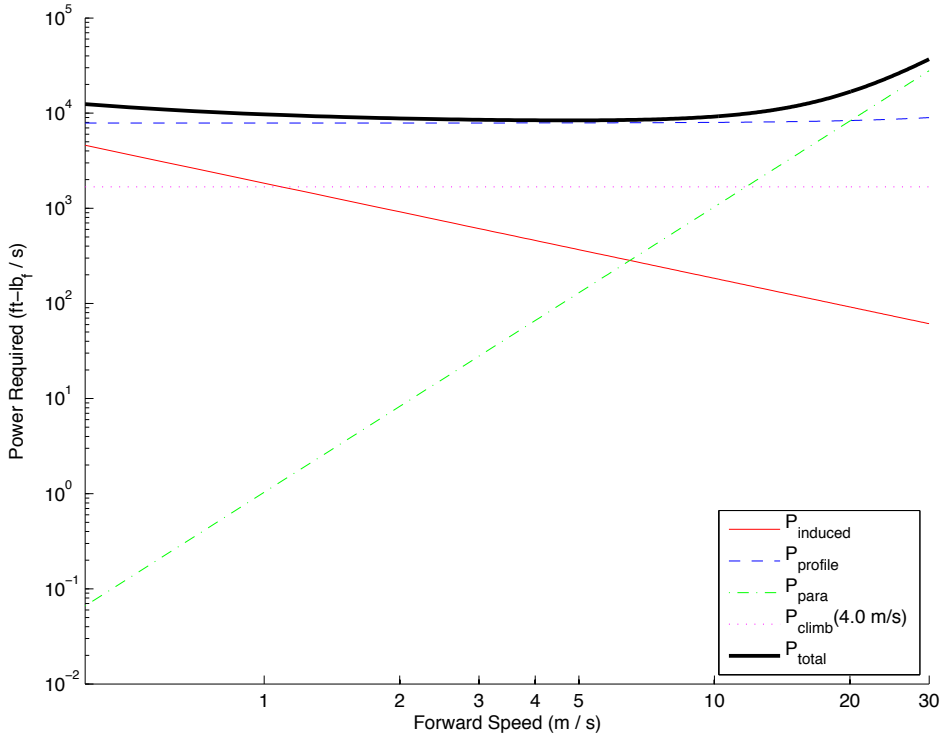


Figure 6.2: RMAX power requirement contributions in level flight; climb power for a 4.0 m/s climb is also shown for comparison.

The distance endurance metric, energy per meter, is shown for the RMAX in Figure 6.3. Note that as forward speed continues to increase past 30 m/s, the increase in energy expended per meter becomes nearly linear with forward speed. The optimal forward speed for distance efficiency is the minimum value on this graph, 67.26 ft/s (20.50 m/s). The RMAX cannot

travel this fast; its maximum air speed is 20 m/s. In practice, the USL limits its air speed to 10 m/s when instruments are mounted for stability purposes. Therefore, the maximum distance can be covered when the helicopter is moving as fast as it is capable.

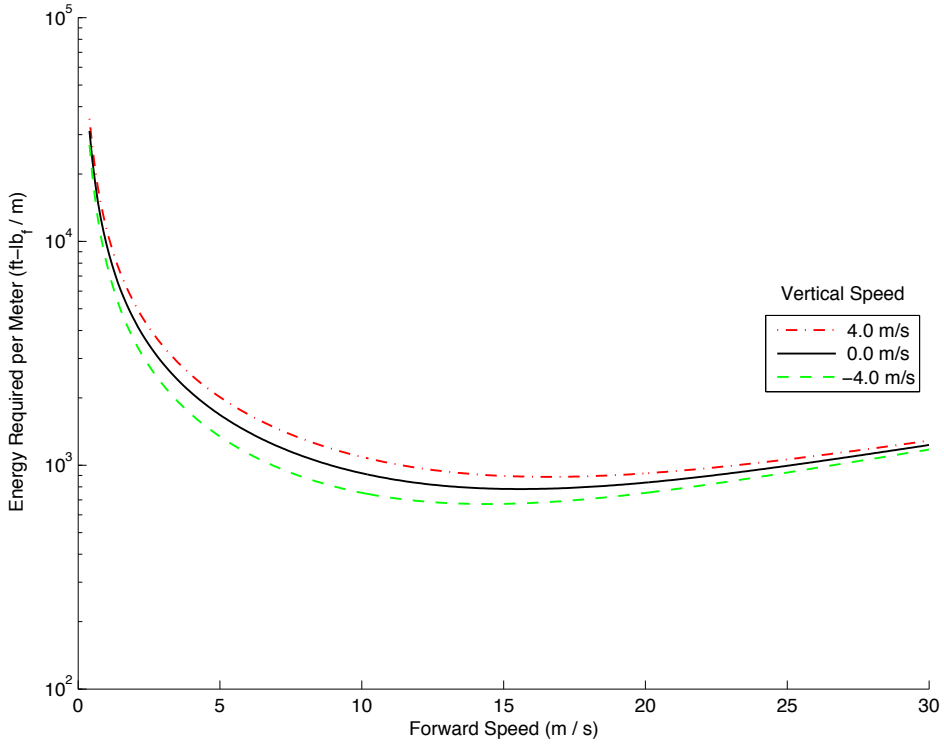


Figure 6.3: RMAX energy efficiency by foward and vertical speeds

6.2 Optimizing Fuel Use

The power calculations in Section 6.1 permit the construction of a nonlinear function that takes as inputs forward and vertical velocities and provides as output required power. When a known distance must be traveled, the required time is trivially computed by $t = \frac{distance}{velocity}$ and the total energy required is $E = Pt$. This result may be used as the objective function

in a nonlinear optimization problem to determine optimal ingress and egress paths for an arbitrary terrain profile.

6.2.1 Problem Structure and Constraints

Vehicle path planning is a diverse and well-studied field in robotics, and especially in unmanned vehicles. The implementation of a robust, general path-planning algorithm is beyond the scope of this thesis. Implementing some simplifying assumptions reduces the problem to a tractable optimization problem to demonstrate the utility of the power calculations in Section 6.1.

Assuming that the helicopter flies in a straight line from the origin to the destination with respect to the x-y plane reduces the problem to two physical dimensions, x and z. The general shape of the path is constrained to two endpoints and two midpoints. The flight then has four stages: the helicopter *a*) begins at the origin; *b*) flies at some positive forward speed and nonnegative vertical velocity to the second point; *c*) flies at the most-energy-efficient forward speed and zero vertical velocity to the third point; and *d*) descends at some positive forward speed and nonpositive vertical velocity to the destination point. These stages suggest restrictions on the choice of the two interior points. The elevation of the first point must be at least the elevation of the initial point, the second and third points (the break points) have the same elevation, and the third point has an elevation at least as high as that of the final point. Further, the second break point's x-position x_r must be at least

as great as the first break point's x-position x_l.

This flight pattern is an appropriate simplification because it approximates necessary stages of any ingress path. Because the problem has only two dimensions, the helicopter must fly over the tallest terrain feature (possibly a mountain or a building). Therefore, it must at some point climb to the necessary altitude, fly over that feature, and descend again to the end point. The proposed flight pattern discretizes those steps and limits the entire flight to the four characteristic points of that pattern. Because the initial and final points are defined by the environment, the optimization algorithm must choose only the x-position of the break points and the velocities of the first and third legs of the flight.

A final consideration for this problem is that the helicopter should not crash into the terrain. In fact, the helicopter should maintain some clearance above the ground to avoid such obstacles as people or cars that a topographical scan could not have mapped. For the simulations presented in this section, the clearance is defined at 10 m. The requirement not to crash into terrain sets minimum slopes for both the climb and descent phases. Because the power calculations do not apply very near the hover condition, a minimum forward speed $v_{xmin} = 0.4\,\mathrm{m/s}$ was imposed, leading to maximum slopes for climb and descent as well. (Recall the helicopter's forward speed is also limited to $v_{xmax} = 10\,\mathrm{m/s}$.) Because the initial and final points are fixed, these extreme slopes define a range of possible x-positions for the

two break points:

$$x_{lmin} \leq x_l \leq x_{lmax}$$

$$x_{rmin} \leq x_r \leq x_{rmax}.$$

6.2.2 Nonlinear Optimization Problem

The optimization problem may now be defined in terms of the constraints developed above. The nonlinear objective function GETTOTALENERGY takes as inputs the (x, z) positions of all four characteristic points as well as the forward velocities of the climb and descent legs and returns the total energy required to traverse the path. The vertical velocities v_{zl} and v_{zr} are defined by the point positions and forward velocities: the vertical displacement must take the same amount of time as the horizontal displacement, and the vertical velocity must not exceed the helicopter's capability. This constraint is nonlinear, necessitating the use of the MATLAB function FMINCON. The optimization problem is parameterized as follows:

$$x_1 \equiv x_l \qquad x_3 \equiv x_r$$

$$x_2 \equiv v_{xl} \qquad x_4 \equiv v_{xr}.$$

The initial and final point coordinates are (x_i, z_i) and (x_f, z_f), respectively. The problem constraints may be stated as follows:

$$\begin{aligned}
\text{minimize} \quad & z = \textsc{getTotalEnergy}(\text{initialPoint}, \text{firstBreak}, \text{lastBreak}, \text{finalPoint}, x) \\
\text{subject to} \quad & x_{lmin} \leq x_1 \leq x_{lmax} \\
& x_{rmin} \leq x_3 \leq x_{rmax} \\
& v_{xmin} \leq x_2 \leq v_{xmax} \\
& v_{xmin} \leq x_4 \leq v_{xmax} \\
& x_1 \leq x_3 \\
& t_l = \frac{x_1 - x_i}{x_2} \\
& v_{zl} = \frac{z_l - z_i}{t_l} \\
& v_{zl} \leq 4\,\text{m/s} \\
& t_r = \frac{x_3 - x_i}{x_4} \\
& v_{zr} = \frac{z_f - z_r}{t_r} \\
& v_{zr} \geq -4\,\text{m/s}.
\end{aligned}$$

To demonstrate the behavior of this algorithm, a set of randomized terrains were created by drawing from a uniform distribution for each potential x-position between the launch area and the search area. The terrain was smoothed using linear interpolation to provide interesting topographical contours against which to test the optimization algorithm. Selected results from these simulations are shown in Figure 6.4. The solid thin black line is the true ground; the dashed black line shows the 10 m ground clearance requirement. The helicopter starts at this height above the terrain, as well. The dashed red line shows the prescribed altitude of the break points, while the red and blue zones denote the possible climb and descent paths as constrained by the terrain. The thick black line with circles at its vertices traces the energy-optimal path from the initial point on the left to the final point on the right. The mean time required to calculate the optimal path over 500 m of terrain was less than 0.0953 s, so computation time is not a factor with this method. These figures show several diverse possibilities for this algorithm. Figure 6.4(a) is a very simple contour that requires an ascent on one side and descent on the other. Note that, as is almost universal for this algorithm, the paths of highest forward speed and lowest vertical velocity are the chosen paths. This result is a consequence of the energy-optimal forward speed being outside the helicopter's limits.

Despite this generalization, the path in Figure 6.4(b) shows that the helicopter could choose any point to begin its descent, but waits until it is halfway across the map to do so. The descent phase is the least energy-intensive phase of the path, and the faster the descent, the greater the energy savings. Descending very quickly, however, would necessitate a much

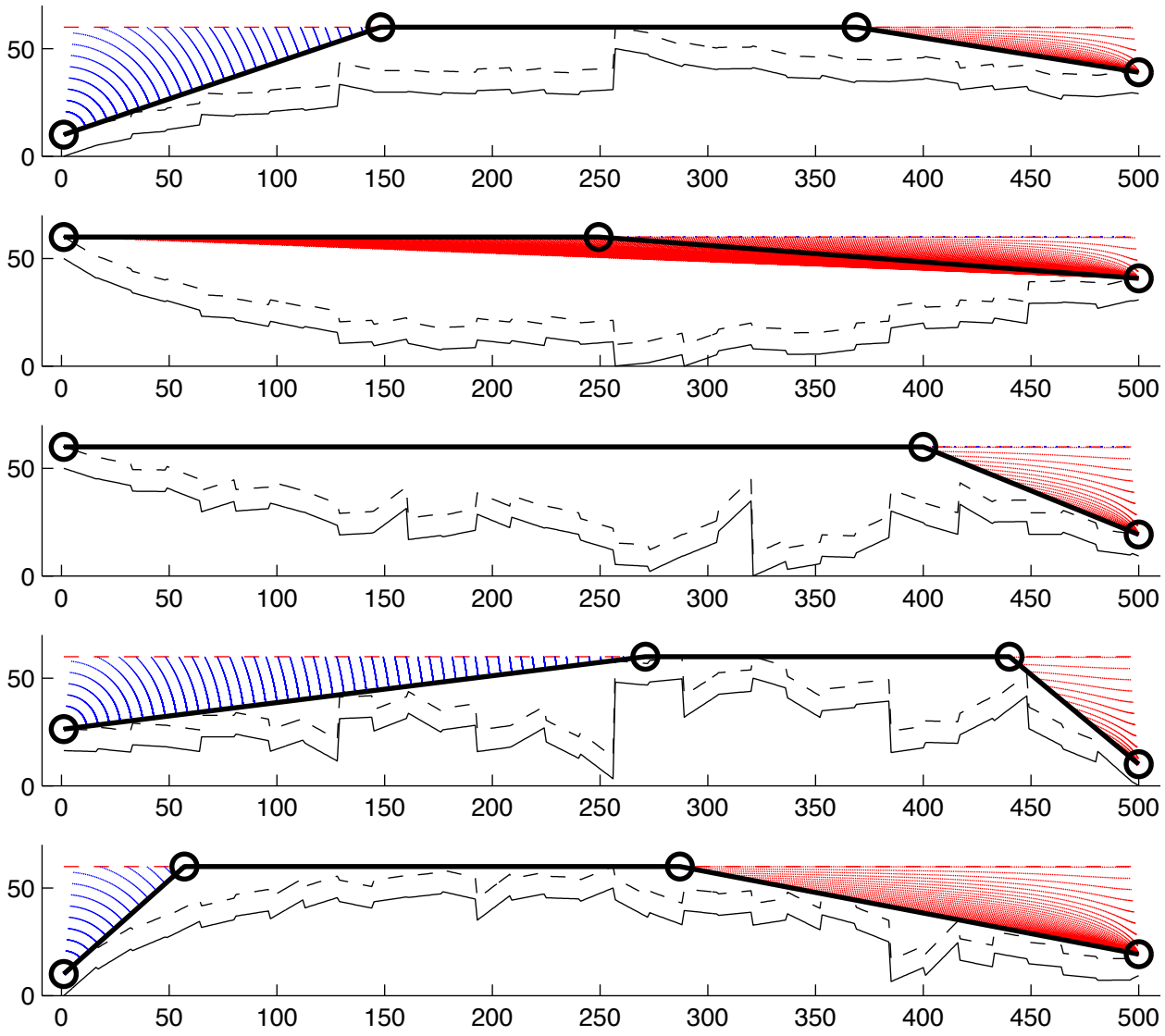


Figure 6.4: Ingress path optimization results

lower forward speed, and the forward speed drives the main component of the power curve at high forward speeds. The algorithm therefore directs the helicopter to descend at the maximum rate while also maintaining the maximum forward speed. Also note that the left and right break points are coincident because the initial point was already the highest.

Figure 6.4(c) shows a different result for a similar map. In this case, the descent region is bounded by the terrain, but the algorithm still maximized the time spent descending. In Figure 6.4(d), the descent region is very short and steep, while the climb path is limited by the terrain. Note that the same power considerations apply to ascending as descending. Finally, Figure 6.4(e) demonstrates a shortcoming of the strict ground clearance: in the case of gently-sloping terrain such as between 0 and 200 meters in this trial, the ascent region is very short and power-consuming as the helicopter climbs to avoid the ground. An algorithm with high-quality real data may be able to determine when breaking the ground clearance rule was safe enough to risk, and thereby save energy and prolong its flight time.

It is clear that the nonlinear optimization algorithm presented above, when applied to the required power calculations developed in this chapter, yields a feasible minimum solution that can guide a helicopter on an energy-optimal path from the launch area to the area of interest. It is useful to note that, for a given terrain contour, the paths are optimal in both directions. That is, after a contour has been detected and traced, the same solution is energy-optimal from the destination back to the origin.

Chapter 7

Conclusion

This thesis presents the research, development, implementation, and testing of a complete unmanned helicopter-based mission to detect, localize, and analyze an arbitrary number of uncharacterized radioactive point sources. The unmanned aerial vehicle platform, the Yamaha RMAX helicopter, presented a unique set of constraints with respect to its size, payload capacity, and endurance limits. The algorithms developed in this thesis successfully operate within the constraints imposed by the platform. Specifically, using an inexpensive, lightweight, nondirectional NaI radiation detector provided by Sandia National Laboratories, the procedures developed above apply nonlinear optimization, optimal information gain, recursive Bayesian estimators, and a novel contour analysis algorithm to accurately estimate the position and activity of an unknown number of radioactive sources. This chapter summarizes the contributions of this thesis in detail in Section 7.1 and suggests future avenues of exploration in Section 7.2.

7.1 Summary of Contributions

The everpresent threat of nuclear catastrophe or attack provides a demand for the ability to quickly and safely detect, localize, and characterize radioactive material. An unmanned aerial vehicle (UAV) is the ideal platform for this task, as it avoids rubble or other changed topography in the aftermath of an event and keeps human operators safely out of the danger of radiation contamination. The Unmanned Systems Laboratory (USL) at Virginia Tech has outfitted a Yamaha RMAX helicopter with detection equipment to carry out this mission. The process of localizing sources of radioactivity relies heavily on direction finding, as gamma rays are emitted uniformly from a source; while the SNL detector did not have the requisite directionality to aid in localization, two algorithms were shown to successfully localize at least one source despite this limitation.

A grid-based recursive Bayesian estimator (RBE) was developed that localizes a single radioactive source of known intensity with zero error within a mean of 25 s ($\sigma = 4.90$) after the initial detection of the source. Removing all of the assumptions that allowed the grid-based RBE to perform so efficiently necessitated the development of a mapping-based approach to radioactive source localization. Procedures were presented to quickly trace a level contour of radiation counts and apply a customized implementation and expansion of the Hough transform to localize an arbitrary number of sources. Tracing the contour generated by a single weak source, a single strong source, two equal sources, two unequal sources, and three unequal sources was completed in roughly 5 minutes, 20 minutes, 25 minutes, 27 minutes,

and 17 minutes, respectively. All of these times are within the flight-time constraints of this vehicle, and easily may be adapted to other vehicles with different constraints. The contour analysis algorithm was demonstrated through simulation to localize sources within a mean of less than 20 m ($\sigma = 12.11$) with a minimum at zero error. That procedure also successfully identified through confidence analysis the number of sources present for all five cases, proving efficacy up to three sources and suggesting fitness for a greater number of sources. An iterative source intensity compensation algorithm was presented that corrected raw source intensity estimates by as much as 30% to produce relative source strength errors of 25% or less.

The customized grid-based technique is a highly accurate application of RBE with a completely nondirectional detector on an unmanned aerial platform with limited computation ability. However, the truly novel contribution of this thesis to localization research is the contour analysis algorithm that removes the requirement to have specific prior knowledge of the source to be localized and limits the number of sources to one. This relaxation is especially important in emergency response situations where information is disorganized or unavailable and the response cannot wait for preliminary analysis. The contour analysis procedure permits emergency personnel to focus on other issues while a UAV autonomously investigates a field of interest and determines not only source presence, but the number and strength of the sources while simultaneously establishing an initial estimate of the spread of radiation from the event.

Both localization algorithms rely on efficient, rapid detection of the sources in a potentially

large field. This thesis presents a thorough, but time-efficient, search pattern customized to the helicopter's navigation capabilities and its detector's sensitivity.

Finally, the understanding that following a large radiation contour may require a large fraction of the UAV's fuel resources prompted investigation into optimal ingress and egress paths to reserve as much fuel as possible for the investigation itself. A nonlinear optimization problem was developed and solved to provide energy-optimal velocities across arbitrary terrain (with the assumption that travel from the origin to the destination must occur along only that vector with respect to the x-y plane).

The work presented in this thesis comprises a complete radiation detection, localization, and analysis mission from lift-off to touch-down.

7.2 Suggestions for Future Work

The contributions of this thesis improve significantly on previous work on radiation localization with inexpensive detectors on small robotic platforms. However, several aspects of the algorithms developed present new avenues of exploration to further improve their performances.

The primary need for expansion is in the robust testing and proving of the efficacy of these algorithms in a real experiment with unshielded nuclear sources. While the radioactivity was modeled accurately for the simulated isotopes and the helicopter and detector were both modeled carefully, no simulation can substitute for true experimentation. The source

configurations chosen for analysis in this thesis proved very useful in testing the limits of the contour analysis procedure and should be replicated in an experimental setting to validate these simulations.

The measure of radioactive intensity employed for all simulations in this thesis was the gross counts received at a given location. The SNL detector provides a complete spectrum of counts, however, and that additional data may offer additional accuracy. By focusing on only isotopes of interest and avoiding gross counts from other isotopes or Compton backscatter, the estimates of source locations and intensities may be improved. The decomposition of an energy spectrum into its component isotopes is ongoing work, potentially permitting the vehicle platform to autonomously identify and isolate the isotopes present in the search field. Until those algorithms are available, however, expert human operators can easily identify source composition via visual inspection of the spectrograph, giving the estimate improvement with minimal loss of autonomy.

These algorithms may also be applied to to sensing point sources of other kinds of emissions. Researchers have employed RBE methods for many years in search and rescue operations involving light or radio beacons. The grid-based method presented in Chapter 4 will also function well in that field. Using the contour analysis method developed in this thesis, any point source with a model relating received intensity to distance from the source, such as a biological or chemical agent released into still air, may be localized with similar efficiency and accuracy. These applications should be investigated to verify and expand these algorithms' utility in the localization of various materials.

It will be useful to investigate the performance of these algorithms when the source of radiation is not a point source but a distribution of radioactive material. The contour following will, of course, continue to be useful to determine safe areas for personnel, but its utility in localizing material for sample collection should be analyzed, and, if necessary, new techniques should be developed to appropriately model that situation. Preliminary considerations indicate that localization should proceed normally, and the resulting estimates would be the centroids of distributed areas.

The Hough transform implementation developed for the work in this thesis focused on localizing circle shapes in the contour image. However, the results from contour following, especially in the three-source configuration, demonstrate that the shapes are in fact ellipses (and in the case of a single source, the ellipse collapses into a circle, losing no generality). Analysis of the eccentricity of the ellipses thus localized should provide additional insight into the correct location and intensity of the sources generating the contour. A significant drawback to this expansion is the massive increase in computational complexity. An ellipse is parameterized by four values: the center (x_c, y_c), the semi-major axis length a, and the semi-minor axis length b. Adding a dimension to the search exponentially increases the required computation time, requiring more efficient calculation techniques to make this improvement feasible in a time-critical situation.

Localizing ellipses instead of circles may also improve the position estimate error. While the algorithms developed in this thesis successfully localized all sources, some errors were larger than desired. Some of these errors, especially in the three-source configuration, appear

consistent and predictable. Analysis of the predictability of these errors, possibly using the iterative intensity correction technique as a computational basis, may lead to extreme reductions in source position estimate errors in some cases.

Another option for improving the position estimates involves adding a gradient-based optimization step to the end of the Hough transform analysis. Such a global optimizer is intractable and unlikely to yield useful results when applied to the entire search space. However, after the contour analysis step, the number of states to optimize has been limited to (x, y, r) for each final source estimate, and an appropriate initial position has been found at which the gradient-based optimization may start for a rapid, accurate solution. This optimizer would use as its error function the sum of the position error incurred by the current configuration relative to the contour traced by the helicopter.

Finally, computing optimal ingress paths in this thesis required several assumptions that resulted in only constrained optimality. Expanding the search via a method such as A^* path planning may lead to even more efficient flight paths and therefore more time for the contour detection and analysis once the vehicle arrives at the search area.

The algorithms presented in this thesis represent significant utility to emergency response efforts in the aftermath of a nuclear disaster. Proving and expanding upon the contributions of the algorithms of this thesis along the avenues suggested above may lead to significant improvement in the accuracy and verifiability of the estimates of radioactive source location and intensity.

Bibliography

[1] K.H. Tan A.A. Kassim T. Tan. "A Comparitive Study of Efficient Generalised Hough Transform Techniques". In: *Image and Vision Computing* 17.10 (1999), pp. 737–748.

[2] Hal O. Anger and Donald H. Davis. "Gamma-Ray Detection Efficiency and Image Resolution in Sodium Iodide". In: *The Review of Scientific Instruments* 35.6 (1964), pp. 693–697.

[3] D.H. Ballard. "Generalizing the Hough Transform to Detect Arbitrary Shapes". In: *Pattern Recognition* 13.2 (1981), pp. 111–122.

[4] Frederic Bourgault, Tomonari Furukawa, and Hugh F. Durrant-Whyte. "Coordinated Decentralized Search for a Lost Target in a Bayesian World". In: *Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2003, pp. 48–53.

[5] Reinhard Braunstingl, Jokin Mujika, and Juan Pedro Uribe. "A Wall Following Robot With A Fuzzy Logic Controller Optimized By A Genetic Algorithm". In: *Proceedings of*

the 1995 IEEE International Conference on Fuzzy Systems. Vol. 5. Yokohama, Japan, 1995, pp. 77–82.

[6] Sean M. Brennan, Angela M. Mielke, and David C. Torney. "Radioactive Source Detection by Sensor Networks". In: *IEEE Transactions on Nuclear Science* 52.3 (2005), pp. 813–819.

[7] Eric Brewer. "Autonomous Localization of $1/R^2$ Sources Using an Aerial Platform". MA thesis. Blacksburg, VA: Virginia Polytechnic Institute and State University, 2009.

[8] J M Camhi and E N Johnson. "High-frequency steering maneuvers mediated by tactile cues: antennal wall-following in the cockroach". In: *Journal of Experimental Biology* 202.5 (1999), pp. 631–643.

[9] John Cardarelli, Mark Thomas, and Timothy Curry. "Environmental Protection Agency (EPA) airborne gamma spectrometry system for environmental and emergency response surveys". In: *Proc. SPIE*. Vol. 7812. San Diego, California, USA, 2010.

[10] M.Y. Cheng and C.C. Lee. "Motion controller design for contour-following tasks based on real-time contour error estimation". In: *Industrial Electronics, IEEE Transactions on* 54.3 (2007), pp. 1686–1695.

[11] R. Andres Cortez and Herbert G. Tanner. *Radiation Mapping Using Multiple Robots*. Tech. rep. University of New Mexico, 2008.

[12] R. Andres Cortez et al. "Smart Radiation Sensor Management". In: *IEEE Robotics & Automation Magazine* 15.3 (2008), pp. 85–93.

[13] Richard O. Duda and Peter E. Hart. "Use of the Hough Transformation to Detect Lines and Curves in Pictures". In: *Communications of the ACM* 15.1 (1972), pp. 11–15.

[14] Alberto Elfes. "Sonar-Based Real-World Mapping and Navigation". In: *IEEE Journal of Robotics and Automation* RA-3.3 (1987), pp. 249–265.

[15] James H. Galbraith and Donald F. Saunders. "Rock Classification by Characteristics of Aerial Gamma-Ray Measurements". In: *Journal of Geochemical Exploration* 18 (1983), pp. 49–73.

[16] N.J. Gordon, D.J. Salmond, and A.F.M. Smith. "Novel approach to nonlinear/non-Gaussian Bayesian state estimation". In: *Radar and Signal Processing, IEE Proceedings F* 140.2 (1993), pp. 107–113.

[17] R.L. Grasty, H. Mellander, and M. Parker. "Airborne Gamma-ray Spectrometer Surveying". In: *Technical Report Series*. 323. Vienna: IAEA, 1991.

[18] Ajith Gunatilaka, Branko Ristic, and Ralph Gailis. "On Localisation of a Radiological Point Source". In: *Proc. Information, Decision and Control* (2007), pp. 236–241.

[19] Paul V.C. Hough. "Machine Analysis of Bubble Chamber Pictures". In: *International Conference on High Energy Accelerators and Instrumentation*. 1959.

[20] Paul V.C. Hough. "Method and Means for Recognizing Complex Patterns". US Patent US 3069654 (US). 1962.

[21] James W. Howse, Lawrence O. Ticknor, and Kenneth R. Muske. "Least squares estimation techniques for position tracking of radioactive sources". In: *Automatica* 37 (2001), pp. 1727–1737.

[22] Han-Pang Huang and N. Harris McClamroch. "Time-Optimal Control for a Robotic Contour Following Problem". In: *IEEE Journal of Robotics and Automation* 4.2 (1988), pp. 140–149.

[23] R.E. Kalman. "A New Approach to Linear Filtering and Prediction Problems". In: *Transactions of the ASME—Journal of Basic Engineering* 82 (1960), pp. 35–45.

[24] Genshiro Kitagawa. "Monte Carlo Filter and Smoother for Non-Gaussian Nonlinear State Space Models". In: *Journal of Computational and Graphical Statistics* 5.1 (1996), pp. 1–25.

[25] Glenn F. Knoll. *Radiation Detection and Measurement*. 3rd. New York: John Wiley and Sons, 2000.

[26] A. Kumar et al. "Automated Sequential Search for Weak Radiation Sources". In: *14th Mediterranean Conference on Control and Automation, 2006. MED '06.* 2006, pp. 1–6.

[27] Nicholas Metropolis and S. Ulam. "The Monte Carlo Method". In: *Journal of the American Statistical Association* 44.247 (1949), pp. 335–341.

[28] Hans P. Moravec and Alberto Elfes. "High Resolution Maps from Wide Angle Sonar". In: *Proc. IEEE Int. Conf. Robotics and Automation*. 1985, pp. 116–121.

[29] Mark Morelande, Branko Ristic, and Ajith Gunatilaka. "Detection and parameter estimation of multiple radioactive sources". In: *10th International Conference on Information Fusion*. Quebec, 2007, pp. 1–7.

[30] Bruce R. Munson, Donald F. Young, and Theodore H. Okiishi. *Fundamentals of Fluid Mechanics*. Fifth. John Wiley and Sons, 2006.

[31] Kenneth R. Muske and James W. Howse. "Comparison of Recursive Estimation Techniques for Position Tracking Radioactive Sources". In: *Proceedings of the American Control Conference*. 2001, pp. 1656–1660.

[32] James A. Pitkin and Joseph S. Duval. "Design parameters for aerial gamma-ray surveys". In: *Geophysics* 45.9 (1980), pp. 1427–1439.

[33] R. W. Prouty. *Helicopter Performance, Stability, and Control*. Melbourne, FL: Krieger Publishing Company, 1995.

[34] D.C.W. Sanderson, A.J. Cresswell, and D.C. White. "The effect of flight line spacing on radioactivity inventory and spatial feature characteristics of airborne gamma-ray spectrometry data". In: *International Journal of Remote Sensing* 29.1 (2008), pp. 31–46.

[35] L.D. Stone, C.A. Barlow, and T.L. Corwin. *Bayesian Multiple Target Tracking*. Mathematics in Science and Engineering. Boston: Artech House, 1999.

[36] Sebastian Thrun, William Burgard, and Dieter Fox. *Probabilistic Robotics*. Cambridge,MA,USA: The MIT Press, 2005.

[37] El-Mane Wong, Frederic Bourgault, and Tomonari Furukawa. "Multi-vehicle Bayesian Search for Multiple Lost Targets". In: *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*. 2005, pp. 3169–3174.

[38] Qun Zhang et al. "Imaging of a Moving Target With Rotating Parts Based on the Hough Transform". In: *IEEE Transactions on Geoscience and Remote Sensing* 46.1 (2008), pp. 291–299.

Appendix A

Grid-Based Localization Code

A.1 1-D Grid-Based RBE

The one-dimensional grid-based RBE example given in Section 4.2 may be recreated by running `grid_based_localization_1d.m`, which requires `get_source_counts.m`.

Listing A.1: grid_based_localization_1d.m

```
% Example of grid-based localization method in one dimension

close all;
clear variables;
clc;

doMovie = true;

% Establish grid
minX = -200;
maxX = 200;
cellResolution = .1;

grid.x = minX:cellResolution:maxX;
grid.y = zeros(size(grid.x));
grid.prior = (1 / numel(grid.x)) * ones(size(grid.x));
grid.likelihood = ones(size(grid.prior));

% Initialize target
target.x = 100;
target.y = 0;
target.z = 0;
target.type = 5;

[target.E1, target.yield1, target.mu11, target.mu12, target.At1,~] =...
    GetGammas(target.type,1.7);

% Initialize sensor
sensor.x = -50;
sensor.y = 0;
```

```
sensor.z = 60; % m
sensor.background = 1000;

% Find maximum detection distance for this target
%  This distance is in the x-y plane
sensor.maxDetectionDistance =...
    fminsearch(@(x)(abs(get_source_counts(0,0,sensor.z,target,x,0) -...
                        sensor.background)),...
               sensor.z);

% Initialize controls and history arrays
control.x = -0.5;
control.y = 0;

positionEstimateIndex = (maxX - minX) / 2;
positionEstimates     = grid.x(positionEstimateIndex);
positionErrors        = positionEstimates(1) - target.x;
confidence            = grid.prior(positionEstimateIndex);
sourceDetectedHist    = false;
elapsedTimeHist       = 0;

% Initialize plots
figure;

priorPlot = subplot(5,1,1:4);
% Plot
priorHandle = semilogy(grid.x,grid.prior,'.-','MarkerSize',20,'LineWidth',2);
hold on;
% Plot settings
xlim([minX maxX]);
ylim([10^-6 1]);
ylabel('Prior PDF Value');
set(priorPlot,'YTick',[]);
set(priorPlot,'xticklabel',[]);
set(priorPlot,'Box','off');

semilogy([minX maxX],[0.8 0.8],'k--','LineWidth',3);

positionPlot = subplot(5,1,5);
hold on;
% Plots
estimateHandle = plot(positionEstimates,0,'r^','MarkerSize',20,...
                                               'MarkerFaceColor','r');
sensorHandle   = plot(sensor.x,0,'k.','MarkerSize',20,'MarkerFacecolor','k');
% Plot settings
xlim([minX maxX]);
%xlabel('Meters');
p = get(positionPlot,'Pos');
p(2) = p(2) + 0.045;
set(positionPlot,'Pos',p);
set(positionPlot,'Box','off');
set(positionPlot,'TickLength',[0 0]);
set(positionPlot,'YTick',[]);
set(positionPlot,'XTick',[]);

if(doMovie == true)
    % Set up movie recorder
    aviobj = avifile('grid_based_1d_movie.avi','fps',12);
end

%% Main loop
curIteration = 1;
sourceFound = false;
while(sourceFound == false)
```

```
    tic;
    curIteration = curIteration + 1;
    turnedAround = false;

    % Get current observation
    sensorToTargetDistance = sqrt((sensor.x - target.x)^2 +...
                                  (sensor.y - target.y)^2 +...
                                  (sensor.z - target.z)^2);
    observedCounts = get_source_counts(sensor.x,sensor.y,sensor.z, target);

    % Calculate likelihoods
    if(observedCounts > sensor.background)
        % Source is detected
        sourceDetected = true;

        expectedCounts = get_source_counts(sensor.x, sensor.y, sensor.z,...
                                           target,...
                                           grid.x, grid.y);
        grid.likelihood = poisspdf(floor(observedCounts),...
                                   floor(expectedCounts));
    else
        % Source is not detected
        sourceDetected = false;

        expectedCounts = get_source_counts(sensor.x, sensor.y, sensor.z,...
                                           target,...
                                           grid.x,grid.y);
        probabilityOfDetection = 1 - poisscdf(sensor.background,...
                                              expectedCounts);
        grid.likelihood = 1 - probabilityOfDetection;
    end

    % Update prior
    grid.prior = grid.prior .* grid.likelihood;

    % Normalize prior
    grid.prior = grid.prior ./ sum(grid.prior(:));

    % Calculate source position estimate
    sourceEstimateIndex = find(grid.prior == max(grid.prior(:)),1,'first');
    sourceEstimate.x = grid.x(sourceEstimateIndex);
    sourceEstimate.confidence = grid.prior(sourceEstimateIndex);
    if(sourceEstimate.confidence > 0.8)
        sourceFound = true;
    end

    % Calculate control
    % In the 1-D case, just move toward the estimate
    if(sourceDetected == true)
        control.x = -(sensor.x - sourceEstimate.x);
        control.x = 0.5 * sign(control.x) * (control.x / control.x);
    else
        % Search to the edge of the field and back
        if(control.x >= 0)
            % Moving forward
            if(sensor.x + sensor.maxDetectionDistance >= maxX)
                % If we can detect the maximum x, turn around
                control.x = -control.x;
                turnedAround = true;
            end
        else
            % Moving backward
            if(sensor.x - sensor.maxDetectionDistance <= minX)
                % If we can detect the minimum x, turn around
```

```
                control.x = -control.x;
                turnedAround = true;
            end
        end
    end

    % Update sensor position
    sensor.x = sensor.x + control.x;

    % Update Records
    elapsedTimeHist(curIteration,:)    = toc;
    positionEstimates(curIteration,:)  = sourceEstimate.x;
    positionErrors(curIteration,:)     = sourceEstimate.x - target.x;
    confidence(curIteration,:)         = grid.prior(sourceEstimateIndex);
    sourceDetectedHist(curIteration,:) = sourceDetected;

    % Update graphs
    set(priorHandle,'YData',grid.prior);
    set(estimateHandle,'XData',sourceEstimate.x);
    set(sensorHandle,'XData',sensor.x);
    if(control.x < 0)
        set(sensorHandle,'Marker','<');
    else
        set(sensorHandle,'Marker','>');
    end
    if(sourceDetected == true)
        set(estimateHandle,'Marker','d');
    end
    drawnow;

    if((mod(curIteration,30) == 0) ||...
       (turnedAround == true))
        saveas(gcf,sprintf('grid_based_1d_%04d.png',curIteration));
        saveas(gcf,sprintf('grid_based_1d_%04d.pdf',curIteration));
    end

    if(doMovie == true)
        curFrame = getframe(gcf);
        aviobj = addframe(aviobj,curFrame);
    end

%     fprintf('Sensor position: %f\n', sensor.x);
%     fprintf('Source position: %f\n', target.x);
%     fprintf('Source estimate: %f\n', sourceEstimate.x);
%     fprintf('Estimate error : %f\n', sourceEstimate.x - target.x);
end

if(doMovie == true)
    aviobj = close(aviobj);
end

%% Plot results
iterations = 1:curIteration;
iterationsDetected   = iterations(sourceDetectedHist);
iterationsUndetected = iterations(~sourceDetectedHist);


% Time per iteration
figure;
hold on;

elapsedTimeHist(2) = elapsedTimeHist(3);
elapsedTimeDetected   = elapsedTimeHist(sourceDetectedHist);
elapsedTimeUndetected = elapsedTimeHist(~sourceDetectedHist);
```

```
plot(iterationsDetected,   elapsedTimeDetected,   'g-','LineWidth',2);
plot(iterationsUndetected, elapsedTimeUndetected, 'r-');

title({'1-D Grid-Based Localization','Time per Iteration'});
xlabel('Iterations');
ylabel('Iteration Time (s)');
xlim([0 curIteration]);
legend('Detected','Undetected');

saveas(gca,'grid_based_1d_time_per_iteration.png');
saveas(gca,'grid_based_1d_time_per_iteration.pdf');

% Position estimates
figure;
subplot(3,1,1);
hold on;

positionEstimatesDetected   = positionEstimates(sourceDetectedHist);
positionEstimatesUndetected = positionEstimates(~sourceDetectedHist);

plot(iterationsDetected,   positionEstimatesDetected,   'g-', 'LineWidth',2);
plot(iterationsUndetected, positionEstimatesUndetected, 'r-', 'LineWidth',1);

%title('1-D Grid-Based Localization');
ylabel('Source Estimate');
legend('Detected','Undetected','Location','Best');

xlim([0 curIteration]);
set(gca,'XTickLabel',[]);
set(gca,'Box','on');
p = get(gca,'Pos');
p(4) = 0.25;
set(gca,'Pos',p);

% Position estimate error
subplot(3,1,2);
hold on;

positionErrorsDetected   = positionErrors(sourceDetectedHist);
positionErrorsUndetected = positionErrors(~sourceDetectedHist);

plot(iterationsDetected,   positionErrorsDetected,   'g-', 'LineWidth',2);
plot(iterationsUndetected, positionErrorsUndetected, 'r-', 'LineWidth',1);

ylabel('Estimate Error (m)');
legend('Detected','Undetected','Location','Best');

xlim([0 curIteration]);
set(gca,'XTickLabel',[]);
set(gca,'Box','on');
p = get(gca,'Pos');
p(4) = 0.25;
set(gca,'Pos',p);

% Position estimate confidence
subplot(3,1,3);

confidenceDetected   = confidence(sourceDetectedHist);
confidenceUndetected = confidence(~sourceDetectedHist);

semilogy(iterationsDetected,   confidenceDetected,   'g-', 'LineWidth',2);
hold on;
semilogy(iterationsUndetected, confidenceUndetected, 'r-', 'LineWidth',1);
```

```
xlabel('Iterations');
ylabel('Confidence');
legend('Detected','Undetected','Location','Best');

xlim([0 curIteration]);
set(gca,'Box','on');
p = get(gca,'Pos');
p(4) = 0.25;
set(gca,'Pos',p);

saveas(gca,'grid_based_1d_results.png');
saveas(gca,'grid_based_1d_results.pdf');
```

Listing A.2: get_source_counts.m

```
function [Rs] = get_source_counts(x,y,h,target,x1,y1)
% Calculate total source count rate for detector at (x,y) due to 2 point
% sources
% INPUTS:
%           x   - helicopter x-position
%           y   - helicopter y-position
%           h   - helicopter altitude
%           At1 - ?
%           x1  - source x-location
%           y1  - source y-location
%
% dA=(0.1*sig)^2; %Element of area for integration
% Ad=.0413;  % Detector area for 4" x 16" face (m^2)
Ad=.0174;  % Detector area for 3"dia x 9" detector (m^2)
t=0.0508;  % Detector thickness for 2" length (m)

% This function may be parallelized to calculate theoretical observations
% from many points (x1(n),y1(n)). If this is the case, we use those point
% matrices; otherwise, we use the actual target position.
if exist('x1','var') && exist('y1','var')
    havePointMatrix = true;
else
    havePointMatrix = false;
end

totalCounts = 0;
for curTarget = 1:length(target.x)
    Sum = 0;

    % Get values for current target
    At1    = target.At1(curTarget);
    E1     = target.E1(curTarget,:);
    yield1 = target.yield1(curTarget,:);
    mu11   = target.mu11(curTarget,:);
    mu12   = target.mu12(curTarget,:);

    if(havePointMatrix == false)
        x1 = target.x(curTarget);
        y1 = target.y(curTarget);
    end

    nmax = numel(E1) - sum(E1 == 0);  % Gets size of array E1

    for m = 1:nmax  % Index on energies

        A = At1 * yield1(m);
        % Distance from source point (x1,y1,0) to measurement point (x,y,h)
        r = sqrt((x-x1).^2 + (y-y1).^2 + h.^2.*ones(size(x1)));
```

```
        % Air attenuation between source point and detector
        f3 = exp(-mu11(m) .* r);
        % Falloff due to solid angle of detector wrt source
        f4 = h ./ (r .^ 3);
        % Fraction of incident gammas that are captured in detector
        f2 = 1 - exp(-mu12(m) .* (t .* r ./ h));

        Sum = Sum + A .* f3 .* f4 .* f2; % Add contributions

    end  % For m

    totalCounts = totalCounts + Sum*(Ad/(4*pi));
end

Rs = totalCounts;
end
```

A.2 2-D Grid-Based RBE

The two-dimensional grid-based RBE simulations presented in Section 4.3 may be recreated by running `grid_based_localization_2d.m`, which requires both `get_source_counts.m` and `get_velocity_input.m`. The file `get_source_counts.m` requires three proprietary functions to define the properties of a radioactive source; these may be replaced by any function that will return an appropriate simulated count rate.

Listing A.3: grid_based_localization_2d.m

```
% Example of grid-based localization method in two dimensions

close all;
clear variables;
clc;

% Establish grid
minX = -200;
maxX = 200;
minY = -200;
maxY = 200;
cellResolution = 5;

[grid.y grid.x] = meshgrid(minX:cellResolution:maxX,...
                           minY:cellResolution:maxY);
grid.prior = (1 / numel(grid.y)) * ones(size(grid.x));
grid.likelihood = ones(size(grid.prior));

% Initialize target
target.x = 90;
target.y = 90;
target.z = 0;
target.type = 5;

[target.E1, target.yield1, target.mu11, target.mu12, target.At1,~] =...
    GetGammas(target.type, 1.7);

% Initialize sensor
sensor.x = -100;
sensor.y = -50;
sensor.z = 60;
sensor.background = 1000;

% Find maximum detection distance for this target
%  This distance is in the x-y plane
sensor.maxDetectionDistance =...
    fminsearch(@(x)(abs(get_source_counts(0,0,sensor.z,target,x,0) -...
                        sensor.background)),...
               sensor.z);

% Initialize controls and history arrays
control.x = 0.5;
control.y = 0.5;
```

```
positionEstimateIndex.x = ((maxX/cellResolution) - (minX/cellResolution)) / 2;
positionEstimateIndex.y = ((maxY/cellResolution) - (minY/cellResolution)) / 2;
positionEstimates.x     = grid.x(1,positionEstimateIndex.x);
positionEstimates.y     = grid.y(1,positionEstimateIndex.y);
positionErrors.x        = positionEstimates.x - target.x;
positionErrors.y        = positionEstimates.y - target.y;
confidence              = grid.prior(positionEstimateIndex.y,...
                                     positionEstimateIndex.x);
sourceDetectedHist      = false;
elapsedTimeHist         = 0;

% Initialize plots
mapPlot = figure;
hold on;
priorHandle    = surf(grid.x,grid.y,grid.prior,...
                      'EdgeColor','none');
estimateHandle = plot3(positionEstimates.x,...
                       positionEstimates.y,...
                       1,...
                       'r.',...
                       'MarkerSize',20);
sensorHandle   = plot3(sensor.x,...
                       sensor.y,...
                       1,...
                       'k.',...
                       'MarkerSize',20);
view([0 90]);

% Set up movie recorder
framesPerSecond = 5;
aviobj = avifile('grid_based_2d_movie.avi','fps',framesPerSecond);

%% Main loop
curIteration = 1;
detectionIteration = 0;
sourceFound = false;
while(sourceFound == false)
    tic;
    curIteration = curIteration + 1;

    % Get current observation
    sensorToTargetDistance = sqrt((sensor.x - target.x)^2 +...
                                  (sensor.y - target.y)^2 +...
                                  (sensor.z - target.z)^2);
    observedCounts = get_source_counts(sensor.x,sensor.y,sensor.z, target);

    % Calculate likelihoods
    if(observedCounts > sensor.background)
        % Source is detected
        if(sourceDetected == false)
            % this is the first detection iteration, so mark it
            detectionIteration = curIteration;
        end
        sourceDetected = true;

        expectedCounts = get_source_counts(sensor.x, sensor.y, sensor.z,...
                                           target,...
                                           grid.x, grid.y);
        grid.likelihood = poisspdf(floor(observedCounts),...
                                   floor(expectedCounts));
    else
        % Source is not detected
        sourceDetected = false;
```

```
        expectedCounts = get_source_counts(sensor.x, sensor.y, sensor.z,...
                                           target,...
                                           grid.x, grid.y);
        probabilityOfDetection = 1 - poisscdf(sensor.background,...
                                              expectedCounts);
        grid.likelihood = 1 - probabilityOfDetection;
    end

    % Update prior
    grid.prior = grid.prior .* grid.likelihood;

    % Normalize prior
    grid.prior = grid.prior ./ sum(grid.prior(:));

    % Calculate source position estimate
    [sourceEstimateIndex.x sourceEstimateIndex.y] =...
        find(grid.prior == max(grid.prior(:)),1,'first');
    sourceEstimate.x = grid.x(sourceEstimateIndex.x, 1);
    sourceEstimate.y = grid.y(1, sourceEstimateIndex.y);
    sourceEstimate.confidence = grid.prior(sourceEstimateIndex.y,...
                                           sourceEstimateIndex.x);
    if(sourceEstimate.confidence > 0.9)
        sourceFound = true;
    end

    % Calculate control
    % In the 2-D case, we want to circle the estimate
    if(sourceDetected == true)
        sensorToSourceVector = [sourceEstimate.x sourceEstimate.y] -...
                               [sensor.x sensor.y];
        control = get_velocity_input(sensorToSourceVector);
    else
        % Circle the center of the field instead
        sensorToCenterVector = [grid.x(((maxX/cellResolution) -...
                                        (minX/cellResolution)) / 2,1),...
                                grid.y(1,((maxY/cellResolution) -...
                                          (minY/cellResolution)) / 2)] -...
                               [sensor.x sensor.y];
        control = get_velocity_input(sensorToCenterVector);
    end

    % Update sensor position
    control = 4 * control ./ norm(control);
    sensor.x = sensor.x + control(1);
    sensor.y = sensor.y + control(2);

    % Update records
    elapsedTimeHist(curIteration)     = toc;
    positionEstimates.x(curIteration) = sourceEstimate.x;
    positionEstimates.y(curIteration) = sourceEstimate.y;
    positionErrors.x(curIteration)    = sourceEstimate.x - target.x;
    positionErrors.y(curIteration)    = sourceEstimate.y - target.y;
    confidence(curIteration)          = grid.prior(sourceEstimateIndex.y,...
                                                   sourceEstimateIndex.x);
    sourceDetectedHist(curIteration)  = sourceDetected;

    % Update graphs
    set(priorHandle,'ZData',grid.prior);
    set(estimateHandle,'XData',sourceEstimate.x);
    set(estimateHandle,'YData',sourceEstimate.y);
    set(sensorHandle,'XData',sensor.x);
    set(sensorHandle,'YData',sensor.y);
    if(confidence(curIteration) > 0.5)
```

```
        set(estimateHandle, 'MarkerFaceColor','g');
        set(estimateHandle, 'MarkerEdgeColor','g');
    end
    drawnow;

    curFrame = getframe(gcf);
    aviobj = addframe(aviobj,curFrame);

%     fprintf('Sensor position: (%9.4f,%9.4f)\n', sensor.x, sensor.y);
%     fprintf('Source position: (%9.4f,%9.4f)\n', target.x, target.y);
%     fprintf('Source estimate: (%9.4f,%9.4f)\n', sourceEstimate.x, sourceEstimate.y);
end

% Add some frames at the end for encoding purposes
%  (The Matlab Mac encoder is not excellent)
for k = 1:(5 * framesPerSecond)
    aviobj = addframe(aviobj,curFrame);
end
aviobj = close(aviobj);

% Iterations from detection to confidence > 90%
detectionIterations = curIteration - detectionIteration;

%% Plot results
iterations = 1:curIteration;
iterationsDetected   = iterations(sourceDetectedHist);
iterationsUndetected = iterations(~sourceDetectedHist);

figure;
hold on;

elapsedTimeHist(2) = elapsedTimeHist(3);
elapsedTimeDetected   = elapsedTimeHist(sourceDetectedHist);
elapsedTimeUndetected = elapsedTimeHist(~sourceDetectedHist);

plot(iterationsDetected,   elapsedTimeDetected,   'g-','LineWidth',2);
plot(iterationsUndetected, elapsedTimeUndetected, 'r-');

title({'2-D Grid-Based Localization','Time per Iteration'});
xlabel('Iterations');
ylabel('Iteration Time (s)');
xlim([0 curIteration]);
legend('Detected','Undetected','Location','Best');

saveas(gca,'grid_based_2d_time_per_iteration.png');
saveas(gca,'grid_based_2d_time_per_iteration.pdf');

% Position estimates
figure;
hold on;

positionEstimatesDetected.x   = positionEstimates.x(sourceDetectedHist);
positionEstimatesUndetected.x = positionEstimates.x(~sourceDetectedHist);
positionEstimatesDetected.y   = positionEstimates.y(sourceDetectedHist);
positionEstimatesUndetected.y = positionEstimates.y(~sourceDetectedHist);

plot(positionEstimatesDetected.x,   positionEstimatesDetected.y,   'd',...
    'MarkerEdgeColor','black','MarkerFaceColor','green',...
    'MarkerSize',10,'LineWidth',2);
plot(positionEstimatesUndetected.x, positionEstimatesUndetected.y, 'o',...
    'MarkerEdgeColor','black','MarkerFaceColor','red',...
    'MarkerSize',10,'LineWidth',2);

title({'2-D Grid-Based Localization','Source Position Estimates'});
```

```
legend('Detected','Undetected','Location','Best');

axis([minX maxX minY maxY]);

saveas(gca, 'grid_based_2d_source_estimates.png');
saveas(gca, 'grid_based_2d_source_estimates.pdf');

% Position estimate error
figure;
subplot(2,1,1);
hold on;

positionErrorsDetected   = sqrt(positionErrors.x(sourceDetectedHist).^2 +...
                                positionErrors.y(sourceDetectedHist).^2);
positionErrorsUndetected = sqrt(positionErrors.x(~sourceDetectedHist).^2 +...
                                positionErrors.y(~sourceDetectedHist).^2);

plot(iterationsDetected, positionErrorsDetected,     'g-', 'LineWidth',2);
plot(iterationsUndetected, positionErrorsUndetected, 'r-', 'LineWidth',1);

title('2-D Grid-Based Localization');
legend('Detected','Undetected','Location','Best');

xlim([0 curIteration]);
set(gca,'XTickLabel',[]);

% Position estimate confidence
subplot(2,1,2);
hold on;

confidenceDetected   = confidence(sourceDetectedHist);
confidenceUndetected = confidence(~sourceDetectedHist);

plot(iterationsDetected, confidenceDetected,     'g-', 'LineWidth',2);
plot(iterationsUndetected, confidenceUndetected, 'r-', 'LineWidth',1);

legend('Detected','Undetected','Location','Best');

xlim([0 curIteration]);
xlabel('Iterations');

saveas(gca,'grid_based_2d_results.png');
saveas(gca,'grid_based_2d_results.pdf');
```

Listing A.4: get_velocity_input.m

```
function u = get_velocity_input(sourceVector, radius,...
                                maxVelocity, circleDirection)

sourceVector = sourceVector(:);

if ~exist('radius','var')
    radius = 50;
end

if ~exist('maxVelocity','var')
    maxVelocity = 2;
end

if ~exist('circleDirection','var')
    circleDirection = 'counterclockwise';
end

% If we're on top of the estimated source, don't move
```

```
if norm(sourceVector) == 0
    u = [0;0];
else
    radiusCorrection = 0.1 * (norm(sourceVector) - radius);

    % Compute input
    if sourceVector(2) ~= 0
        % During the search, we would like to move on a line
        %  tangent to the circle centered on the source estimate
        %  that intersects the helicopter's current position
        % Multiplying by -sign(sourceVector(2)) ensures that the
        %  control will be in the counterclockwise direction
        tang = -sign(sourceVector(2)) * [-1; sourceVector(1) / sourceVector(2)];
        u = maxVelocity * tang / norm(tang);

        if(strcmp(circleDirection,'clockwise') == 1)
            % We want to circle clockwise, so we reverse direction
            u = -u;
        end

        u = u +...
            radiusCorrection * sourceVector / norm(sourceVector);
    else % our y value is the same as the source; move straight (up/down)
        if(strcmp(circleDirection,'counterclockwise') == 1)
            if(sourceVector(1) > 0)
                % we are on the left of the source, so move backward
                u = [0;-maxVelocity];
            else
                % we are on the right of the source, so move forward
                u = [0;maxVelocity];
            end
        else
            if(sourceVector(1) > 0)
                % left of source, move forward
                u = [0;maxVelocity];
            else
                % right of source, move backward
                u = [0;-maxVelocity];
            end
        end
    end
end
```

Appendix B

Contour Analysis Code

The contour following and analysis algorithms presented in Chapter 5 are implemented in `contour_following_demo.m` and `contour_following_poisson_demo.m`, which both require `follow_contour_only.m`, `get_poisson_counts.m`, and `get_radiation_counts.m`.

Listing B.1: contour_following_demo.m

```
% Demonstrate effect of changing speed and time step on
%  contour following performance

close all;
clear;
clc;

% set to true to get pretty pictures
%  otherwise, we collect large amounts of data
getPlots = false;

%% Set constants
sensor.x = 500;
sensor.y = 0;
sensor.z = 60;
sensor.yaw = pi;
sensor.yawRateLimit = 6*pi/180;
sensor.yawLimit = pi;

grid.minX = -500;
grid.maxX = 500;
grid.minY = -500;
grid.maxY = 500;

options.contourCounts = 2000;

if(getPlots == true)
    options.doPlots = true;
    dts = [.1 .25 .5 1];
    speeds = 10;
else
```

```
    options.doPlots = false;
    dts = [1 2 4 10].^-1;
    speeds = [0.5:0.5:4 5 10];
end

%% Super loop
for target_type = 1:4

switch(target_type)
    case 1
        extension = '_3_targets';
        targets = [-250 0 0 5000;...
                    0    0 0 10000;...
                    250  0 0 5000];
    case 2
        extension = '_2_equal_targets';
        targets = [-250 0 0 20000;...
                     250 0 0 20000;];
    case 3
        extension = '_2_unequal_targets';
        targets = [-250 0 0 50000;...
                     250 0 0 5000];
    case 4
        extension = '_1_big_target';
        targets = [0 0 0 50000];
    case 5
        extension = '_1_small_target';
        targets = [0 0 0 5000];
end

filename = ['contour_following' extension '.mat'];

results = zeros(length(dts) * length(speeds),6);
fullResults = cell(length(dts)*length(speeds),1);
%% Main loop
for curDtIndex = 1:length(dts)
    options.dt = dts(curDtIndex);
    for curSpeedIndex = 1:length(speeds)
        close all;

        sensor.speed = speeds(curSpeedIndex);

        if(getPlots == true)
            options.filename = sprintf('ideal_contour%s_%ds_%dmps.png',...
                                       extension,...
                                       floor(options.dt*100),...
                                       floor(sensor.speed*100));
        end

        curIteration = (curDtIndex - 1)*length(speeds)+curSpeedIndex;

        fprintf('Iteration %4d: dt=%4.2f, speed=%4.2f\n',...
                curIteration,options.dt,sensor.speed);

        outerTic = tic;
        curResult = follow_contour_only(sensor,targets,grid,options);
        toc(outerTic);

        results(curIteration,:) = [options.dt, sensor.speed,...
                                   curResult.totalIterations,...
                                   curResult.contourIterations,...
                                   mean(abs(curResult.error.count(:))),...
                                   median(abs(curResult.error.count(:)))];
       fullResults{curIteration} = curResult;
```

```
        fprintf('    dt: %3.2f -- sp: %3.2f -- iter: %5d -- err: %9.4f -- errmed: %9.4f\n',...
                results(curIteration,1),...
                results(curIteration,2),...
                results(curIteration,4),...
                results(curIteration,5),...
                results(curIteration,6));
    end
end

if(getPlots == false)
    save(['full_' filename]);
end

end

break;

%% Plot results
close all; %#ok<UNRCH>
clear;
clc;

for target_type = 1:5

switch(target_type)
    case 1
        filename = 'contour_following_3_targets.mat';
        extension = '_3_targets';
    case 2
        filename = 'contour_following_2_equal_targets.mat';
        extension = '_2_equal_targets';
    case 3
        filename = 'contour_following_2_unequal_targets.mat';
        extension = '_2_unequal_targets';
    case 4
        filename = 'contour_following_1_big_target.mat';
        extension = '_1_big_target';
    case 5
        filename = 'contour_following_1_small_target.mat';
        extension = '_1_small_target';
end

filename = ['long_' filename];
load(filename,'results','speeds','dts');

% Header
fprintf('+--------------------------------+\n');
fprintf('| %30s |\n',extension);
fprintf('+--------------------------------+\n');

% Contour Time
contourIterations = reshape(results(:,4),length(speeds),length(dts));
contourTimes = contourIterations .*...
               repmat(dts,size(contourIterations,1),1);
meanContourTimes = mean(contourTimes,2);

% plot(results(1:length(speeds),2),meanContourTimes,'.-','MarkerSize',10)
% title('Time Required to Trace Contour (After Detection)');
% xlabel('Speed (m/s)');
% ylabel('Time (s)');

%saveas(gca,['contour_time' extension '.png']);
%saveas(gca,['contour_time' extension '.pdf']);
```

```

for k = 1:length(meanContourTimes)
    fprintf('Speed: %9.4f -- Time: %9.4f\n',results(k,2),meanContourTimes(k));
end

% Mean Error
figure;
hold on;

styles = {'b^','gs','r+','ko'};

meanErrors = reshape(results(:,5),length(speeds),length(dts));
medianErrors = reshape(results(:,6),length(speeds),length(dts));

for k = 1:size(meanErrors,2)
    plot(results(1:length(speeds),2),medianErrors(:,k),styles{k},'MarkerSize',10)
end
set(gca,'YScale','log');
%title('Mean Count Error');
xlabel('Speed (m/s)');
ylabel('Median Error (Counts)');

legendStrings = cell(1,length(dts));
for k = 1:length(dts)
    legendStrings{k} = sprintf('%2d Hz',1/dts(k));
end
hLegend = legend(legendStrings,'Location','SouthEast');
set(get(hLegend,'title'),'string',{'Update','Frequency'});
%saveas(gca,['ideal_contour_median_error' extension '.png']);
saveas(gca,['ideal_contour_median_error' extension '.pdf']);

end
```

Listing B.2: contour_following_poisson_demo.m

```
% Demonstrate effect of changing speed and time step on
%  contour following performance

close all;
clear;
clc;

% set to true to get pretty pictures
%  otherwise, we collect large amounts of data
getPlots = true;

%% Set constants
sensor.x = 500;
sensor.y = 0;
sensor.z = 60;
sensor.yaw = pi;
sensor.yawRateLimit = 15*pi/180;
sensor.yawLimit = pi;

grid.minX = -500;
grid.maxX = 500;
grid.minY = -500;
grid.maxY = 500;

options.contourCounts = 2000;
options.poisson = true;

if(getPlots == true)
    options.doPlots = true;
```

```
    dts = 1;
    speeds = 2;
else
    options.doPlots = false;
    dts = [1 2 4 10].^-1;
    speeds = [0.5:0.5:4 5 10];
end

%% Super loop
for target_type = 1:5

switch(target_type)
    case 1
        extension = '_3_targets';
        targets = [-250 0 0 5000;...
                    0    0 0 10000;...
                    250  0 0 5000];
    case 2
        extension = '_2_equal_targets';
        targets = [-250 0 0 20000;...
                    250 0 0 20000;];
    case 3
        extension = '_2_unequal_targets';
        targets = [-250 0 0 50000;...
                    250 0 0 5000];
    case 4
        extension = '_1_big_target';
        targets = [0 0 0 50000];
    case 5
        extension = '_1_small_target';
        targets = [0 0 0 5000];
end

filename = ['contour_following_poisson' extension '.mat'];

results = zeros(length(dts) * length(speeds),6);
fullResults = cell(length(dts) * length(speeds),1);
%% Main loop
for curDtIndex = 1:length(dts)
    options.dt = dts(curDtIndex);
    for curSpeedIndex = 1:length(speeds)
        close all;

        sensor.speed = speeds(curSpeedIndex);

        if(getPlots == true)
            options.filename = sprintf('poisson_contour%s_%ds_%dmps.png',...
                                       extension,...
                                       floor(options.dt*100),...
                                       floor(sensor.speed*100));
        end

        curIteration = (curDtIndex - 1)*length(speeds)+curSpeedIndex;

        fprintf('Iteration %4d: dt=%4.2f, speed=%4.2f\n',...
                curIteration,options.dt,sensor.speed);

        outerTic = tic;
        curResult = follow_contour_only(sensor,targets,grid,options);
        toc(outerTic);

        results(curIteration,:) = [options.dt, sensor.speed,...
                                   curResult.totalIterations,...
                                   curResult.contourIterations,...
```

```
                                    mean(abs(curResult.error.count(:))),...
                                    median(abs(curResult.error.count(:)))];
        fullResults{curIteration} = curResult;
        fprintf('    dt: %3.2f -- sp: %3.2f -- iter: %5d -- err: %9.4f -- errmed: %9.4f\n',...
                results(curIteration,1),...
                results(curIteration,2),...
                results(curIteration,4),...
                results(curIteration,5),...
                results(curIteration,6));
    end
end

if(getPlots == false)
    save(filename);
end

end

break;

%% Plot results
close all; %#ok<UNRCH>
clear;
clc;

for target_type = 1:5

switch(target_type)
    case 1
        filename = 'contour_following_poisson_3_targets.mat';
        extension = '_3_targets';
        targets = [-250 0 0 5000;...
                   0    0 0 10000;...
                   250  0 0 5000];
    case 2
        filename = 'contour_following_poisson_2_equal_targets.mat';
        extension = '_2_equal_targets';
        targets = [-250 0 0 20000;...
                    250 0 0 20000;];
    case 3
        filename = 'contour_following_poisson_2_unequal_targets.mat';
        extension = '_2_unequal_targets';
        targets = [-250 0 0 50000;...
                    250 0 0 5000];
    case 4
        filename = 'contour_following_poisson_1_big_target.mat';
        extension = '_1_big_target';
        targets = [0 0 0 50000];
    case 5
        filename = 'contour_following_poisson_1_small_target.mat';
        extension = '_1_small_target';
        targets = [0 0 0 5000];
end

load(filename);

% Header
fprintf('+--------------------------------+\n');
fprintf('| %30s |\n',extension);
fprintf('+--------------------------------+\n');

% Contour Time
contourIterations = reshape(results(:,4),length(speeds),length(dts));
contourTimes = contourIterations .*...
```

```
            repmat(dts,size(contourIterations,1),1);
meanContourTimes = mean(contourTimes,2);

% plot(results(1:length(speeds),2),meanContourTimes,'.-','MarkerSize',10)
% title('Time Required to Trace Contour (After Detection)');
% xlabel('Speed (m/s)');
% ylabel('Time (s)');

% saveas(gca,['contour_time' extension '.png']);
% saveas(gca,['contour_time' extension '.pdf']);

for k = 1:length(meanContourTimes)
    fprintf('Speed: %9.4f -- Time: %9.4f\n',results(k,2),meanContourTimes(k));
end

% Mean Error
figure;
hold on;

styles = {'b^','gs','r+','ko'};

meanErrors = reshape(results(:,5),length(speeds),length(dts));
medianErrors = reshape(results(:,6),length(speeds),length(dts));

for k = 1:size(meanErrors,2)
    plot(results(1:length(speeds),2),medianErrors(:,k),styles{k},'MarkerSize',10)
end
%set(gca,'YScale','log');
%title('Mean Count Error');
xlabel('Speed (m/s)');
ylabel('Median Error (Counts)');

legendStrings = cell(1,length(dts));
for k = 1:length(dts)
    legendStrings{k} = sprintf('%2d Hz',1/dts(k));
end
hLegend = legend(legendStrings,'Location','SouthEast');
set(get(hLegend,'title'),'string',{'Update','Frequency'});
saveas(gca,['poisson_contour_median_error' extension '.png']);
saveas(gca,['poisson_contour_median_error' extension '.pdf']);

% Distance errors
[grid.x grid.y] = meshgrid(-500:500,-500:500);
grid.z = 60 * ones(size(grid.x));

counts = get_radiation_counts(targets,...
                              [grid.x(:) grid.y(:) grid.z(:)],...
                              [size(grid.x,1) size(grid.y,2)]);

[C,~] = contour(grid.x,grid.y,counts,[2000 2000]);
[cstruct cfix] = splitcontours(C);

contourPoints = [cstruct(1).x(:) cstruct(1).y(:)];

meanDistanceErrors = zeros(length(fulLResults),1);
medianDistanceErrors = zeros(length(fulLResults),1);
for k = 1:length(fulLResults)
    curResult = fulLResults{k};

    minDistances = zeros(size(curResult.contourPoints,1),1);
    for kk = 1:size(curResult.contourPoints,1)
        minDistances(kk) =...
            min(sqrt((curResult.contourPoints(kk,1)-contourPoints(:,1)).^2 +...
                     (curResult.contourPoints(kk,2)-contourPoints(:,2)).^2));
```

```
    end
    meanDistanceErrors(k) = mean(minDistances);
    medianDistanceErrors(k) = median(minDistances);
end

meanDistanceErrors = reshape(meanDistanceErrors,length(speeds),length(dts));
medianDistanceErrors = reshape(medianDistanceErrors,length(speeds),length(dts));

figure;
hold on;
for k = 1:size(meanErrors,2)
    plot(results(1:length(speeds),2),medianDistanceErrors(:,k),styles{k},'MarkerSize',10)
end
xlabel('Speed (m/s)');
ylabel('Mean Error (m)');

legendStrings = cell(1,length(dts));
for k = 1:length(dts)
    legendStrings{k} = sprintf('%2d Hz',1/dts(k));
end
hLegend = legend(legendStrings,'Location','SouthEast');
set(get(hLegend,'title'),'string',{'Update','Frequency'});

saveas(gca,['poisson_contour_median_error_distance' extension '.png']);
saveas(gca,['poisson_contour_median_error_distance' extension '.pdf']);

end
```

Listing B.3: follow_contour_only.m

```
function result = follow_contour_only(sensor,targets,grid,options)

%close all;

% Set default options
if(~exist('options','var'))
    options.contourCounts = 2000;
    options.dt = 1;
    options.doPlots = true;
    options.poisson = false;
end

if(~isfield(options,'filename'))
    options.filename = [];
end

if(~isfield(options,'poisson'))
    options.poisson = false;
end

if(~isfield(options,'kp'))
    options.kp = -23.664799315075129;
end
if(~isfield(options,'kd'))
    options.kd = -11.983513803719720;
end
if(~isfield(options,'ki'))
    options.ki =  -0.003682103144682;
end

% Initialize RMAX
if(~exist('sensor','var'))
    sensor.x = 500;
    sensor.y = 0;
```

```
    sensor.z = 60;
    sensor.yaw = pi;
end
if(~isfield(sensor,'speed'))
    sensor.speed = 1; % in m/s
end
if(~isfield(sensor,'yawRateLimit'))
    sensor.yawRateLimit = 6*pi/180; % in deg/s
end
if(~isfield(sensor,'yawLimit'))
    sensor.yawLimit = 2*pi; % in rad
end
sensor.yawRate = 0;

% Initialize radiation field
if(~exist('grid','var'))
    grid.minX = -500;
    grid.maxX = 500;
    grid.minY = -500;
    grid.maxY = 500;
end
[grid.x grid.y] = meshgrid(grid.minX:grid.maxX,grid.minY:grid.maxY);
grid.z = sensor.z * ones(size(grid.x));

if(~exist('targets','var'))
    targets = [-250 0 0 5000;...
                0    0 0 10000;...
                250  0 0 5000];
end

counts = get_radiation_counts(targets,...
                              [grid.x(:),grid.y(:),grid.z(:)],...
                              [size(grid.x,1) size(grid.y,2)]);

sigma = sqrt(options.contourCounts);
%countTolerance = 3 * sigma; % 3*sigma, from Chebyshev's Inquality
%minCounts = options.contourCounts - countTolerance;
%maxCounts = options.contourCounts + countTolerance;
nominalContours = [options.contourCounts - (3*sigma)...
                   options.contourCounts - (2*sigma)...
                   options.contourCounts - (1*sigma)...
                   options.contourCounts...
                   options.contourCounts + (1*sigma)...
                   options.contourCounts + (2*sigma)...
                   options.contourCounts + (3*sigma)];

countLevels = union(nominalContours,...
                    1000:1000:10000);
if(options.doPlots == true)
    figureHandle = figure;
    contour(grid.x,grid.y,counts,countLevels);
    hold on;
    contourHandleOver  = plot(0,0,'gx');
    contourHandleUnder = plot(0,0,'r+');
    colorbar;

    rmaxHandle = plot(sensor.x,sensor.y,'go','MarkerSize',20);

    figure;
    errorHandle = plot(0,0,'.');
    title(sprintf('ki = %f -- kd = %f -- kp = %f',options.ki,options.kd,options.kp));
    xlabel('Iteration');
    ylabel('Count Error');
end
```

```
% Initialize state machine
machineState = 'spiral';

allPoints = [sensor.x sensor.y 0];

error.count = 0;
error.diff  = 0;
error.int   = 0;

sensor.totalYaw = 0;
sensor.isInsideContour = 0; % 1 if inside, 0 if outside
sensor.changedSides = false;

curContour.startX    = NaN;
curContour.startY    = NaN;
curContour.startYaw  = NaN;
curContour.startIter = NaN;

curContour.started = false; % have we found the contour?
curContour.canEnd  = false; % can we start looking for the end yet?
curContour.ended   = false; % have we found the end yet?
curContour.points  = [];

%% Main Loop
curIteration = 1;
while curContour.ended == false
    curIteration = curIteration + 1;

    % Get counts here
    curCount = get_poisson_counts(targets,[sensor.x sensor.y sensor.z],...
                                  options.poisson);
    lastCount = allPoints(end,3);

    % Record point
    allPoints(end+1,:) = [sensor.x sensor.y curCount];
    if(curContour.started == true)
        curContour.points(end+1,:) = [sensor.x sensor.y curCount];
    end
    switch(machineState)
        case 'spiral'
            % This is a placeholder for the actual spiral search algorithm
            sensor.yawRate = 0;

            if(curCount >= (options.contourCounts + sqrt(options.contourCounts)))
                % we are inside the contour by at least one sigma
                machineState = 'contour';
                curContour.startX    = allPoints(end,1);
                curContour.startY    = allPoints(end,2);
                curContour.startYaw  = sensor.yaw;
                curContour.startIter = curIteration;
                curContour.started   = true;
            end

            error.count(curIteration) = 0;
            error.diff(curIteration)  = 0;
            error.int(curIteration)   = 0;
        case 'contour'
            % Update error values
            error.count(curIteration) = (curCount - options.contourCounts)/sigma;
            error.diff(curIteration)  = curCount - lastCount;
            error.int(curIteration)   = error.int(curIteration-1) +...
                                        error.count(curIteration);
```

```
            % Determine yaw rate based on error
            sensor.yawRate = options.kp/sigma * error.count(curIteration) +...
                             options.kd/sigma * error.diff(curIteration) +...
                             options.ki/sigma * error.int(curIteration);

            sensor.yawRate =...
                min(abs(sensor.yawRate),...
                    abs(sensor.yawRateLimit)) * sign(sensor.yawRate);

            % Check whether we have switched contour sides
            if((sensor.isInsideContour  && (curCount <= options.contourCounts)) ||...
               (~sensor.isInsideContour && (curCount >= options.contourCounts)))
                % We just changed sides of the contour
                sensor.changedSides = true;
                sensor.isInsideContour = 1 - sensor.isInsideContour;

                % Reset the yaw counter to whatever the current value is
                sensor.totalYaw = sensor.yawRate * options.dt;

                % Reset error integration
                error.int(curIteration) = 0;
            else
                % We are on the same side as the last iteration
                sensor.changedSides = false;

                % Keep counting yaw
                sensor.totalYaw = sensor.totalYaw + sensor.yawRate * options.dt;
            end

            % Check whether we have exceeded the yaw limit
            if(abs(sensor.totalYaw) >= sensor.yawLimit)
                if(false)%options.poisson == true)
                    % For Poisson model, apply all the time
                    sensor.yawRate = 0;
                else
                    % Otherwise, only on the inside
                    if(sensor.isInsideContour == true)
                        sensor.yawRate = 0;
                    end
                end
            end

            % Check whether we can start looking for the end of the contour
            if(curContour.canEnd == false)
                if(sensor.yaw > curContour.startYaw + pi)
                    % We have turned around completely, so we can start
                    %  looking for the end
                    curContour.canEnd = true;
                end
            end

            % Check whether we have returned to the start
            if(curContour.canEnd == true)
                distanceToStart = sqrt((sensor.x - curContour.startX)^2 +...
                                       (sensor.y - curContour.startY)^2);
%                 if(distanceToStart < 40)
%                     curContour.ended = true;
%                 end
                if(sensor.yaw > curContour.startYaw + 2*pi)
                    curContour.ended = true;
                end
            end
    end
```

```
%     fprintf('Counts:    %9.4f\n',  curCount);
%     fprintf('Error:     %9.4f\n',  error.count(curIteration));
%     fprintf('Diff:      %9.4f\n',  error.diff(curIteration));
%     fprintf('Int:       %9.4f\n\n',error.int(curIteration));
%
%     fprintf('Yaw Rate:  %9.4f\n',sensor.yawRate);
%     fprintf('Total Yaw: %9.4f\n\n',sensor.totalYaw);

    % Move the sensor
    sensor.x   = sensor.x   + (options.dt * sensor.speed * cos(sensor.yaw));
    sensor.y   = sensor.y   + (options.dt * sensor.speed * sin(sensor.yaw));
    sensor.yaw = sensor.yaw + (options.dt * sensor.yawRate);

    % Update the graphs
    if(options.doPlots == true)
        set(rmaxHandle,'XData',sensor.x);
        set(rmaxHandle,'YData',sensor.y);
        if(~isempty(allPoints))
            set(contourHandleUnder, 'XData',...
                allPoints(allPoints(:,3) < options.contourCounts,1));
            set(contourHandleUnder, 'YData',...
                allPoints(allPoints(:,3) < options.contourCounts,2));

            set(contourHandleOver, 'XData',...
                allPoints(allPoints(:,3) >= options.contourCounts,1));
            set(contourHandleOver, 'YData',...
                allPoints(allPoints(:,3) >= options.contourCounts,2));

            if(curContour.started == true)
                set(errorHandle,'XData',curContour.startIter:curIteration);
                set(errorHandle,'YData',error.count(curContour.startIter:curIteration));
            end
        end

        % Only actually redraw the graphs every hundred points
        if(mod(length(allPoints),100) == 0)
            drawnow;
        end
    end
end

result.contourPoints = curContour.points;
result.totalIterations = curIteration;
result.contourIterations = curIteration - curContour.startIter;
result.error = error;

if(~isempty(options.filename))
    saveas(figureHandle,options.filename);
    saveas(figureHandle,strrep(options.filename,'png','pdf'));
end
```

Listing B.4: get_poisson_counts.m

```
function counts = get_poisson_counts(targets, sensor, doPoisson)
% Returns a count from the appropriate Poisson distribution
%
% Intended for use with a single sensor
%
% targets - an n-by-4 list of n targets, where each row is
%           [x y z intensity]
%           intensity should be the number of counts read directly over the
%            target at 60m
%
% sensor - a 1-by-3 [x y z] vector to get counts for
```

```
%
% counts - the counts received at this location
%
% doPoisson - true to draw from Poisson distribution
%             false to return ideal counts

if(~exist('doPoisson','var'))
    doPoisson = false;
end

numTargets = size(targets,1);

sensorList = kron(sensor,ones(numTargets,1));

distanceSquare = (sensorList(:,1)-targets(:,1)).^2 +...
                 (sensorList(:,2)-targets(:,2)).^2 +...
                 (sensorList(:,3)-targets(:,3)).^2;

counts = sum(targets(:,4) .* (60^2 ./ distanceSquare));

if(doPoisson == true)
    counts = poissrnd(counts);
else
    counts = floor(100*counts)/100;
end
```

Listing B.5: get_radiation_counts.m

```
function counts = get_radiation_counts(targets, sensorPoints, gridSize)
% Creates a grid of radiation values given a list of targets
%
% targets - an n-by-4 list of n targets, where each row is
%           [x y z intensity]
%           intensity should be the number of counts read directly over the
%            target at 60m
% sensorPoints - an m-by-3 list of points to get counts for
%                [x y z]
%
% counts - a 1-by-m matrix of counts for each sensor
%          [s1 s2 s3 ... sn]
%
%          if gridSize is passed, the 'counts' output is that size

numTargets = size(targets,1);
numSensors = size(sensorPoints,1);

targetList = repmat(targets,numSensors,1);
sensorList = kron(sensorPoints,ones(numTargets,1));

distances = sqrt((sensorList(:,1)-targetList(:,1)).^2 +...
                 (sensorList(:,2)-targetList(:,2)).^2 +...
                 (sensorList(:,3)-targetList(:,3)).^2);

counts = targetList(:,4) .* (60^2 ./ distances.^2);
counts = reshape(counts,numTargets,numSensors);
counts = sum(counts,1);

if exist('gridSize','var')
    counts = reshape(counts,gridSize);
end
```

Appendix C

Ingress Path Optimization Code

The terrain generation and flight path optimization presented in Chapter 6 are implemented in `optimize_ingress_path.m`, which requires `energy_optimizer.m`, `get_total_energy.m`, `get_required_power.m`, `generate_random_terrain.m`, and `simple_smooth_array.m`.

Listing C.1: optimize_ingress_path.m

```
% Find most efficient ingress path given random (x,z) terrain

close all;
clear;
clc;

% Get a random terrain
terrain.minX = 1;
terrain.maxX = 500;
terrain.resolution = 1;
terrain.smallestFeature = 10;
terrain.groundClearance = 10;
terrain.maxZ = 50;

terrain.x = terrain.minX:terrain.resolution:terrain.maxX;
terrain.z = generate_random_terrain(terrain.maxX - terrain.minX + terrain.resolution,...
                                    terrain.resolution,...
                                    terrain.smallestFeature);
terrain.z = (terrain.z - min(terrain.z))/max(terrain.z - min(terrain.z)) * terrain.maxZ;
terrain.z = transpose(terrain.z);

helicopter.minForwardSpeed  = 0.4; % m/s
helicopter.maxForwardSpeed  = 10;  % m/s
helicopter.maxVerticalSpeed = 4;   % m/s up or down

figure;
plot(terrain.x,terrain.z);
axis image;

%% Break point constraints
% The altitude of each breakpoint is terrain.groundClearance above the
```

```
%  tallest terrain feature
breakPoint.z = max(terrain.z) + terrain.groundClearance;

initialAltitude = terrain.z(1)   + terrain.groundClearance;
finalAltitude   = terrain.z(end) + terrain.groundClearance;

% Get slope from initial point to every possible point
terrain.leftSlope  = (terrain.z(2:(end-1)) - terrain.z(1)) ./...
                     (terrain.x(2:(end-1)) - terrain.x(1));
% Get slope from every possible point to final point
terrain.rightSlope = (terrain.z(2:(end-1)) - terrain.z(end)) ./...
                     (terrain.x(2:(end-1)) - terrain.x(end));

% Get slope from initial point to every possible break point
breakPoint.leftSlope  = (breakPoint.z         - initialAltitude) ./...
                        (terrain.x(2:(end-1)) - terrain.x(1));
breakPoint.rightSlope = (breakPoint.z         - finalAltitude) ./...
                        (terrain.x(2:(end-1)) - terrain.x(end));

% figure;
% hold on;
% plot(terrain.x(2:(end-1)),terrain.leftSlope,'b');
% plot(terrain.x(2:(end-1)),terrain.rightSlope,'r');
%
% plot(terrain.x(2:(end-1)),breakPoint.leftSlope,'b:');
% plot(terrain.x(2:(end-1)),breakPoint.rightSlope,'r:');

% Find maximum left break point
for curBreakIndex = 1:(length(terrain.x) - 2)
    if(any(breakPoint.leftSlope(curBreakIndex) < terrain.leftSlope(1:curBreakIndex)))
%         fprintf('Fail left at %f\n',terrain.x(curBreakIndex + 1));
        break;
    end
end
breakPoint.maxLeftIndex = curBreakIndex - 1;
breakPoint.maxLeft = terrain.x(breakPoint.maxLeftIndex);

% Find minimum right break point
for curBreakIndex = (length(terrain.x) - 2):-1:1
    if(any(breakPoint.rightSlope(curBreakIndex) > terrain.rightSlope(curBreakIndex:end)))
%         fprintf('Fail right at %f\n',terrain.x(curBreakIndex + 1));
        break;
    end
end
breakPoint.minRightIndex = curBreakIndex + 1;
breakPoint.minRight = terrain.x(breakPoint.minRightIndex);

% Find minimum left break point
%  The helicopter must be able to maintain a forward velocity of
%  at least 0.4 m/s and cannot climb faster than 4 m/s
maxClimbSlope = 4/0.4;
breakPoint.minLeftIndex =...
    find(breakPoint.leftSlope > maxClimbSlope, 1, 'last') + 1;
if(isempty(breakPoint.minLeftIndex))
    breakPoint.minLeftIndex = 1;
end
breakPoint.minLeft = terrain.x(breakPoint.minLeftIndex);

% Find maximum right break point
%  The helicopter must be able to maintain a forward velocity of
%  at least 0.4 m/s and should not descend faster than 4 m/s
maxDescentSlope = -4/0.4;
breakPoint.maxRightIndex =...
    find(breakPoint.rightSlope < maxDescentSlope, 1, 'first') - 1;
```

```
if (isempty(breakPoint.maxRightIndex))
    breakPoint.maxRightIndex = length(breakPoint.rightSlope);
end
breakPoint.maxRight = terrain.x(breakPoint.maxRightIndex);

fprintf('Left range:  [%9.4f %9.4f]\n',breakPoint.minLeft,  breakPoint.maxLeft);
fprintf('Right range: [%9.4f,%9.4f]\n',breakPoint.minRight, breakPoint.maxRight);

figure;
hold on;
plot(terrain.x,terrain.z,'k-');
plot(terrain.x,(terrain.z + terrain.groundClearance),'k--');
plot([terrain.x(1) terrain.x(end)],[breakPoint.z breakPoint.z],'r--');
axis([terrain.x(1)-10 terrain.x(end)+10 0 breakPoint.z+10]);
set(gca,'DataAspectRatio',[1 1 1]);
% axis image;

for curLeftIndex = breakPoint.minLeftIndex:breakPoint.maxLeftIndex
    plot([terrain.x(1)    terrain.x(curLeftIndex + 1)],...
         [initialAltitude breakPoint.z],'b:');
end

for curRightIndex = breakPoint.minRightIndex:breakPoint.maxRightIndex
    plot([terrain.x(curRightIndex + 1) terrain.x(end)],...
         [breakPoint.z                 finalAltitude],'r:');
end

%% Optimize path
% x = [leftBreak.x  leftBreak.forwardSpeed...
%      rightBreak.x rightBreak.forwardSpeed]
%
% Constraints: ( 1) leftBreak.x  >= breakPoint.minLeft
%              ( 2) leftBreak.x  <= breakPoint.maxLeft
%              ( 3) rightBreak.x >= breakPoint.minRight
%              ( 4) rightBreak.x <= breakPoint.maxRight
%
%              ( 5) leftBreak.forwardSpeed  >= helicopter.minForwardSpeed
%              ( 6) leftBreak.forwardSpeed  <= helicopter.maxForwardSpeed
%              ( 7) rightBreak.forwardSpeed >= helicopter.minForwardSpeed
%              ( 8) rightBreak.forwardSpeed <= helicopter.maxForwardSpeed
%
%              ( 9) leftBreak.x <= rightBreak.x

%     lbx lbf rbx rbf
A = [-1   0   0   0;... %( 1)
      1   0   0   0;... %( 2)
      0   0  -1   0;... %( 3)
      0   0   1   0;... %( 4)
      0  -1   0   0;... %( 5)
      0   1   0   0;... %( 6)
      0   0   0  -1;... %( 7)
      0   0   0   1;... %( 8)
      1   0  -1   0];   %( 9)

b = [-breakPoint.minLeft;...          %( 1)
      breakPoint.maxLeft;...          %( 2)
     -breakPoint.minRight;...         %( 3)
      breakPoint.maxRight;...         %( 4)
     -helicopter.minForwardSpeed;...  %( 5)
      helicopter.maxForwardSpeed;...  %( 6)
     -helicopter.minForwardSpeed;...  %( 7)
      helicopter.maxForwardSpeed;...  %( 8)
      0];                             %( 9)
```

```
% Guess in the middle for initial values
x0 = [ mean([breakPoint.minLeft,breakPoint.maxLeft]);...
       mean([helicopter.minForwardSpeed,helicopter.maxForwardSpeed]);...
       mean([breakPoint.minRight,breakPoint.maxRight]);...
       mean([helicopter.minForwardSpeed,helicopter.maxForwardSpeed])];

% Set up constants
initialPoint.x = terrain.x(1);
initialPoint.z = initialAltitude;
leftBreak.z    = breakPoint.z;
rightBreak.z   = breakPoint.z;
finalPoint.x   = terrain.x(end);
finalPoint.z   = finalAltitude;

% Create optimization options
optOpts = optimset('Algorithm','interior-point');

[x fval] = fmincon(@(x)(energy_optimizer(initialPoint,leftBreak,rightBreak,finalPoint,x)),...
                   x0,A,b,...
                   [],[],[],[],[],optOpts);

plot([initialPoint.x x(1) x(3) finalPoint.x],...
     [initialPoint.z leftBreak.z rightBreak.z finalPoint.z],'ko-',...
     'MarkerSize',10,'LineWidth',2);
```

Listing C.2: energy_optimizer.m

```
function energy = energy_optimizer(initialPoint,leftBreak,rightBreak,finalPoint,x)
% Converts x into appropriate values for get_total_energy and interprets
%  the results for an optimizer
%
% x contains values in meters and meters/s

diagnostics = false;

feetPerMeter = 3.2808399;
if(diagnostics == true)
    disp(x(:).');
end

leftBreak.x              = x(1);
leftBreak.forwardSpeed   = x(2) * feetPerMeter;
leftBreak.time =...
    (leftBreak.x  - initialPoint.x) / leftBreak.forwardSpeed;
leftBreak.verticalSpeed  =...
    (leftBreak.z  - initialPoint.z) / leftBreak.time;

rightBreak.x             = x(3);
rightBreak.forwardSpeed  = x(4) * feetPerMeter;
rightBreak.time =...
    (finalPoint.x - rightBreak.x)   / rightBreak.forwardSpeed;
rightBreak.verticalSpeed =...
    (finalPoint.z - rightBreak.z)   / rightBreak.time;

% Do the math
result = get_total_energy(initialPoint,leftBreak,rightBreak,finalPoint);
energy = result.total.energy;

% Diagnostics
if(diagnostics == true)
    if(exist('result','var'))
        fprintf('l.x: %6.2f l.f: %5.2f l.v: %5.2f l.p: %9.4f\n',...
            leftBreak.x, leftBreak.forwardSpeed/feetPerMeter,...
            leftBreak.verticalSpeed/feetPerMeter, result.firstLeg.power(5));
```

```
        fprintf('r.x: %6.2f r.f: %5.2f r.v: %5.2f r.p: %9.4f\n',...'
            rightBreak.x, rightBreak.forwardSpeed/feetPerMeter,...
            rightBreak.verticalSpeed/feetPerMeter, result.thirdLeg.power(5));
        fprintf('m.p: %9.4f e:%15.0f e.1:%15.0f e.2:%15.0f e.3:%15.0f\n',...
            result.secondLeg.power(5), energy, result.firstLeg.energy,...
            result.secondLeg.energy, result.thirdLeg.energy);
    else
        fprintf('l.x: %6.2f l.f: %5.2f l.v: %4.2f r.x: %6.2f r.f: %5.2f r.v: %4.2f e:%15.0f\n',...
            leftBreak.x, leftBreak.forwardSpeed/feetPerMeter,...
            leftBreak.verticalSpeed/feetPerMeter,...
            rightBreak.x, rightBreak.forwardSpeed/feetPerMeter,...
            rightBreak.verticalSpeed/feetPerMeter,...
            energy);
    end
end
```

Listing C.3: get_total_energy.m

```
function trip = get_total_energy(initialPoint,leftBreak,rightBreak,finalPoint, heliStruct)
% Calculates total power required (in ft-lbs) to travel from initialPoint
%  to leftBreak to rightBreak to finalPoint using the horizontal and
%  vertical speeds specified in leftBreak and rightBreak
%
% The helicopter travels at (leftBreak.forwardSpeed,leftBreak.verticalSpeed)
%  from (initialPoint.x,initialPoint.z) to (leftBreak.x,leftBreak.z)
%
% It travels at 4.5 m/s from (leftBreak.x,leftBreak.z) to
%  (rightBreak.x, rightBreak.z)
%
% It travels at (rightBreak.forwardSpeed,rightBreak.verticalSpeed) from
%  (rightBreak.x,rightBreak.x) to (finalPoint.x,finalPoint.z)
%
% It is assumed that the speeds are coincident; that is, that
%  (leftBreak.x - initialPoint.x) / leftBreak.verticalSpeed ==...
%  (leftBreak.z - initialPoint.z) / leftBreak.forwardSpeed)
%  and the same is true for the rightPoint and finalPoint.
%
% It is further assumed that leftBreak.z == rightBreak.z and that
%  initialPoint.x < leftBreak.x < rightBreak.x < finalPoint.x
%
% The helicopter used is the one specified in the heliStruct argument
%  There are reasonable defaults, however

if(~exist('heliStruct','var'))
    heliStruct.length = 9.3373; % ft
    heliStruct.width  = 1.7060; % ft
    heliStruct.height = 2.3294; % ft

    heliStruct.mass = 127.868; % 58 kg in lbs

    heliStruct.bladeRadius = 5.1099; % ft
    heliStruct.bladeArea = 2.3190; % ft^2
    heliStruct.numBlades = 2;
    heliStruct.rotorMaskArea = 6.8597; % ft^2
    heliStruct.rotorAngularVelocity = 89.0118; % 850 rpm in rad/s

    % Initial values for variable values
    heliStruct.forwardSpeed = 13.1200; % 4 m/s in ft/s
    heliStruct.verticalSpeed   = 0;

    heliStruct.temperature  = 70; % degrees F
    heliStruct.pressure     = 14.504; % psi, 1 atm
end
```

```
% Get energy required for initial leg
heliStruct.forwardSpeed  = leftBreak.forwardSpeed;
heliStruct.verticalSpeed = leftBreak.verticalSpeed;
firstLeg.power = get_required_power(heliStruct);

firstLeg.time = (leftBreak.x - initialPoint.x) / leftBreak.forwardSpeed;

firstLeg.energy = firstLeg.power(5) * firstLeg.time;

% Get energy required for second leg
heliStruct.forwardSpeed = 14.7638; % 4.5 m/s in ft/s
heliStruct.verticalSpeed = 0;
secondLeg.power = get_required_power(heliStruct);

secondLeg.time = (rightBreak.x - leftBreak.x) / 4.5;

secondLeg.energy = secondLeg.power(5) * secondLeg.time;

% Get energy required for last leg
heliStruct.forwardSpeed  = rightBreak.forwardSpeed;
heliStruct.verticalSpeed = rightBreak.verticalSpeed;
thirdLeg.power = get_required_power(heliStruct);

thirdLeg.time = (finalPoint.x - rightBreak.x) / rightBreak.forwardSpeed;

thirdLeg.energy = thirdLeg.power(5) * thirdLeg.time;

% Get totals
total.time   = firstLeg.time   + secondLeg.time   + thirdLeg.time;
total.energy = firstLeg.energy + secondLeg.energy + thirdLeg.energy;

trip.firstLeg = firstLeg;
trip.secondLeg = secondLeg;
trip.thirdLeg = thirdLeg;
trip.total = total;
```

Listing C.4: get_required_power.m

```
function power = get_required_power(heliStruct)

if(~exist('heliStruct','var'))
    heliStruct.length = 9.3373; % ft
    heliStruct.width  = 1.7060; % ft
    heliStruct.height = 2.3294; % ft

    heliStruct.weight = 127.868; % 58 kg in lbf

    heliStruct.bladeRadius = 5.1099; % ft
    heliStruct.bladeArea = 2.3190; % ft^2
    heliStruct.numBlades = 2;
    heliStruct.rotorMaskArea = 6.8597; % ft^2
    heliStruct.rotorAngularVelocity = 89.0118; % 850 rpm in rad/s

    heliStruct.forwardSpeed = 13.1200; % 4 m/s in ft/s
    heliStruct.verticalSpeed   = 0;

    heliStruct.temperature  = 32; % degrees F, STP (0 C)
    heliStruct.temperature  = 70; % degress F
    heliStruct.pressure     = 14.504; % psi,   STP (101325 Pa)
end

gc = 32.174; %lbm*ft/(lbf*s^2)

% p/(R*T), where R = 53.3533 ft*lbf/(lbm*R)
```

```
density = 144 * heliStruct.pressure / (53.3533 * heliStruct.temperature); % in lbm/ft^3

%% Induced Power
thrustRequired = (1 + 0.3*heliStruct.rotorMaskArea / (pi*heliStruct.bladeRadius^2)) *...
                 heliStruct.weight; % Prouty, p. 7 % in lbf
inducedPower = thrustRequired^2 / (2 * (density/gc) * pi*heliStruct.bladeRadius^2 * heliStruct.forwardSpeed);

%% Parasite Power
referenceSpeed = heliStruct.forwardSpeed;

% Drag in forward flight (from Prouty, pp.306-308)
% Basic fuselage
fuselageFrontalArea = 0.189983; %m^2
finenessRatio = heliStruct.length / max(heliStruct.width,heliStruct.height);
cdf = 0.08; % based on fineness ratio, kind of (Prouty, p. 294)
fuselageFlatPlate  = fuselageFrontalArea * cdf; % m^2

% Main rotor hub and shaft
mainHubFrontalArea   = 1.5833 * 0.2083; % ft^2
mainShaftFrontalArea = 0.1183 * 0.4000; % ft^2
mainShaftDiameter    = 0.1183; % ft
cdmh = 0.15; % based on Prouty, p. 298, at 0% RPM
mainDragRatio = 1.00/0.95; % based on Prouty, p. 299, 100% RPM, 0 shaft angle
cdmh = cdmh * mainDragRatio;
mainHubFlatPlate = mainHubFrontalArea * cdmh;
shaftReynoldsNumber = 6400 * referenceSpeed * mainShaftDiameter;
cdms = 1.2;
mainShaftFlatPlate = mainShaftFrontalArea * cdms;
mainRotorAssemblyFlatPlate = mainHubFlatPlate + mainShaftFlatPlate;
hubGapToPylonWidthRatio = 4.65 / 5.675; % in/in
hubPylonInterferenceDragFactor = 0; % Prouty, p. 301
mainRotorAssemblyFlatPlate = (1 + hubPylonInterferenceDragFactor) * mainRotorAssemblyFlatPlate;

% Tail rotor hub and shaft
tailRotorAssemblyFrontalArea = 0.1642 * 0.5083; % ft^2
cdth = 0.04; % Prouty, p. 298, at 0% RPM
tailDragRatio = 1.00/0.95; % based on Prouty, p. 299, 100% RPM, 0 shaft angle
cdth = tailDragRatio * cdth;
tailFlatPlate = tailRotorAssemblyFrontalArea * cdth;

% Main landing gear
mlgCrossbarFrontalArea = 2.2083 * 0.0833; % ft^2
mlgSkidsFrontalArea    = 0.0667 * 0.4375; % ft^2
mlgArmsFrontalArea     = 0.0583 * 1.0833; % ft^2
mlgFrontalArea = mlgCrossbarFrontalArea + mlgSkidsFrontalArea + mlgArmsFrontalArea;
cdmlg = 0.4; % Prouty, p. 303
mlgFlatPlate = mlgFrontalArea * cdmlg;

% Rotor-fuselage interference drag
deltacd = 0.018; % Prouty, p. 302
interferenceFlatPlate = fuselageFrontalArea * deltacd;

% Miscellaneous drag
stereoBoomFrontalArea = 5.125 * 0.2583; % ft^2
stereoBoomReynoldsNumber = 6400 * referenceSpeed * 0.2583; % based on Prouty, p. 288
cdsb = 1.2; % drag coefficient of the stereo boom
            % based on Munson, Young, Okiishi p. 526
stereoBoomFlatPlate = stereoBoomFrontalArea * cdsb;

radPodFrontalArea = 1.8333 * 0.6875; % ft^2
cdrp = 0.80; % drag coeffficient of the rad pod
             % based on Munson, Young, Okiishi p. 537
radPodFlatPlate = radPodFrontalArea * cdrp;

```

```
% Totals
totalFlatPlate = fuselageFlatPlate + mainRotorAssemblyFlatPlate + tailFlatPlate +...
                 mlgFlatPlate + interferenceFlatPlate + stereoBoomFlatPlate +...
                 radPodFlatPlate;

parasitePower  = 0.5 * (density/gc) * heliStruct.forwardSpeed^3 * totalFlatPlate;

%% Profile Power
cdb = 0.0083; % blade cd, Prouty p. 23
rotorAdvanceRatio = heliStruct.forwardSpeed /...
                    (heliStruct.rotorAngularVelocity * heliStruct.bladeRadius);
profilePower = (density/gc) * heliStruct.bladeArea * heliStruct.numBlades *...
               (heliStruct.rotorAngularVelocity * heliStruct.bladeRadius)^3 *...
               (cdb / 8) * (1 + 3 * rotorAdvanceRatio^2); % Prouty p. 133



%% Climb Power
% climbPower = (heliStruct.weight * 32.2 * heliStruct.verticalSpeed) ./...
%               ((density/gc) * (pi*heliStruct.bladeRadius^2) *...
%                (heliStruct.rotorAngularVelocity * heliStruct.bladeRadius).^3);
climbPower = heliStruct.weight * heliStruct.verticalSpeed;

%% Total power
totalPower = inducedPower + profilePower + parasitePower + climbPower;

power = [inducedPower profilePower parasitePower climbPower totalPower];

% %% Spit it out
% fprintf('Induced Power:  %6.2f\n',inducedPower);
% fprintf('Profile Power:  %6.2f\n',profilePower);
% fprintf('Parasite Power: %6.2f\n',parasitePower);
% fprintf('Climb Power:    %6.2f\n',climbPower);
% fprintf('Total Power:    %6.2f\n',totalPower);
```

Listing C.5: generate_random_terrain.m

```
% Generates random 2-dimensional (x-z) "terrain" maps

function terrainMap = generate_random_terrain(maxX,resolution,smallestFeature)

if(~exist('maxX','var'))
    maxX = 500;
end

if(~exist('resolution','var'))
    resolution = 1;
end

if(~exist('smallestFeature','var'))
    smallestFeature = 1;
end

grid.x = 1:resolution:maxX;

baseArray = rand(length(grid.x),1);

% Smooth array
terrainMap = zeros(size(baseArray));
largestFeature = 2^round(log2(maxX)) / 2;
initialSize = largestFeature;

figure;
hold on;
```

```
while(largestFeature >= smallestFeature)
    newTerrain = simple_smooth_array(baseArray,largestFeature) * largestFeature;
    terrainMap = terrainMap + newTerrain;
    largestFeature = largestFeature / 2;

    plot(grid.x,terrainMap / initialSize,...
        'Color',[max(log2(largestFeature)/log2(initialSize),0),...
                 max(log2(largestFeature)/log2(initialSize),0),...
                 1]);
end

terrainMap = terrainMap / initialSize;
```

Listing C.6: simple_smooth_array.m

```
function smoothArray = simple_smooth_array(array, ratio)

arraySize = size(array);
array = array(:);

if(~exist('ratio','var'))
    ratio = 2;
end

indices = (1:length(array)) ./ ratio;

intIndices = ceil(indices);
fracIndices = indices - intIndices + 1;

smoothArray = array(intIndices) .* fracIndices(:) +...
              array(mod(intIndices,length(array)) + 1) .* (1 - fracIndices(:));

smoothArray = reshape(smoothArray,arraySize);
```