Discriminating Between Splitting and Crossing Targets: A Radar Tracking Problem

A Major Qualifying Project Report

submitted to the Faculty of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Bachelor or Science

by

Christopher J. Cleary

Sara I. Durán

Eric A. Scheid

Date: 10/10/2007

Sponsor: MIT Lincoln Laboratory
Supervisor: Dr. Stephen Weiner

WPI Advisors: Professor Kevin Clements
Professor Germano Iannacchione

Authorship

This project was certainly a collaborative effort of the team members, however certain elements of it drew more heavily on our respective disciplines (Physics for Mr. Scheid, Electrical and Computer Engineering for Mr. Cleary and Ms. Duran). This section attempts to outline what parts of the project are more specifically related to one concentration or the other, as well as the primary contributor to each section of the report.

The majority of the principles described in the background section of our report are physical in nature; in particular, section 2.1 Radar Basics and section 2.4 Range-Time Intensity Plots rely heavily on the physics of radar and propagating signals. Physical principles also guided all the code that was used to simulate the trajectory and radar profiles of the objects viewed, such as in section 2.2. The quantification of our heuristics in section 3.1.2 and Appendix A were grounded in physical and mathematical concepts. Lastly, section 4.2.2 outlined the physical dependence of the apparent angular and translational rates of objects on the viewing geometry of the radar.

This project had a heavy simulation component that relied on Electrical and Computer Engineering principles. Specifically, the random generation of RTIs required multiple upgrades to the simulation software provided by Lincoln Laboratory staff. The specifics of these upgrades are outlined in section 3.2.2 and the code can be found in Appendix E. Organizing and presenting the data for sections 4.1 and 4.3 required writing additional MATLAB scripts, an example of which can be found in Appendix F.

Table of Contents

Table of Figures

Table of Tables

Abstract

It can be difficult to discern between crossing and splitting targets when looking at radar tracks. Radar tracking problems such as this are important to modern ballistic missile defense, but parameters such as the radar bandwidth, visibility time, and the relative speed of the objects can obscure interpretation. A human decision-making model was developed to aid in interpretation, and 3001 simulated radar tracks were analyzed at MIT-Lincoln Laboratory using this algorithm. Operating curves were created to describe the model's performance.

Executive Summary

During World War II, Germany launched the first ballistic missile - the V-2, or *Vergeltungswaffe Zwei* [1] - which struck British soil in September of 1944. Shooting down a V-2 after it was in flight was impossible at the time[2], making the investigation of missile defense imperative. The threat has evolved from these ponderous early missiles, to the massive arsenal of the Soviet Union, to the modern danger of rogue states using small numbers of intercontinental ballistic missiles. Presently, one particular difficulty is that of tracking objects of interest. Tracking an object allows the radar operator to see what path it has taken and predict where it will be in the future – vital to the defense's ability to engage the targets. As the threat complex changes from the initial ballistic missiles to the final cloud of reentry vehicles, decoys, and debris, the defense can form a better idea of which targets are dangerous by linking together successive tracks[3].

Background and Purpose

To perform any significant analysis of a radar tracking problem, one must first understand the basic physics behind a radar system and learn how to read radar tracks. A radar works by sending out a radio signal and counting the time elapsed until it reflects back. The range to the object can be easily calculated due to the constant speed of electromagnetic waves. This simple process is then repeated to gain an understanding of the object's time evolving behavior.

The radar tracks show how the objects in question behave over time, and are aptly called Range-Time Intensity plots. Figure 1 is, like the rest of the radar images in this report, a simulated RTI. It shows an example plot along with its corresponding physical scenario on the left for edification purposes (RLOS denotes the radar line of sight).

---

[1] Kaplan, Dr. Laurence M. "Missile Defense: The First Sixty Years". Missile Defense Agency. 27 September 2006. Accessed September 10, 2007. <http://www.mda.mil/mdalink/pdf/first60.pdf>. Page 1.

[2] Werrel, Kenneth P. "Hitting a Bullet with a Bullet: A History of Ballistic Missile Defense." College of Aerospace Doctrine, Research and Education. Air University. Research Paper 2000-02. (2000). Page 2.

[3] Weiner, Stephen D. Private conversations.

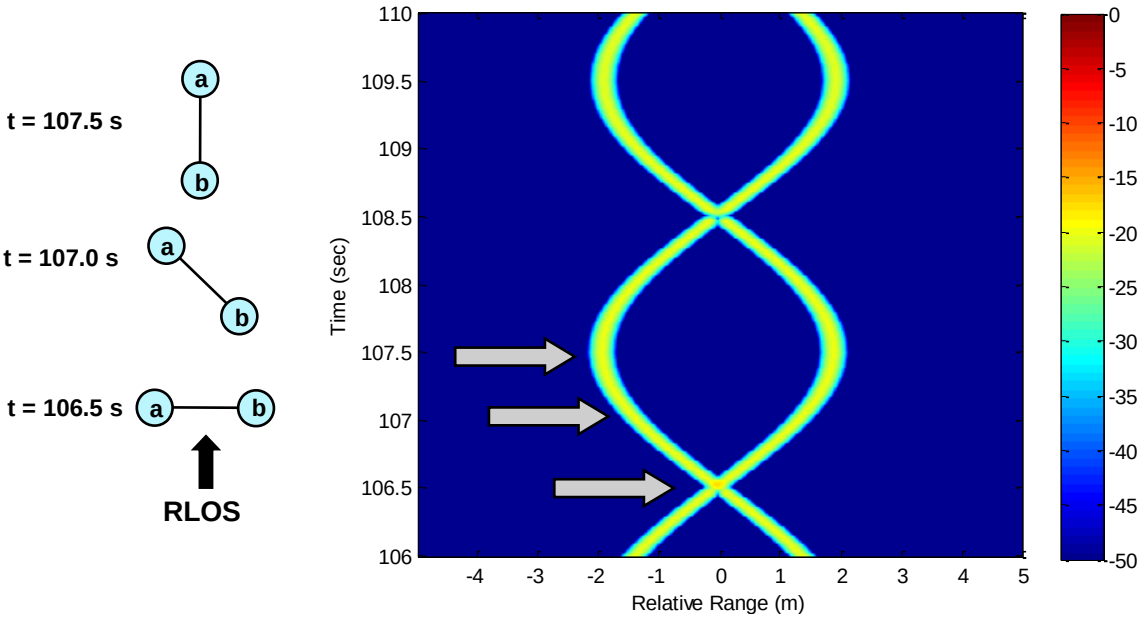

Figure 1: Rotating Dumbbell Diagram and RTI

Many of the objects tracked using these types of plots have angular velocities. In Figure 1 it can be seen that the relative ranges of the tracks from the two scatterers oscillate over time; this is a graphical manifestation of the angular motion of the dumbbell. The periodic nature of the plot is a direct result of measuring the range relative to the object's center of mass.

Radar is typically very good at measuring the range to an object, but a radar's angular resolution is comparatively much poorer. As a result, objects that appear to be crossing or colliding on a radar track may in actuality be quite separate. One problematic consequence of this poor angular resolution is the prevalence of crossing events seen on radar tracks. It is of paramount importance to be able to discern between crossing events and splitting events so that accurate tracks can be maintained. Although seemingly trivial, the usually stark differences between a split and a cross can be obscured by many parameters. Most important to situational clarity – and of particular significance to this project – are the bandwidth of the radar, the visibility time before the event, and the intersection angle of the two objects in question as measured on the RTI.

Another aspect of this project involved analyzing the role of the human radar operator in interpreting RTIs. For many of the tasks crucial to ballistic missile defense humans are excluded entirely, as they lack the reaction time and multi-tasking ability that is needed for many operations. On the other hand, humans can provide flexibility in a way that

machines cannot. For example, computers can have great difficulty correctly interpreting crossing targets in noise, even at high signal to noise ratios, whereas humans have little difficulty[4]. Our project tackled a particular radar tracking problem - discerning between splitting and crossing targets - from the human observer perspective.

Methodology and Scope

This project had three main objectives: to develop a set of heuristics that allow us to decide whether an event is a split or a cross, to develop a human decision-making model that codified these heuristics, and to produce operating curves that exhibited the effectiveness of the human decision-making model. To make the project manageable for a seven week assignment, we put some constraints on the problem to ease the analysis. There are many parameters that affect situational clarity, but we decided to focus on only the bandwidth, visibility time, and intersection angle. Furthermore, we only considered binary interactions, and modeled the objects involved as identical reentry vehicles. To simplify the statistical analysis, we constructed every event so that it had to be either a split or a cross (as opposed to possible situations where both or neither occur).

To generate sample radar tracks for analysis, we wrote a MATLAB program that randomly generates splitting and crossing events between two objects. The program randomly chooses one of three distinct bandwidths, and then chooses either a splitting template or a crossing template. The intersection angle displayed on the RTI – which is also analogous to the relative speed of the objects - is randomly chosen from the Gaussian weighted distribution that describes the respective scenario. Finally, to keep the number of trivial cases low, we limited the visibility time prior to the event to a randomly determined value between -1.5 and 1.0 seconds.

We then performed some exploratory exercises with these randomly generated radar tracks. Each of us examined a large set of RTIs, labeled each event as a split or a cross, and wrote down the reasoning behind our decision. We then compared our decisions with the actual events logged in a record file produced by the MATLAB program, and took note of our accuracy. The exercises allowed us to verify that the process was random enough that we

[4] Weiner, Stephen D. Private conversations.

were not recognizing patterns, and that there was an adequate ratio of edge cases to obvious ones. Additionally, they provided further insight on how each of the observables (bandwidth, intersection angle, and visibility time) affected our ability to discriminate between splitting and crossing targets.

After performing the exploratory exercises we narrowed down our reason pool to three main rules: the time before rule, the width rule, and the intersection angle rule. The time before rule delineates the minimum time that one needs to be able to detect two distinct tracks if given the width of a track and the angle of intersection. The width rule handles cases where the event can be seen, but the time before rule does not apply. It suggests that if the track is significantly wider than it should be at $t = 0$, the event is probably a cross. If all else fails, the intersection angle rule handles all remaining cases by postulating that higher intersection angles correspond to crosses while lower ones correspond to splits.

This led directly to the development of a human decision-making model. We ordered the rules logically, and derived their quantitative analogues so that they were less subjective. Once we had our standardized human decision-making model, we began applying it to radar tracks in earnest to provide a substantial sample size for our final analysis. We generated 3001 radar tracks, which gave us many varying combinations of intersection angle, bandwidth, and time before the event.

Using our decision-making model, we examined the RTIs and documented our answers in an Excel sheet. We also recorded the specific route used by the human decision-making model to reach the decision for each RTI. Excel then made it easy to calculate the overall error rate of our model, and furthermore, the success rate of each individual heuristic. The next step was to condense this deluge of data into something readable and presentable. We decided to construct operating curves that show the overall effectiveness of our human decision-making model.

Results and Discussion

Once we had our final set of data, we wrote a program that condensed it into easy to read operating curves, as seen in Figure 2.

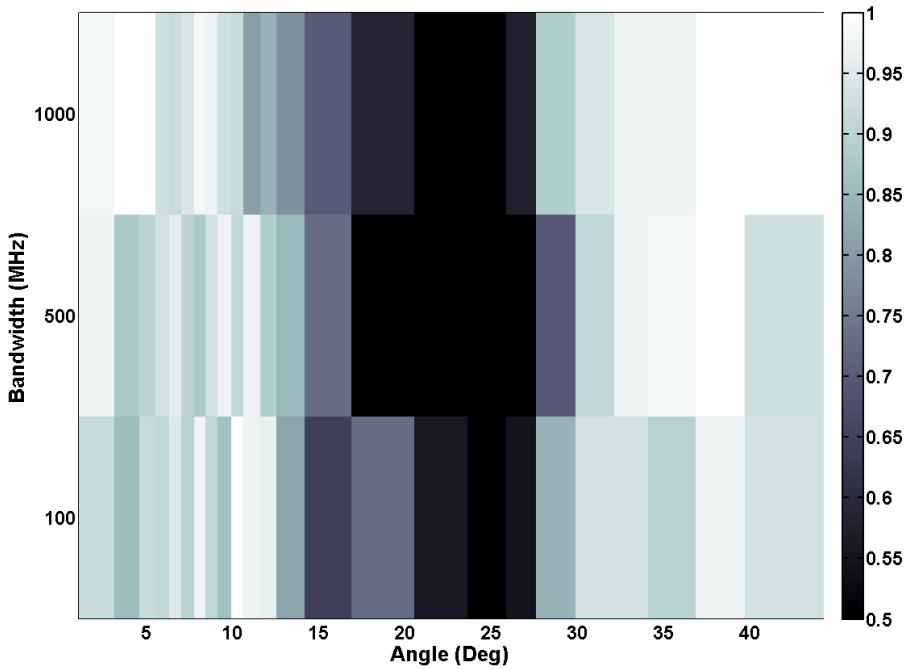

Figure 2: System Performance Given Bandwidth and Intersection Angle

It can be seen that the operating curve displays three dimensions of data: intersection angle on the x-axis, bandwidth on the y-axis, and probability of correct identification on the color-axis. We used three distinct bandwidths in our RTI generation software, hence the 3 separate rows. To enable calculations of realistic probabilities, we quantized the x-axis; quantization allowed us to combine the data from different angles in the same neighborhood, and calculate a probability of correct identification for that specific neighborhood of angles. This operating curve example related bandwidth to intersection angle, but we also created a curve relating the time before the event to intersection angle. In this case, we quantized the time axis for the same reasons as the intersection angle axis.

The operating curve shown in Figure 2 demonstrates probability of correct interpretation at various levels of intersection angle and bandwidth. Performance increases with bandwidth and as the angles move away from the threshold, giving near perfect performance at 1000 MHz bandwidth when the angle is more than 10° above or below the threshold of 27.9°.

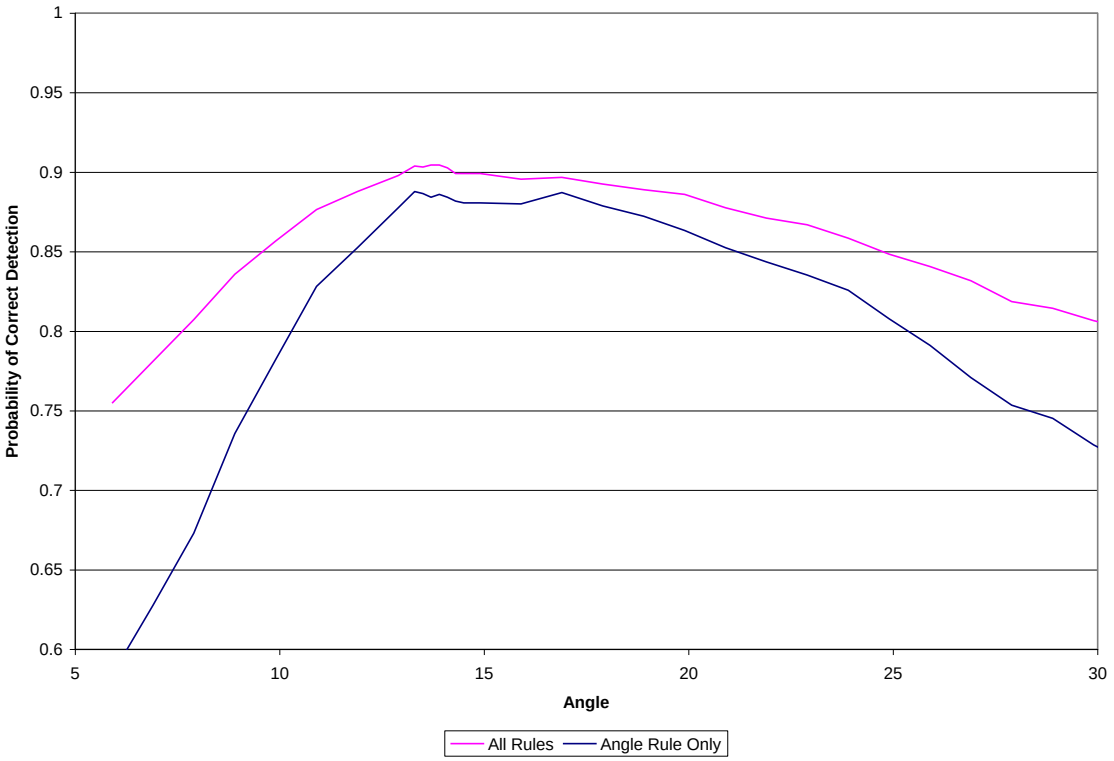

Figure 3: Angle versus Probability of Correct Identification for Original System

Additional analysis of the dataset, presented in Figure 3, showed that the threshold calculated from the distribution of the variables used to generate the RTIs was not the ideal threshold. The cause of this discrepancy was the radar viewing geometry, which caused the apparent angles and velocities to be smaller than their true values. Using the existing dataset, the group was able to produce statistics to describing the data while taking into account the viewing geometry used. This resulted in a new calculated threshold, and led to a great increase in performance.

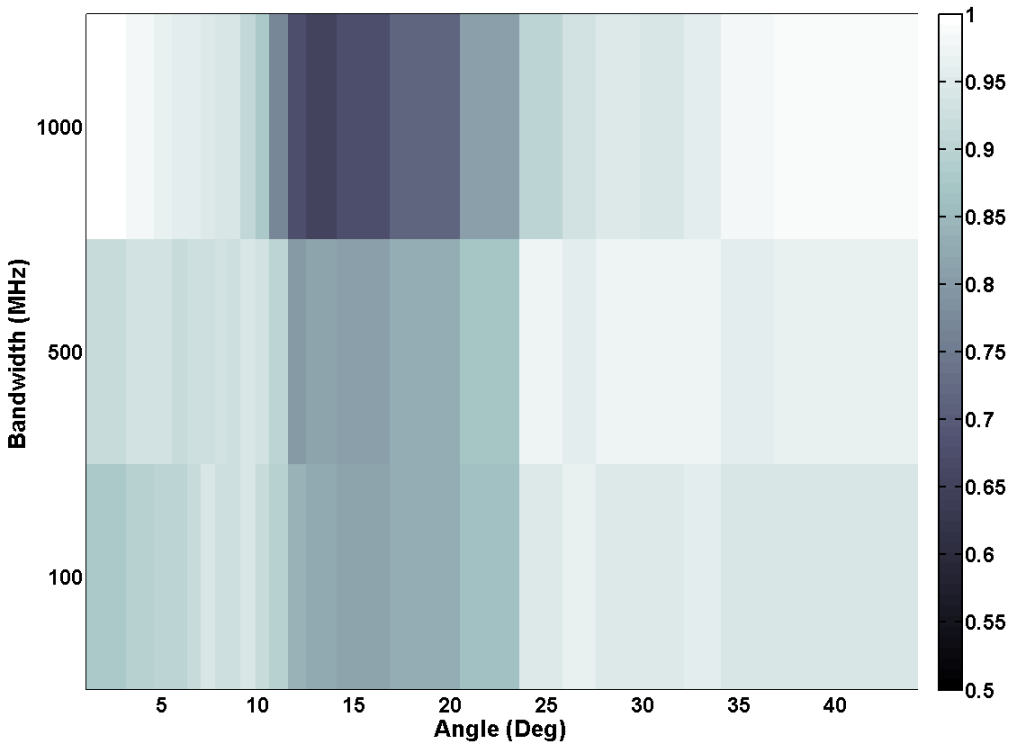

Figure 4: System Performance Given Bandwidth and Intersection Angle with Revised Threshold

Figure 4 demonstrates system performance using the new threshold. The minimum level of performance is much higher than that observed in Figure 4. The low-performance band is shifted to be about the new threshold of 13.9°, and is both narrower and shallower than that observed with the old threshold, occupying a 12° window and reaching a minimum value of 60% correct detections. For angles of less than 10°, the performance improves as bandwidth increases, although this seems to be reversed for angles near the new threshold. For larger angles, performance is excellent for all bandwidths.

Conclusions and Future Recommendations

The project team found that our human decision-making model performed well, exhibiting a 15% error rate while examining mostly difficult edge cases. We succeeded in both modeling and improving system performance using simulated data. The conclusion of our project leaves open several possibilities for further research; for example, research could be conducted using parameter values derived from actual test data, or into developing an entirely automated system

1. Introduction

During World War II, Germany launched the first ballistic missile, the V-2 or *Vergeltungswaffe Zwei* (Kaplan 1), which struck British soil in September of 1944. Shooting down a V-2 after it was in flight was impossible at the time (Werrel 2), making the investigation of missile defense imperative. This effort can be divided into two phases, according to the method used to intercept a threat: the nuclear warhead era and the non-nuclear era. Destroying incoming threats by detonating a nuclear warhead in their vicinity was the defense modus operandi from 1946 until 1983 (Weiner), when the Reagan administration started the Strategic Defense Initiative (SDI) (United States Department of Defense). The objective of this program necessitated the development of non-nuclear interceptors. The goal was to have the interceptors physically impact the incoming missiles, that is, to "hit a bullet with a bullet".

Obtaining and interpreting information is one of the greatest difficulties in ballistic missile defense. The defender may have to track thousands of objects, hundreds of kilometers away, with only a handful of sensors, and come to a decision on what objects are threats in minutes. One particular difficulty is that of tracking objects of interest. Tracking an object allows the radar operator to see what path an object has taken and predict where it will be in the future – vital to the defense's ability to engage the targets. As the threat complex changes from the initial ballistic missiles to the final cloud of reentry vehicles, decoys, and debris, the defense can form a better idea of which targets are dangerous by linking together successive tracks (Weiner).

The defense's ability to track is limited in several ways. Many sensors may only be able to maintain a certain number of tracks at once. This problem can be exacerbated by the offense's use of decoys (Weiner and Rocklin, 73-74). The quality of the track is also limited by how long the defender has observed the object and how often external objects interfere with the track (Weiner). Performance limitations in the

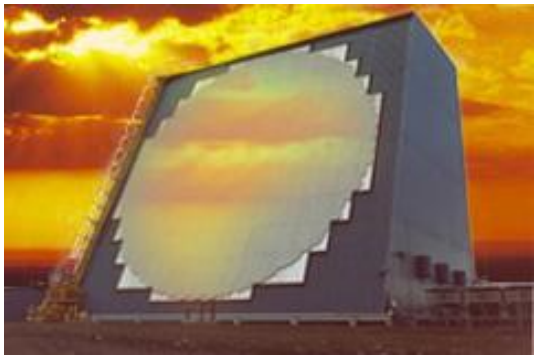

Figure 1-1: Cobra Dane Radar

sensors available will also affect the quality of the tracks by limiting the information available. For example, Cobra Dane (seen in Figure 1-1) is one of the key sensors in the National Missile Defense system, and is an L-band radar operating at 200 MHz bandwidth (Amoozegar 6) that produces narrowband images which cannot provide detailed information on the objects tracked (Raytheon).

To perform any significant analysis of a radar tracking problem, one must first understand the basic physics behind a radar system and learn how to read radar tracks. A radar works by sending out a radio signal and counting the time elapsed until it reflects back. The distance to the object can then be easily calculated. This simple process is then repeated again and again to gain an understanding of the object's time evolving behavior. The radar tracks we analyze show how the objects in question behave over time; this concept can be seen in Figure 1-2, along with an illustration of the physical situation on the left (RLOS denotes the radar line of sight).

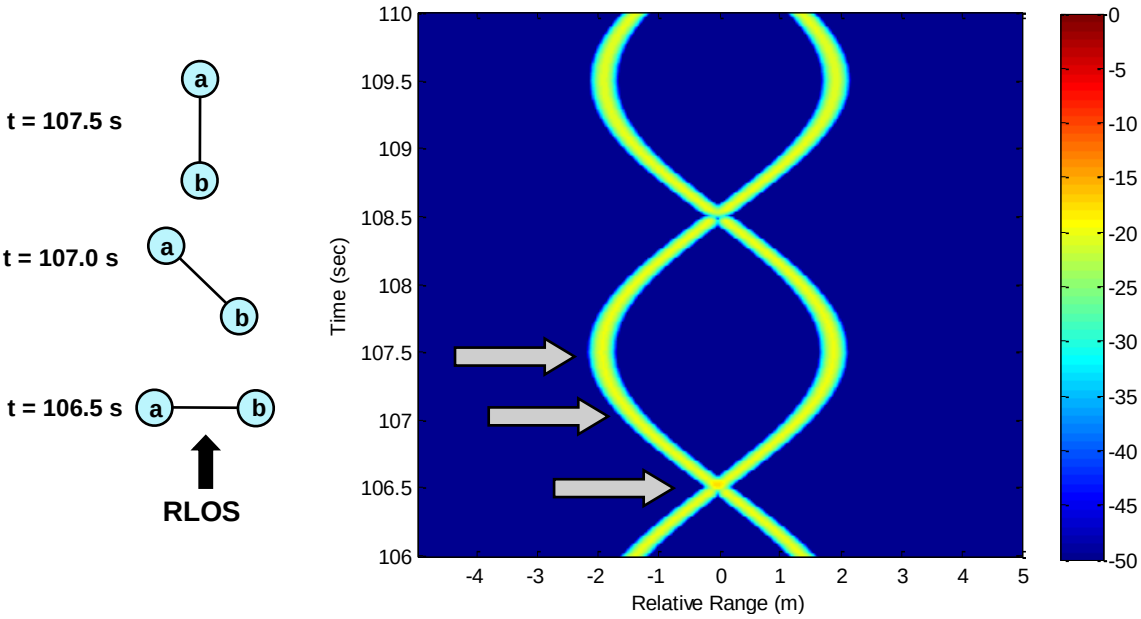

Figure 1-2: Rotating Dumbbell Diagram and RTI

This radar track is of a single object with two highly reflective returns. The range (relative to the center of the object) of each return as viewed by the radar is measured along the x-axis. Time and power are measured along the y-axis and the color axis respectively. It can be seen that the motion of the object is periodic; this is a common feature of radar tracks since most objects have a constant angular velocity.

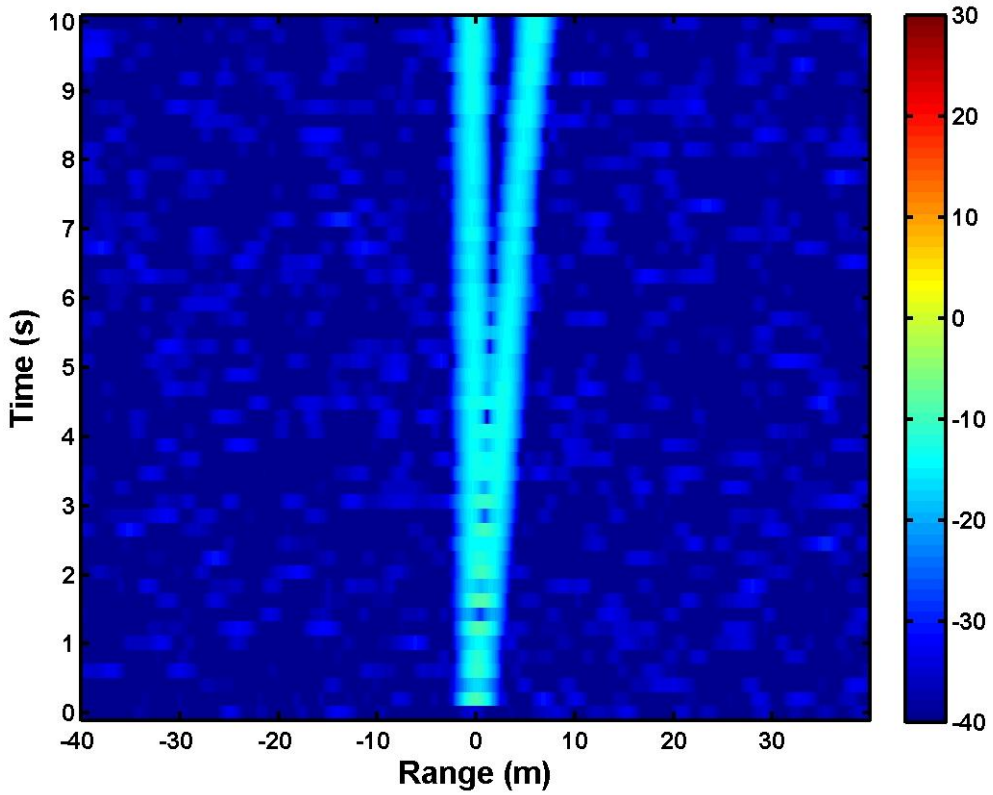

Figure 1-3: Two Crossing Targets

The radar track depicted in Figure 1-3 is quite clear, and there is little doubt as to what is happening to the object in question. However, there are many parameters that affect the clarity of radar tracks. The most important of these for our research are the following: bandwidth, relative velocity of the objects, and the visibility time before and after multi-object events (see Figure 1-3). In this figure, the tracks from two objects are intersecting, but it is unclear whether they have split from a common object or are merely crossing. The low bandwidth and small visibility time before the event obscure the true nature of the objects' behaviors. It is important to correctly identify an event as a split or a cross, and to make decisions of this nature one must understand the observables and known parameters associated with each event.

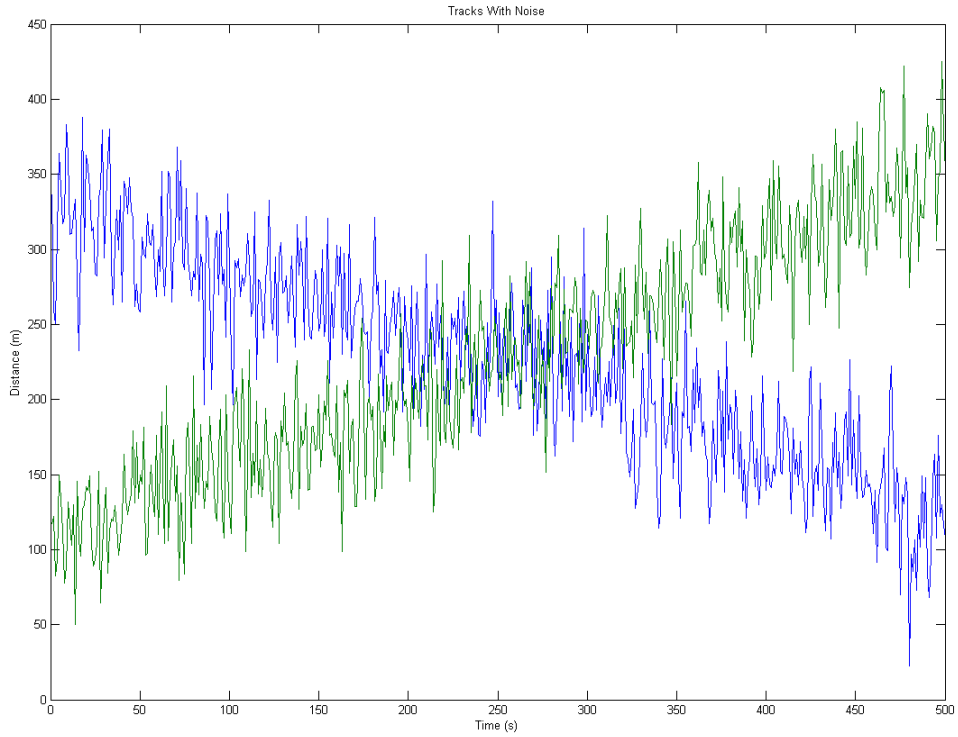

Figure 1-4: Crossing Targets with Noise

The role that humans should play in a ballistic missile defense system is a topic of great importance to system designers, but it is not one which has been thoroughly explored. For many of the tasks vital to BMD humans are excluded entirely, as they lack the reaction time and multi-tasking ability that is vital for many operations. However, humans bring many important capabilities into a BMD system. Humans provide flexibility in a way that machines cannot; they have the ability to adapt rapidly to changing or unexpected circumstances, and an intuition which can help guide decisions made on incomplete or even insufficient information (Hawley 7). For example, computers can have great difficulty correctly interpreting crossing targets in noise, even at high signal to noise ratios, while humans have little difficulty (Spence). An example of a situation of this nature can be seen in Figure 1-4. Further study is required, however, to identify other areas of BMD in which humans can perform well, and to provide detailed models of this performance.

Our project focuses on a very specific problem: the discrimination between splitting and crossing targets. Our goal is to develop a series of operating curves that model the discrimination performance and the probability of correct identification. In order to arrive at

the operating curves we must first translate the mental process humans go through to identify these events into a set of heuristics. These will then be organized to form a human decision-making model that will be applied to a large sample of randomly-generated radar tracking images. We will record the accuracy obtained using the model; that is, to determine the probability of correct identification with respect to different parameters such as bandwidth, time before and after the event, relative speed. This information will be presented in the form of quantized operating curves (see Figure 1-5). Although our problem is a very concrete one, we hope that our methodology can be extrapolated to other specific problems. Each small problem solved is a step along the way towards a successful ballistic missile defense system. The ballistic missile defense problem is one that cannot be solved without working "from the bottom up".

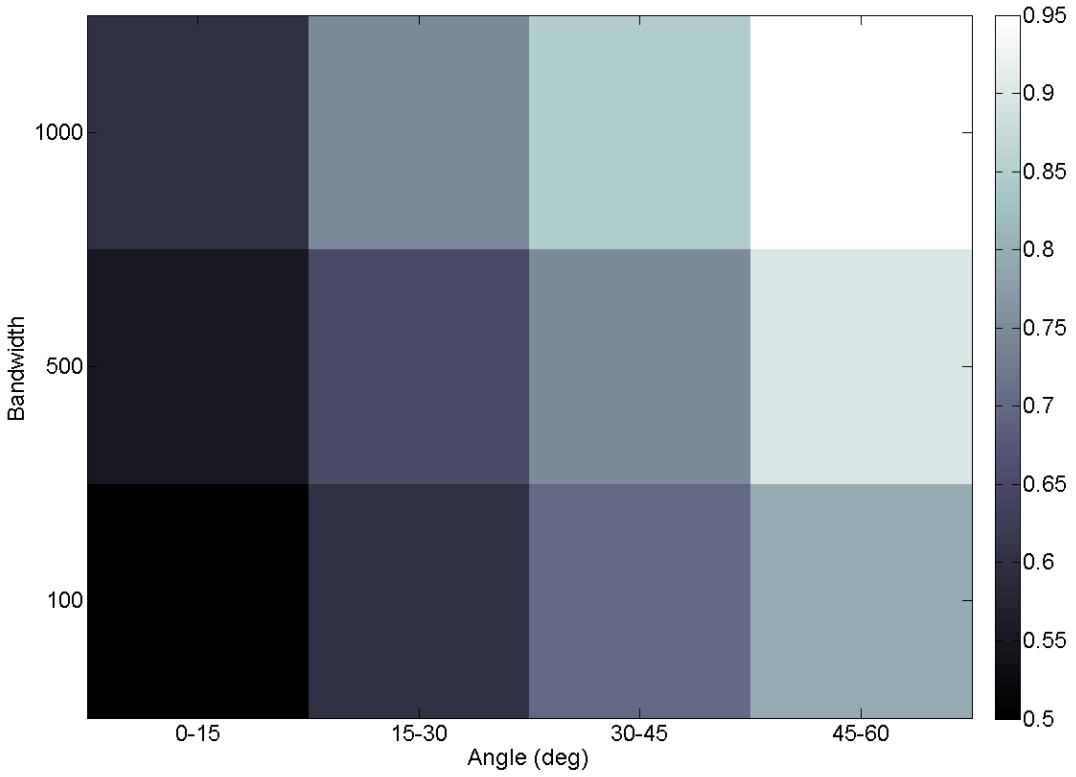

Figure 1-5: Sample operating curve

2. Background

It is important when interpreting radar measurements to discern where different objects originate from. Our project deals with a specific problem of this nature: analyzing the tracks of two objects that are close to each other. This task is complicated by the prevalence of imperfect tracks – for example, a track that starts only a second before an event, or is obscured by noise. Our project seeks to develop heuristics that humans can use when analyzing these types of problems.

While most of the air defense effort is automated, this does not eliminate the need for a human element in decision making.

The utility of automating the engagement process was dramatically demonstrated with the success of the Patriot system in countering the Iraqi tactical ballistic missile (TBM) threat during Operation Desert Storm and most recently during Operation Iraqi Freedom (OIF). In both Gulf wars, TBMs were successfully engaged by Patriot employed in a fully automatic, operator-monitored mode. The down side of these successes was an unacceptable number of fratricidal engagements attributable to track misclassification problems, particularly during OIF. (Hawley, Mares and Giammanco 2)

Our project tackles a particular radar tracking problem - discerning between splitting and crossing targets- from the human observer perspective. Our hope is that our decision-making model will be able to tackle situations that may be challenging for a computer algorithm to correctly interpret (Spence). In order to fully understand the problem at hand one must first understand the principles of radar tracking.

Radar Basics

Radar is an acronym that stands for Radio Detection and Ranging. Radar is used to determine the presence of an object, its distance from the radar and its speed. The basic concept has been around for over a century. Christian Hulsmeyer saw a practical implication in Heinrich Hertz's work in the late 1800s, and built a rudimentary radar system in 1904 that could detect ships hidden by fog. However, it was not until the genesis of air warfare years later that radar became the widely researched application that is today (Skolnik 14-15).

The principles of radar imaging are fairly simple to explain, especially with the usual scenario where the transmitter and the receiver share the same antenna. The transmitter will send out a radio wave pulse toward the target in question, the pulse will then reflect off of the object, and return at a lesser power back to the receiver. The range to an object is simple to formulize since radio waves are a form of electromagnetic radiation, and travel at the speed of light. Thus the range to a target can be calculated as expressed in Equation 2-1, where c is the speed of light and t is the time elapsed between the pulse emission and its reception.

$$R = \frac{ct}{2} \tag{2-1}$$

The factor of two accounts for the fact that the pulse must travel to and from the target before it is measured.

Due to the fact that the power of the radio signal has usually decreased significantly when it is received, another important characteristic of any radar is its maximum radar range. This is determined mostly by properties of the radar itself, and is given by Equation 2-2, where P is the transmitted power (W), G is the gain of the antenna, A is the effective aperture of the antenna (m^2), σ is the radar cross section of the target (m^2), and S is the minimum detectable signal of the radar (W) (Skolnik 30).

$$R_{\max} = \sqrt[4]{\frac{PGA\sigma}{8\pi S}} \tag{2-2}$$

This equation is an oversimplification of a typical situation, but can generally give an approximation of the maximum range, and more importantly instructs the radar user on

various parameters that affect maximum range. A more sophisticated analysis of the radar equation would have to be handled probabilistically by taking into account the possibility of false alarm (Skolnik 31).

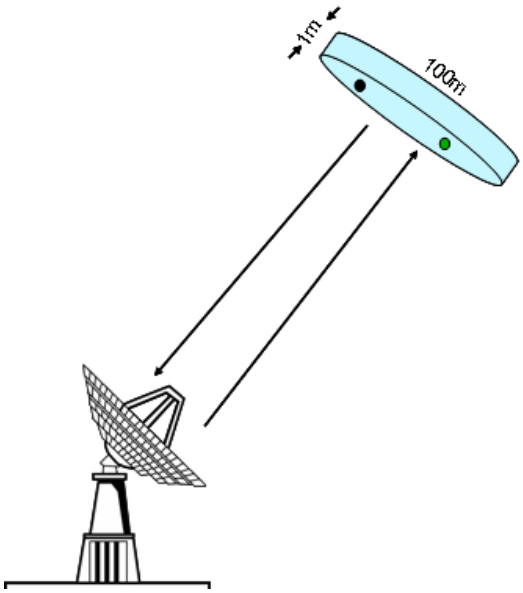

Figure 2-1: Range versus angular resolution

Another interesting feature of radar is its relatively poor angular resolution. Radar is very good at measuring range to an object, due to the constant speed at which electromagnetic radiation travels, but comparatively much poorer at measuring its angle in the sky. As a result, there is always a thin pancake region within which the object could be. A key implication of this is that objects that seem right next to each other on a radar image may actually be quite separate spatially (Weiner).

Radar beamwidth is defined as "the lateral dimension (in angle) of the principal lobe (main lobe or main beam) of an antenna pattern" (Toomay and Hannen 247). The beamwidth determines an antenna's resolution cell, that is, the area of the circle in Figure 2-1. Without the use of multiple radar beams or multiple sweeps we cannot be sure of where in that angle cell a point scatterer is. Angular accuracy δ_θ, when using multiple beams or sweeps, is determined by Equation 2-3. (Toomay and Hannen 115), where θ_{3dB} is the 3dB bandwidth and S/N is the signal to noise ratio.

$$\delta_\theta = \frac{\theta_{3dB}}{\sqrt{2\frac{S}{N}}} \qquad \textbf{2-3}$$

Equation 2-3 only holds true when there is only one object in the angular cell. When there is more than object in an angle cell, as is the case for our project, the two unresolved targets can interfere and appear to be one and the same. The ability to resolve the two targets is directly related to the resolution size (Weiner).

Simulation Software

Our project team used two radar simulation packages while performing our research. One was used as provided, while the other required extensive modification to suit our needs. These simulation packages are outlined in the subsequent sections.

LL6D Trajectory Software

The LL6D (Lincoln Laboratory Six Degrees of Freedom) simulation software is used to create the environments observed by the radar. LL6D is a tool designed for the simulation of ballistic missile threats, and is optimized for quick processing at the expense of simulation detail (Iamaio, 5). The simulation is implemented in Java, but can be interacted with using MATLAB scripts written by Lincoln Laboratory staff and modified by the project team.

LL6D runs off of configuration files (further detailed in the Scenario Definitions section) which describe the objects that will take part in the scenario, and what actions they perform or are performed on them. The code behind LL6D was used as-is, however, we developed additional tools to improve its usefulness, which are described in the Methodology section. LL6D creates trajectory files detailing the motion of the simulated objects using twenty-two different measures. The position and velocity information is recorded in Earth-Centered Inertial coordinates, the angular rates in radians/sec, and the angular position in Earth-Centered Inertial coordinate unit vector component format. Time history files are simpler files, containing the position of the objects in Azimuth, Elevation, and Range coordinates relative to a specified sensor with the angular position in degrees.

It is important to note that LL6D does not perform true six-degrees of freedom simulations, but instead performs 3+3 degrees of freedom simulations. Linear position and velocity calculations are performed independently of those for angular position and velocity. This allows complex situations to be modeled quickly on average desktop computers. Furthermore, this is a reasonable simplification, since the objects used in our simulations are modeled as rigid bodies. The problem of modeling the motion of any rigid object outside the atmosphere can always be split into two easier problems; one can solve for the translational motion of the center of mass independently of the angular motion of the object around its center of mass, and vice versa. This is shown succinctly in Equation 2-4,

$$\mathbf{L} = \mathbf{R} \times \mathbf{P} + \sum \mathbf{r}'_\alpha \times m_\alpha \dot{\mathbf{r}}'_\alpha$$

2-4

where **L** is the total angular momentum of the object as defined by the distance of the center of mass from the origin (**R**), the linear momentum of the center of mass (**P**), and the summation of the angular momenta ($\sum \mathbf{r}'_\alpha \times m_\alpha \dot{\mathbf{r}}'_\alpha$) of each discrete point on the body with respect to the center of mass. The first term in Equation 2-4 models the translational motion of the center of mass from a point of reference, while the second term models the angular rotation of points on the body around the center of mass (Taylor 367-369). LL6D calculates these two components separately when it performs simulations. The center of mass of a body can be easily calculated using Equation 2-5, where *M* denotes the total mass of the body, and $m_\alpha \mathbf{r}_\alpha$ denotes the masses and positions relative to the origin for each discrete point of the body (Taylor 367).

$$\mathbf{R} = \frac{1}{M} \sum m_\alpha \mathbf{r}_\alpha$$

2-5

RFSig

The RFSig (Radio Frequency Signature) software package consists of a MATLAB driver program and extensive Java libraries for high fidelity radar simulation. RFSig performs algorithms in both the time and frequency domains. It is capable of generating plots using all combinations of range, Doppler, and time (Carpenter and Cebula 1). RFSig interfaces closely with LL6D; it combines the trajectory or time history files generated by LL6D with APSM (Augmented Point Scatterer Model) files for each object and a simulated radar - with parameters defined by the user - to produce the desired plots.

APSM files define the reflective characteristics of an object, and are further explained in the Scatterer Definitions section. This software suite was heavily modified by the project team during our summer internship; our goal was to make it more streamlined and user friendly. We designed and implemented a graphical user interface and added supplementary functionality, such as the ability to save and load settings without requiring additional copies of the main program.

Framing the Problem

The analysis we are performing is not taken from actual radar data; we are merely trying to interpret simulations that model possible scenarios. This requires the writing of scatterer definition files to model the geometries of the objects we are observing, and scenario definition files that describe the actual events unfolding.

Scatterer Definitions

Objects are modeled as rigid wireframes in xml files (see Appendix B: Reentry Vehicle XML Code) according to the Augmented Point Scatterer Model, with scattering points that the radio signal reflects off of. The geometries and relative positions of these scattering points are detailed in the file, so the geometry of objects can be easily edited. Furthermore, one can control the strength return from each scatterer, the angular range for which each scatterer is visible, and how sharp the power drop-off outside that range is. This is a useful feature that becomes more evident in the Range-Time Intensity Plots section.

The simulation package we are using contains various scattering models. The dumbbell is the simplest object, composed of two point scatterers separated by a fixed distance. Although it is instructive when learning the essentials of scenario interpretation, it is not very useful when it comes to modeling realistic scenarios. The tank and reentry vehicle (RV) scatterer models are more geometrically complicated and better suited for this purpose.

There are several types of scattering points used in these xml files: point scatterer, slipping, specular, and cavity returns. Point scatterers are rather self explanatory; a point scatterer is a salient zero-dimensional feature that simply reflects the radio signal back to the antenna at reduced power. Physical examples of point scatterers include the nose of a cone, antennae, and the tips of wings. Slipping returns behave similar to point scatterers, but they are not fixed to a point on the object. Slipping returns are usually found on curved surfaces, such as the side of a cylinder. As the cylinder spins, the slipping return "moves" in the opposite direction so that it always faces the radar.

Specular returns typically characterize any flat surface on the object. This return is only seen when the surface is near perpendicular to the radar line of sight; at this time it sends back a very strong return. Cavity returns model any openings or depressions that may exist on the object. When the radio signal enters a cavity, it bounces off the walls of the

cavity, and returns a rather chaotic signal. This generates noise on the radar tracks that is directly proportional to the depth of the cavity.

Scenario Definitions

The scenarios we use are defined with the use of configuration files. These are text files that outline the objects involved and the events that are enacted upon them (see Appendix C: Sample Configuration File). Various parameters such as the objects' masses, moments of inertia, and the gravity model are outlined. The time at which each event occurs and what objects are affected are also recorded in the configuration file.

One can chose from a wide variety of events to implement, from simple modifications of an object's angular velocities to complex ballistic missile guidance algorithms. LL6D is written such that it can read in these configuration files for a simulation as long as they are written following the guidelines in the LL6D manual. The events most important to us are "DeployVehicle" events, which outline the characteristics of a splitting event. We also make extensive use of "setState" events, which enable editing of the objects' velocities; this is useful when characterizing a crossing event.

Range-Time Intensity Plots

There are many types of images that can be generated using the data supplied by a radar. For the purposes of our project we will be studying Range-Time Intensity plots (RTIs). The analysis of RTIs comprises the bulk of our work, so we must have a thorough understanding of how to read them. RTIs illustrate how the different scattering points on objects move over time. Often this motion is periodic because the object in question has some rotational velocity, also known as the object's tumble rate.

As can be seen in Figure 2-2, relative range is shown on the x-axis. This range is the distance to the radar relative to the observed object's center of rotation. Time is shown on the y-axis, and is measured in seconds since the start of the simulation. The intensity of the return is shown using a color axis, and is measured in decibels relative to a square meter (dBsm).

Dumbbell RTI

The dumbbell scattering model is composed of two scattering returns attached by a rigid, non-reflective rod. Figure 2-2 shows a simple example of what the RTI would look like for a tumbling dumbbell. The physical scenario is illustrated to the left of the RTI; the position of the dumbbell is shown with relation to the radar line of sight (RLOS) at three different times.

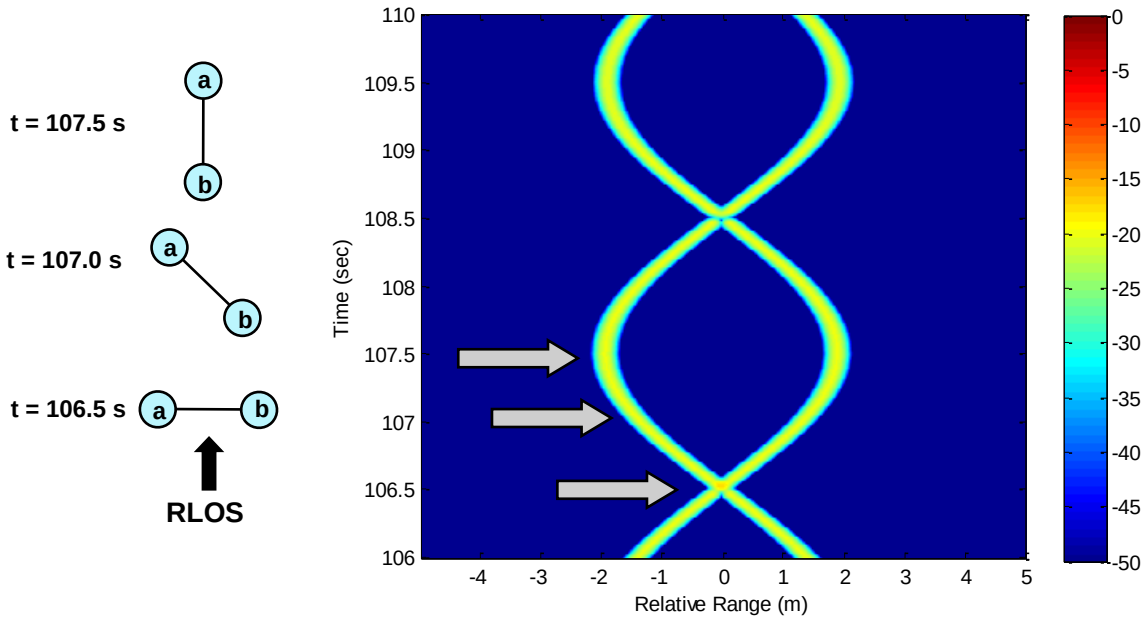

Figure 2-2: Rotating Dumbbell Diagram and RTI

At t = 106.5 s, the dumbbell is perpendicular to the RLOS. Consequently, scatterers *a* and *b* are equidistant from the radar, causing their respective scatterer tracks to overlap in the corresponding RTI (the center of rotation has no radar track since it is not a scatterer). A minor aside: in truth the scatterers are slightly further away than the center of rotation, but this distance is negligible for the usual case where the dumbbells are very short in comparison to the much larger distance to the radar antenna (see Appendix A: Scatterer Distance Clarification).

At time t = 107 s, scatterer *a* is further away from the radar than scatterer *b* due to the tumbling nature of the dumbbell. This results in a positive relative range for *a* and a negative relative range for *b*. At time t = 107.5 s, the dumbbell has tumbled 90 degrees and is aligned parallel to the RLOS. Thus scatterer *a* has reached its apex and is the furthest it will be from the radar with respect to the center of rotation. On the other hand, scatterer *b* is at

its nadir and is the closest it will be to the radar with respect to the center of rotation. This is shown on the RTI by the two very separate tracks.

Reentry Vehicle RTI

The RV scattering model used for our simulations assumes a solid cone-shaped object with no cavities. It has returns at the base and the nose. Figure 2-3 depicts an RTI of an RV tumbling nose over base with its center axis parallel to the RLOS. Between t = 109 s, and t= 110 s, it can be seen that one of the tracks on the RTI disappears. This is due to a phenomenon called shadowing. If the RLOS is perpendicular to the base, as shown in Figure 2-4, the radar cannot see the nose and thus its track disappears from the RTI. This is logical since the base shadows the nose from the sight of the radar (Weiner). This is also the raison d'être for controlling the angular visibility of objects in the APSM files (as described in the Scatterer Definitions section).

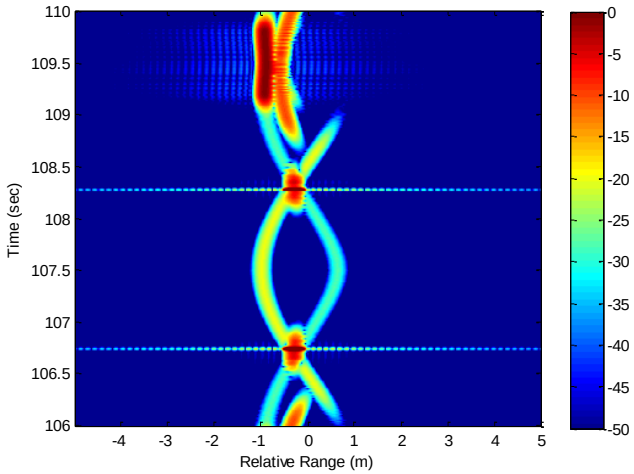

Figure 2-3: RV RTI

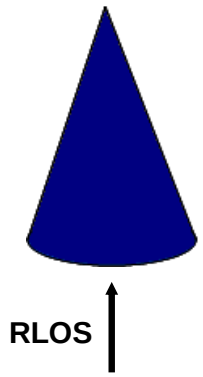

Figure 2-4: RV Base Shadowing Nose

Tank RTI

The tank scattering model is cylindrically shaped and has openings at both ends. The openings are referred to as cavity returns. Figure 2-5 depicts an RTI of a tank tumbling end over end with its center axis parallel to the RLOS. The noisy returns seen in the RTI are a result of the radar signal entering these cavities, bouncing around inside, and returning chaotically (Weiner). One of the ends has a deeper cavity than the other, resulting in different cavity return levels depending on the orientation of the tank. Although less apparent than the RV, shadowing can also be seen on the tank RTI at t ≈ 106.3s and at t ≈ 108.7 s.

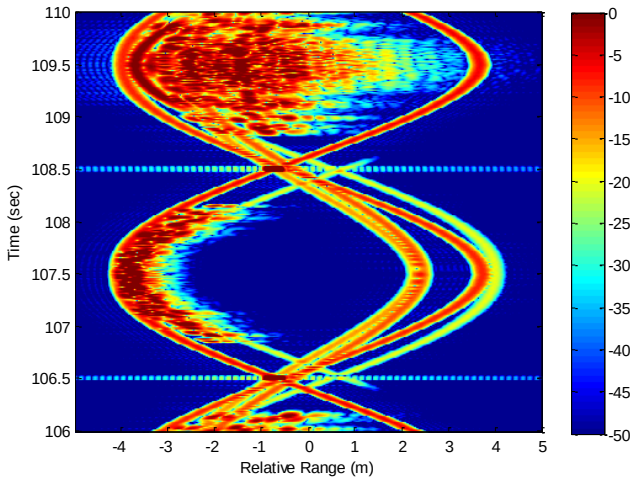

Figure 2-5: Tank RTI

Intersection Angle

The slope of a radar track on an RTI is a graphical representation of its corresponding scatterer's velocity relative to the center of the tracked object. This is a reasonable conclusion, considering the x-axis is measured in meters and the y-axis is measured in seconds. The slope of a track is intrinsically equal to its rise divided by its run, and is measured in seconds per meter. It is then clear that low magnitude slopes correspond to high relative velocities, since fewer seconds would elapse per meter traveled. Similarly, high magnitude slopes indicate low relative velocities, as more seconds would elapse per meter traveled. Positive slopes indicate that the scatterer is moving away from the radar antenna; comparably, negative slopes imply that the scatterer is moving closer to the radar antenna.

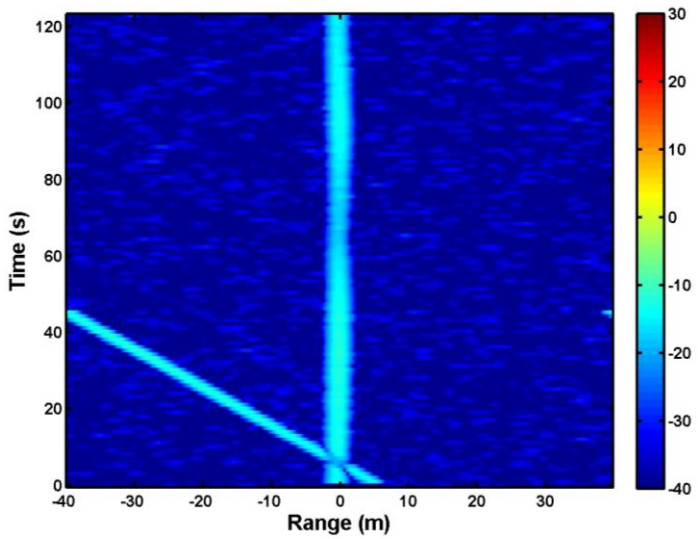

Figure 2-6: Two-track RTI example

Intersection angle is directly related to speed, and this characteristic of RTIs is particularly pertinent to our project. In our simulations we always track two objects and center the RTI on one of them. This results in an RTI that looks similar to Figure 2-6. The relative speed of one object is held constant at zero, and the relative speed of the other object is represented by its intersection angle with the track of the normalized object. When the relative speeds are low, the intersection angle is small, and this can make differentiating between crosses and splits more difficult. Conversely, if the relative speed is large, the intersection angle is big, and this can simplify the process of discerning between crosses and splits.

3. Methodology

The three main objectives of our project are as follows:

- To develop a set of heuristics that will allow us to decide whether an event is a cross or a split.
- To develop a human decision-making model that codifies these heuristics.
- To produce operating curves that exhibit the effectiveness of the human decision-making model.

Due to time constraints and the fact that we could not access real radar data we had to reduce the scope of our project by limiting the number of cases we modeled and by making certain assumptions. The program we wrote to randomize the RTI generation process chooses between four scenario templates: a cross with or without tumble (see Figure 3-1 and Figure 3-2), or a split with or without tumble (see Figure 3-3 Figure 3-4).

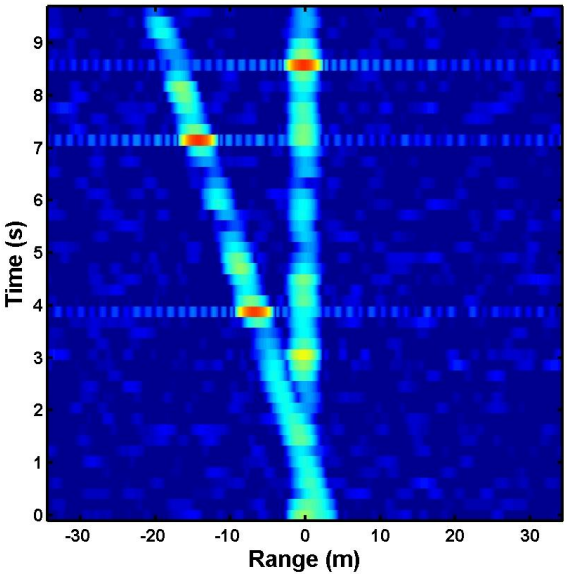

Figure 3-1: Cross with tumble

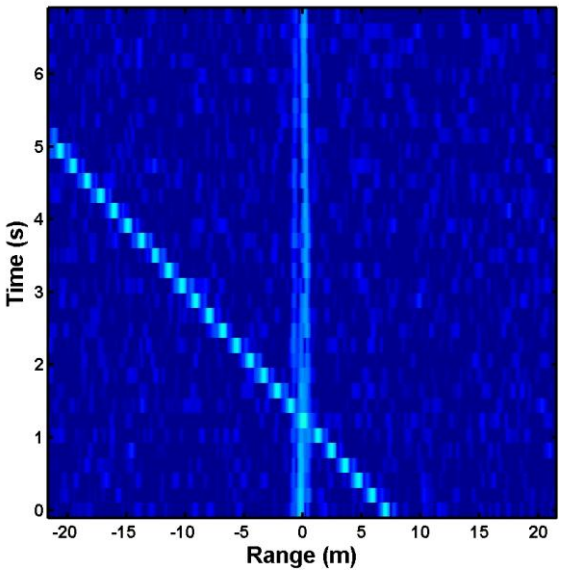

Figure 3-2: Cross without tumble

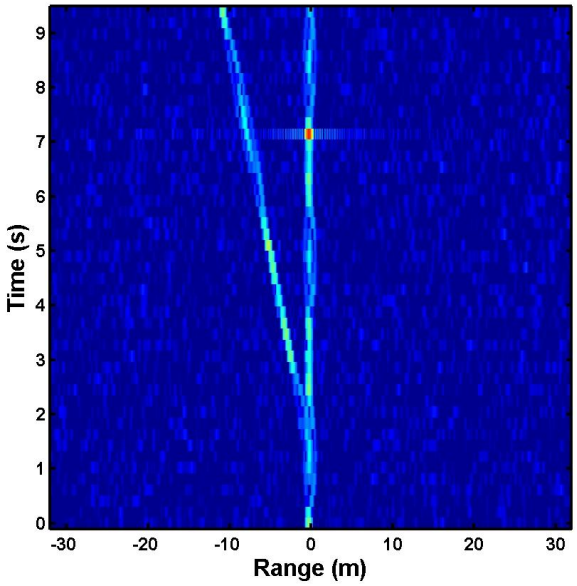

Figure 3-3: Split with tumble

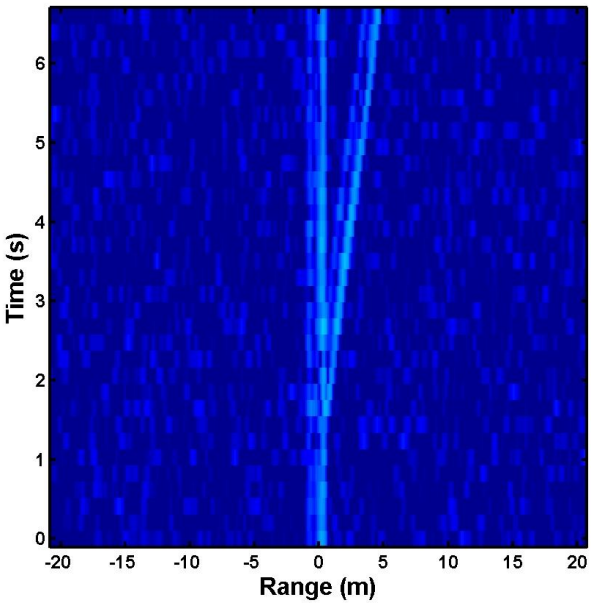

Figure 3-4: Split without tumble

Additionally, we track only two objects at any given time; these objects are modeled as identical RVs. Each Monte Carlo simulation corresponded to a specific bandwidth, relative velocity and observation time before and after the event.

For the purposes of our project we are assuming crosses happen between targets that originate from a common object, and thus their relative speed is smaller than it would be if they were completely unrelated. However, splitting speeds are even smaller because the objects are separating under small forces generated by springs or small thrusters right around the time we start tracking them. We also assumed that the spread of crossing velocities would, for the most part, be wider than the spread of splitting velocities. Both the splitting and the crossing speeds were approximated because we do not have access to statistical data associated with these variables.

Most of the RTIs we generated were edge cases with respect to the time before the event. It is easier to maintain a track than to start one, therefore it is logical to assume we would have more time after the event than before it. We are also focusing on edge cases because they are non-trivial to interpret.

Developing heuristics to distinguish a split from a cross

Before we could translate our mental discrimination process into a human decision-making model we had to develop a set of heuristics that would allow us to distinguish between a cross and a split. In order to do this we had to identify parameters that we (the supposed radar operators) would know, and could thus base our heuristics on. The first of these parameters is the radar's bandwidth, a known technical specification. The other two parameters are the speed/intersection angle of the tracked objects, and time before/after the event, both of which can be estimated from the RTI.

Performing exploratory exercises

Once the relevant parameters were identified, we performed two sets of exploratory exercises. Each of us examined large sets of randomly generated RTIs, labeled each as a split or a cross, and wrote down the reasoning behind our decision. We then compared our decisions with the actual events, logged in a record file produced by the RTI generation program, and took note of our accuracy. The exercises allowed us to characterize our RTI generation program (see section 0), that is, to verify that the process was random enough that we were not recognizing patterns and that there was an adequate ratio of edge cases to obvious ones. Additionally, they provided an indication of how each of the observables (bandwidth, speed/intersection angle and time before/after) affected our ability to discriminate between splitting and crossing targets.

Exercise 1

The purpose of the first exercise was for each of us to independently examine large sets of RTIs and to determine, each using our own method, whether we were looking at a split or a cross. The overall false identification rate was 14%. For the purposes of illustrating the nature of this exercise, let us discuss one team member's approach. He examined a set of 129 RTIs. A list of reasons used to determine the nature of the event as well as the error rates for each reason can be seen in Table 3-1. It was clear from this set that there were too many obvious cases. To provide an example of what we considered obvious, Figure 3-5 and Figure 3-6 show non-obvious cases on the left and obvious cases on the right.

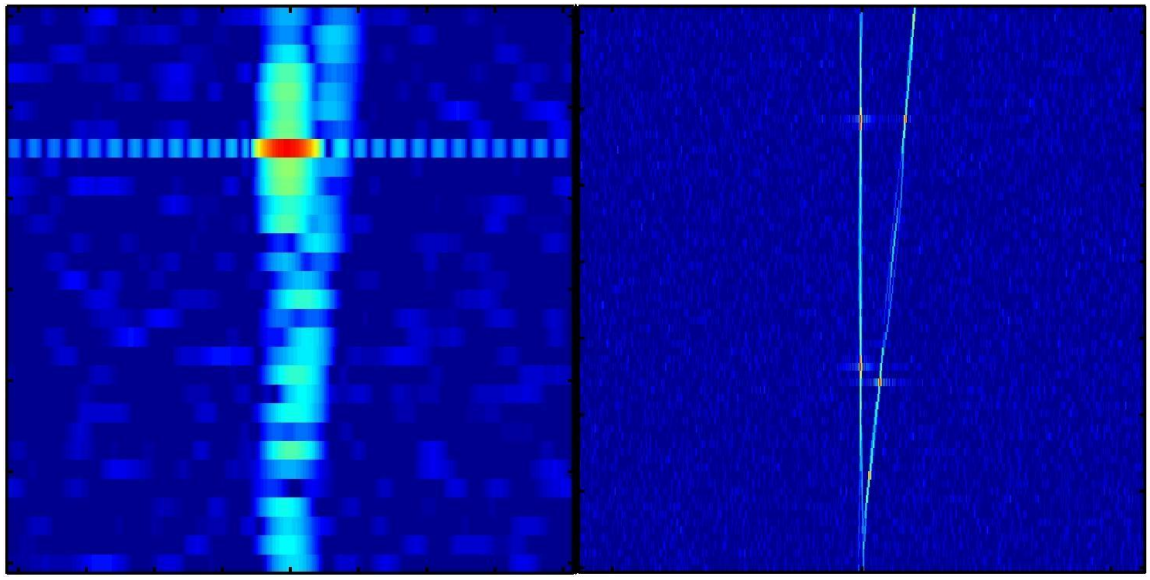

Figure 3-5: Obscure split versus obvious split

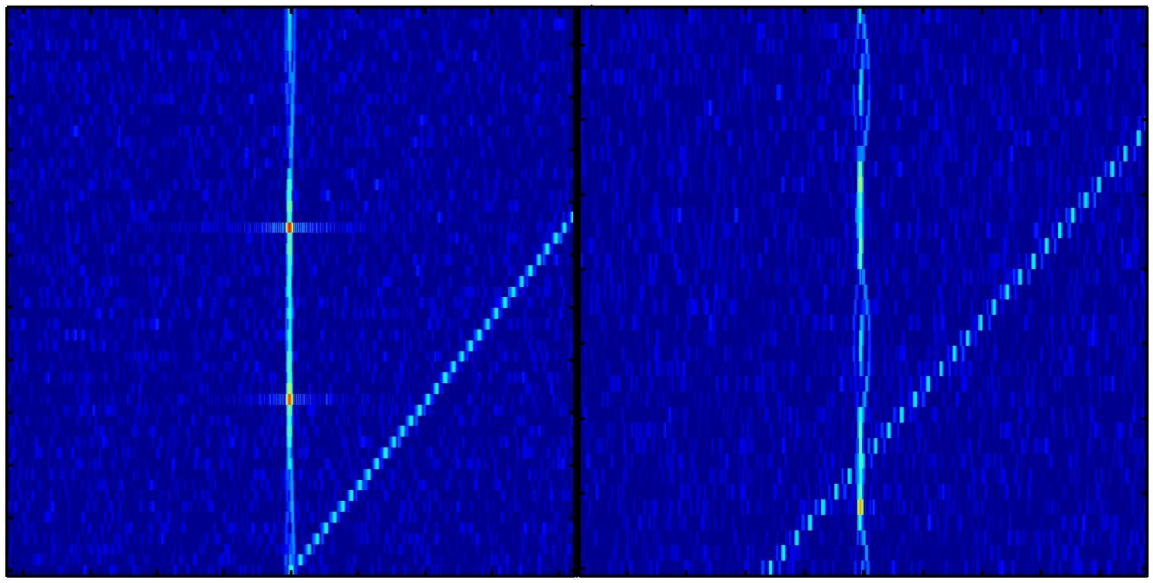

Figure 3-6: Obscure cross versus obvious cross

These figures also convey how changing the bandwidth and the time before the event can greatly affect the ease of interpretation.

The program was changed to emphasize difficult cases: the percentage of obvious cases was decreased from approximately 50% to about 30%. This first exploratory exercise also gave us a feel for how each of us was making decisions, and made it easier for us to standardize a system of decision labeling that we used in the next exploratory exercise.

Reason	% of decisions affected	% Error rate	Number of false identifications
Obvious	50.39	4.62	3
Tumble of second object originates at event	6.20	25.00	2
Large intersection angle (cross)	5.43	0.00	0
Small intersection angle (split)	22.40	20.69	6
Blind guess	2.33	33.00	1
Other	12.40	25.00	4
TOTAL	100.00	12.4	16

Table 3-1: Exercise 1 Results

Exercise 2

For this exercise, we used a more standardized approach; we all used the same labeling system for our decisions. This exercise resulted if further refinement of the parameters of our simulations, and illustrated the need for normal distributions of crosses and splits. With the uniform distributions we initially used, it was too easy to tell if an event was a cross because any event above a certain intersection angle was *always* a cross. Gaussian distributions remove this certainty and more accurately reflect the physical situation. After this exercise, we organized our rules into more rigid heuristics and set the foundation what would eventually become our human decision-making model. The results of this exercise can be seen in Table 3-2. It should be reemphasized that the data from these exercises was not

used at all in our final analysis; these exercises merely helped us hone the methodology and heuristics that would be used to analyze our ultimate data set.

Reason	% of decision affected	% Error rate	Number of false identifications
Time Before	32.64	2.11	3
Large intersection angle (cross)	8.74	0.00	0
Small intersection angle (split)	47.13	32.68	67
Size of track	10.57	19.57	9
Other	0.46	100.00	2

Table 3-2: Exercise 2 Results

Quantifying the heuristics

After performing the exploratory exercises we narrowed down our reason pool to three rules. In order to apply these rules in a systematic fashion, we first needed to quantify them. How we attached numbers to each of the heuristics is explained in subsequent sections.

Time before rule

As previously explained, most of the RTIs we generated are edge cases with respect to the time before the event. We determined that two objects can be resolved before an event occurs if their tracks are at least *L* (the width of the largest track) apart. The time before the event and this minimum separation distance form a right angle (see Figure 3-7). Since the intersection angle can be calculated from the RTI, we can use trigonometry to determine t_{sep}, the minimum time needed in order to resolve two objects before the event (see Equation 3-1).

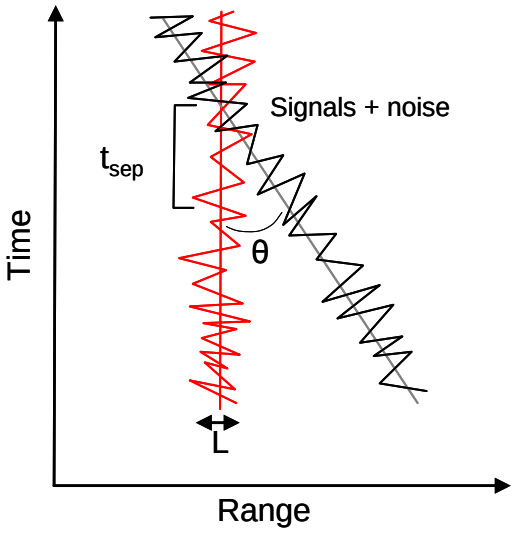

Figure 3-7: Time before rule

$$t_{sep}[s] = \frac{L[m]}{\tan(\theta) \cdot \frac{1m}{1s}}$$

3-1

The magnitude of the ratio of the opposite side L over the adjacent side t_{sep} is equal to tan(θ). Since this is not a standard x versus y graph, but rather an x versus time graph, the units are carried by a constant $1m/1s$.

Width rule

When the time before the event was greater than zero but smaller than t_{sep} we used the width rule when applicable. If the track before the event had a width equal to L (the width of the central track) it suggested the presence of only one object before the event, and therefore we labeled it a split. If the track before the event had a width greater than L, indicating multiple objects before the event, we determined the event to be a cross. If we felt that the case was too ambiguous, we did not use this rule.

Intersection angle rule

As previously discussed, the angle at which two objects intersect on an RTI is directly related to their relative speeds. As explained in section 3.1.1, we assumed the crossing velocities to have a wider spread than the splitting velocities. Our original system randomly chose these velocities from a uniform distribution. Crossing velocities ranged from 0.1 m/s to 6 m/s while splitting velocities spread from 0.1 m/s to 3 m/s. To minimize the error rate we set the decision threshold at the intersection of both distributions; this meant that if the angle rule was applied, any intersection of greater than 3 m/s was classified as a cross, while intersections of less than 3 m/s were labeled splits. In doing this we obtained a 0% false cross rate and a 50% false split rate, for a total error rate of 25% when using the angle rule.

In order for our program to better reflect reality we changed these distributions from uniform to Gaussian. The normal distributions were characterized by the following means and standard deviations:

- $V_{cross} \sim \mu=5$ m/s, $\sigma=2$ m/s

- $V_{split} \sim \mu=3$ m/s, $\sigma=1$ m/s

  Keep in mind that a generic normal distribution is characterized by Equation 3-2.

$$P = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

3-2

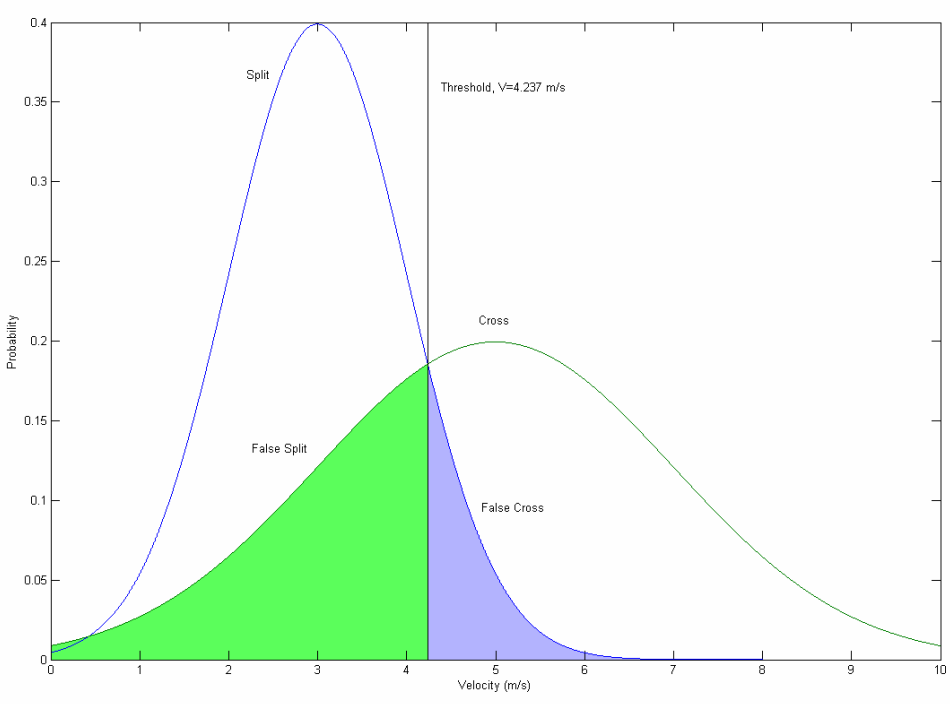

Figure 3-8: Crossing and splitting velocity distributions

For the derivation of the optimum threshold as seen in Figure 3-8, refer to Equations 3-3, 3-4 and 3-5. The optimum threshold is the one that produces the lowest error rate.

$$Error\,Rate = FalseCross + FalseSplit = \frac{1}{\sigma_c\sqrt{2\pi}} \int_{-\infty}^{\tau} e^{\frac{(x-u_c)^2}{2\sigma_c^2}} dx + \frac{1}{\sigma_s\sqrt{2\pi}} \int_{\tau}^{\infty} e^{\frac{(x-u_S)^2}{2\sigma_s^2}} dx$$

$$= \frac{1}{\sqrt{2\pi}} \left(\frac{1}{2} \int_{-\infty}^{\tau} e^{\frac{(x-5)^2}{8^2}} dx + \int_{\tau}^{\infty} e^{\frac{(x-3)^2}{2}} dx\right)$$

3-3

$$Threshold = \tau = \min(Error\ Rate) -> \frac{d}{dx}\frac{1}{\sqrt{2\pi}}\left(\frac{1}{2}\int_{-\infty}^{\tau} e^{-\frac{(x-5)^2}{8^2}}dx + \int_{\tau}^{\infty} e^{-\frac{(x-3)^2}{2}}dx\right)$$

$$= .2e^{-\left(\frac{-\sqrt{2}\tau}{4}-\frac{5\sqrt{2}}{4}\right)^2} - .4e^{-\left(\frac{-\sqrt{2}\tau}{2}-\frac{3\sqrt{2}}{2}\right)^2}$$

3-4

$$Zeros\left(\frac{d}{dx}(Error\ Rate)\right) = 4.237\,m/s$$

3-5

At this threshold, when using only the intersection angle rule, the expected total error rate was 22.97%. The false split error rate was 35.14% and the false cross error rate was 10.8%. This assumed an equal likelihood of splits and crosses.

Since intersection angle is more easily observed on an RTI than relative velocity, we converted the velocity threshold of τ=4.237 m/s to an equivalent intersection angle threshold τ_{angle} of 27.9 degrees (see Equation 3-6).

$$\tau_{angle} = \arctan\left(\frac{\frac{dist}{rangescale}}{\frac{time}{time\,scale}}\right) = \arctan\left(\frac{\frac{42.37m}{80m}}{\frac{10s}{10s}}\right) = 27.9°$$

3-6

Most applications of the intersection angle rule were for cases when there was no time before the event. The time at which the event occurred was calculated by backtracking the paths followed by the objects on the RTI.

Developing a human decision-making model

Two issues we encountered while performing our simulations were the bias and error introduced by using humans to conduct the RTI analysis. In nearly all real radar systems computers are used to perform the bulk of the analysis as they can operate faster and more consistently than humans. This comes at a cost, however, in development time. Computer

algorithms that guide correct interpretation of data from complex sensors such as radars require thousands of man-hours to write and test. Thus, with our time constraints making a computer-based interpretation system unachievable, we designed our procedure and tools to be as efficient and consistent as possible while guarding against human biases.

Organizing the Heuristics

The heuristics developed in section 0 provided several methods of interpreting RTIs, with varying degrees of accuracy and applicability. To optimize the heuristics they were organized for accuracy and efficiency. The two deterministic heuristics - Time Before and Width - are applied first. If either of these rules was applied, then the correct answer was guaranteed, excepting the small chance of operator error. Time Before was performed prior to Width because it was quicker to apply and less susceptible to operator error. The Angle rule was applied last for two reasons. First, it is probabilistic, and has a certain percentage of error even when applied correctly. Second, in contrast to the previous two heuristics, it can be applied to all RTIs. The resulting instruction set is shown below:

I. Time/Angle rule
    a. If $t_b >= t_{sep}$
        i. If $n_{track}=2$ then CROSS
        ii. If $n_{track}=1$ then SPLIT
        iii. If inconclusive, skip to II
    b. If $0< t_b < t_{sep}$ skip to II
    c. Else skip to III

II. Width rule
    d. If $w(t=0) > L$ then CROSS
    e. If $w(t=0) \leq L$ then SPLIT
    f. If inconclusive, skip to III

III. Angle rule
    g. If $\theta > \tau$ then CROSS
    h. Else SPLIT

The first version of this instruction set was tested by the team on 200 RTIs, which exposed several minor issues mostly related to language ambiguities that caused correct rules to be skipped. Following revision, an additional test was conducted using the instruction set presented in this report on another 200 RTIs. These tests presented error rates of 9.5% and 12%, a significant improvement over the results obtained prior to the creation of a standardized instruction set (Exercise 1 and Exercise 2 in section 0). An additional

contributor to the decrease in error during these tests was the introduction of automation for certain interpretation tasks, such as measuring the intersection angle and time before the event. Automating these tasks with Matlab greatly improved interpretation accuracy while also decreasing the amount of time required to examine each RTI.

Generating the Range-Time Intensity Plots

To generate curves illustrating the performance of our heuristics, a large sample size was necessary. As the project team did not possess the necessary security clearances, use of real data was not an option, so simulated RTIs needed to be used. The simulated RTI generation process - as implemented using the original Lincoln Laboratory software - was cumbersome and slow, requiring in excess of five minutes per RTI, all of it demanding the presence of a human operator. Applying these heuristics many times to a small set of RTIs would introduce the potential for heavy bias, as the nuances of each RTI would sink into the observer, such that correctly identifying the event occurring in the RTI would depend on factors that a realistic observer would not have available. Thus, the project team needed to develop a system for rapidly generating large numbers of RTIs with varying parameters.

The new and improved RTI generation system was implemented as a MATLAB program split into three primary script files: RandRTI.m, LL6DMatlabnMQP.m, and RunsimMQP.m. The code for these scripts can be found in Appendix E: MATLAB Code Used For RTI Generation. The first script, RandRTI, was written from scratch. This file contains the code that controls the simulation, allowing the user to set the values for fixed variables and the bounds for random variables, as well as indicate what directories the program will use. RandRTI contains code for each template scenario used by the project. These templates are LL6D configuration files (see Appendix C: Sample Configuration File) which have been written to describe a particular event, but with holes for certain randomized variables. After completing its initial tasks, the program enters a loop, with the number of iterations equal to the number of RTIs that will be generated. The first action performed in the loop is to randomly choose one of the templates.

In addition to the template specific variables, each template is also subject to variation in bandwidth and the amount of time visible before and after the event. The scenario variables are shown in Table 3-4. The templates are described in Table 3-3.

ID	Name	Description	Template-Specific Variables
1	CrossL	Cross, second object moving to the left	Relative Velocity Starting Distance
2	Split	Split, second object moving to the right	Split Time Split Velocity
3	CrossTumbleL	Cross with tumbling objects, second object moving to the left	Relative Velocity Starting Distance Tumble Rate for each object
4	SplitTumble	Split with tumbling objects, second object moving to the right	Split Time Split Velocity Tumble Rate for each object
5	CrossR	Cross, second object moving to the right	Relative Velocity Starting Distance
6	CrosSTumbleR	Cross with tumbling objects, second object moving to the right	Relative Velocity Starting Distance Tumble Rate for each object
7	MQPSplit(-V)	Split, second object moving to the left	Split Time Split Velocity
8	MQPSplitTumble(-V)	Split with tumbling objects, second object moving to the left	Split Time Split Velocity Tumble Rate for each object

Table 3-3: Scenario Templates

Variable	Distribution	Description
Bandwidth	Discrete: 100, 500, 1000 (MHz)	Radar Bandwidth
Time Before	Uniform: -1.5 to 1 (s)	Time Before Event
Time After	Uniform: 5 to 15 (s)	Time After Event
Relative Velocity	Normal: μ=5, σ=2 (m/s)	Closing Velocity of Crossing Objects
Starting Distance	Uniform: 2 to 15 (m)	Starting separation for crossing objects
Split Time	Uniform: 20 to 150 (s)	Time of separation for splitting objects
Split Velocity	Normal: μ=3, σ=1 (m/s)	Separation velocity for splitting objects
Tumble Rate	Uniform: .1 to 2 (Hz)	Rate at which objects spin about their center point

Table 3-4 : Scenario Variables

The velocity distributions were modeled as normal distributions to account for the fact that most objects originating from the same object would have similar relative velocities. The bandwidth was randomly chosen from a set of three possibilities; this allowed us to see how our results would change if a radar with a different bandwidth was used. The other variables were chosen at random from uniform distributions within certain imposed limits. The limits were chosen so as to avoid having a plethora of trivial cases, hence the Time Before parameter has a very narrow range to choose from.

Once the program generated the random values for the selected template, it performed three tasks. First, it wrote the results of the randomizations to a record file, which allowed us to go back and find out the parameters that generated any given scenario. Second, it wrote the actual configuration file, formatted to be read in by LL6D. Third, it began execution of the LL6DmatlabnMQP script.

LL6DmatlabnMQP is a modified version of a script originally written by Lincoln Laboratory staff. Its original function was to produce time history or trajectory files for a given configuration file. To make the program perform as we needed, we changed it in several ways. The first was to alter the way the program performed input and output to suit the needs of runsimMQP, the third script. This primarily involved setting up specific file

paths and changing how output files were named. The second change was to add calculations that would automate some of the RTI interpretation. The program calculates the distance between the two objects at the end of the timescale and uses the result, along with the dimensions of the RTI, to calculate the intersection angle as it is displayed on the screen. These calculations are then returned to RandRTI, which reformats them and feeds them into runsimMQP.

RunsimMQP is another program which was not originally written by the project team, but was modified first during the team's summer internship at Lincoln Laboratory and then further during the course of the project. This program is intended to combine time history files describing a physical situation with scatterer files describing the radar cross section of each object to produce an RTI. The modified version changes the file creation string to include information that assisted in our RTI interpretation. The filename contains the bandwidth, time before, time needed before (calculated using the method in section 0), object width, and intersection angle. Having this data immediately viewable while looking at the RTI greatly improved accuracy and decreased time needed to view each one.

Demonstrating the Effectiveness of the Human Decision-Making Model

Once we developed our human decision-making model, the next step was to test its effectiveness. We generated and analyzed 3001 RTIs. This gave us a large sample size and many varying combinations of intersection angle, bandwidth, and time before the event. A larger sample size with even more RTIs would have been better for statistical analysis, but time constraints required us to limit it.

Using our decision-making model, we examined the RTIs and documented our answers in an Excel sheet. We also recorded the specific route used by the human decision-making model to reach the decision for each RTI. Excel then made it easy to calculate the overall error rate of our human decision making model, and furthermore, the level of accuracy of each individual heuristic in the model. The next step was to condense this deluge of data into something readable and presentable. We decided to construct operating curves that show the overall effectiveness of our human decision-making model. An example operating curve of this nature can be seen in Figure 3-9

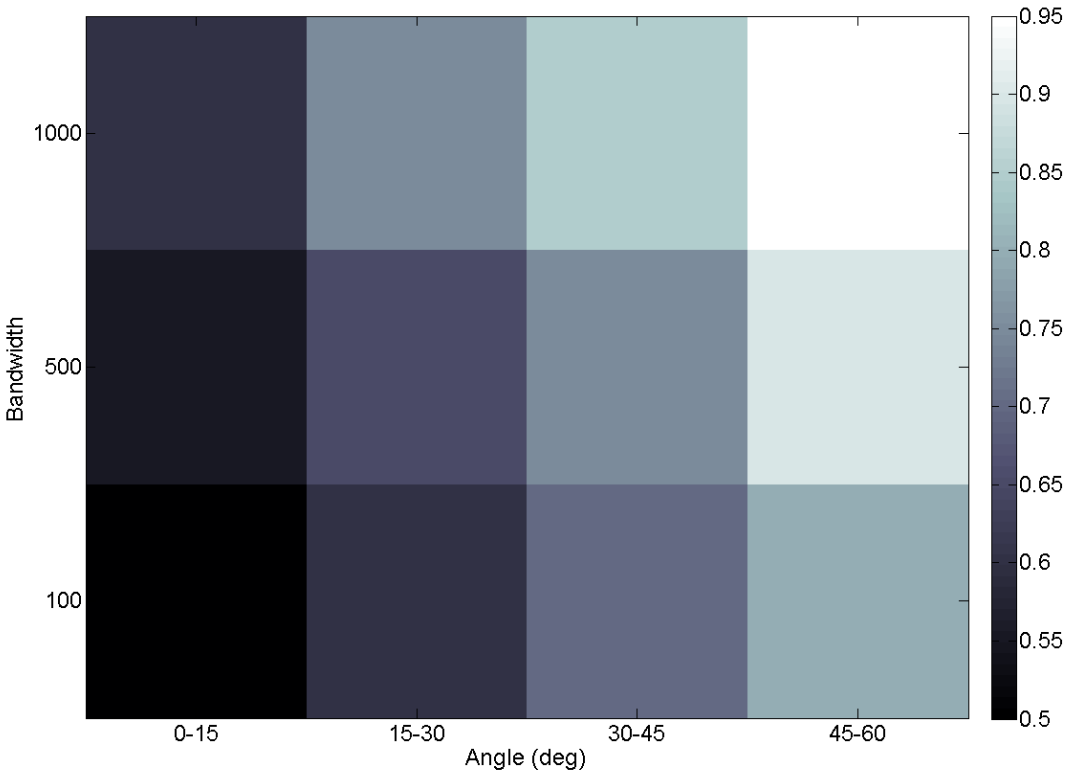

Figure 3-9: Sample operating curve

It can be seen that the operating curve displays three dimensions of data: intersection angle on the x-axis, bandwidth on the y-axis, and probability of correct identification on the color-axis. We used three distinct bandwidths in our RTI generation software, hence the 3 separate rows. To enable calculations of realistic probabilities, we quantized the x-axis. Since our software randomly generates the intersection angle, it was highly unlikely that any angles would be repeated on multiple RTIs. Quantization allowed us to combine the data from different angles in the same neighborhood, and calculate a probability of correct identification for that specific neighborhood of angles.

The probability is illustrated by the color of the cell; darker shades correspond to higher probabilities. It is logical that the lowest probability of correct identification is near 0.50, since one can always randomly guess if the event is a cross or a split. This operating curve example relates bandwidth to intersection angle, but we also created a curve relating the time before the event to intersection angle. In this case, we quantized the time axis for the same reasons as the intersection angle axis.

If we could display the probability of correct identification with respect to time before the event, bandwidth and angle on a four dimensional graph the operating curve would probably be smoother and easier to understand.

4. Results and Discussion

In order to achieve the major goal of this project we had to first accomplish a series of intermediary tasks. First we wrote a program to randomly generate RTIs and we performed exploratory exercises to quantify our heuristics. The results of these exercises, together with observation and research allowed us to quantify the heuristics. Most importantly we were able to organize them logically into the human decision-making model shown below. The bold italicized numbers were used to identify at what point in the model we stopped and made our decision when analyzing a given RTI.

I. Time/Angle rule
    a. If $t_b \geq t_{sep}$
        i. If $n_{track}=2$ then CROSS
        ii. If $n_{track}=1$ then SPLIT
        iii. If inconclusive, skip to II
    b. If $0< t_b < t_{sep}$ skip to II
    c. Else skip to III
II. Width rule
    a. If $w(t=0) > L$ then CROSS
    b. If $w(t=0) \leq L$ then SPLIT
    c. If inconclusive, skip to III
III. Angle rule
    a. If $\theta > \tau$ then CROSS
    b. Else SPLIT

These subtasks and their outcomes are presented in the Methodology because they guided the development the project. Each minor outcome was used as stepping stone to reach the next accomplishment. This process culminated with the creation of the set of operating curves that describe our human decision-making model.

In our final round of testing we examined 3001 RTIs and tabulated the data in Excel spreadsheets similar to Table 4-1.

Decision	Reason	Angle	Bandwidth	Time Before	True Label
S	6	7.812	1000	-.017	C
S	6	3.681	100	.391	S
C	3	39.607	500	.106	C
S	2	19.563	500	.601	S

Table 4-1: Sample data record

We recorded the decision made and the reason behind it, that is, at what point we stopped at in the human decision-making model. To allow more complex analysis of our key discrimination parameters we also recorded the intersection angle, bandwidth, and time before the event for each RTI. Our RTI generation software prints a record file (see Appendix D: Sample Record File) that includes the details of each RTI. Looking at this we were able to record each event's true nature and therefore calculate our accuracy. The Excel spreadsheet was imported into MATLAB to produce operating curves that illustrate the probability of correct detection for different values of angle, bandwidth, and time before the event.

Initial System Performance

Once our data was recorded in the Excel spreadsheet we were able to characterize our system's performance in different ways. Table 4-2 presents a brief summary of our data - number of samples and error rates according to the nature of the event. Table 4-3 is more detailed and breaks down the number of samples and error rates according to what rule was used to identify the event. We consider the mistakes made using the time before and width rules human error and those made using the angle rule probabilistic error.

As can be seen below, the initial system performance indicated areas for improvement. These improvements were made and the revised system performance can be seen in section 0.

	Number of samples	Percent Of Total
Samples	3001	100.00%
Correct	2537	84.54%
Errors	464	15.46%
Cross	1409	46.95%
Splits	1591	53.02%
False Split	409	21.01%
False Cross	54	5.12%

Table 4-2: Data summary

	Time Before Rules		Width Rules		Angle Rules	
Reason (See Human Decision-Making Model)	1	2	3	4	5	6
Number of Samples	581	457	179	228	296	1259
Percent of Total	19.36%	15.23%	5.96%	7.60%	9.86%	41.95%
Combined Number of Samples	1038		407		1555	
Combined Percent of Total	34.59%		13.56%		51.82%	
Number of Errors	44		27		393	
Error Rate When Rule Used	4.24%		6.63%		25.27%	
Percent of Total Error	9.48%		5.82%		84.70%	

Table 4-3: Human Decision-Making Model Performance

In order to produce our initial curves we used the angle threshold of $\theta_t = 27.9°$ calculated in section 0. This meant that anytime we used the angle rule to make a decision, we simply labeled the RTI a cross if the intersection angle was greater than this threshold, and a split if it was less than this threshold. To calculate this angle we minimized the error rate function (see Equations 3-3, 3-4 and 3-5). The physical analogue of this function is the sum of false splits and false crosses. We calculated the error rates at this threshold, assuming an equal likelihood of a split and cross and the application of only the angle rule. Our overall error rate should have been 22.97%, our false split error 35.14% and our false cross error 10.80%. Applying the human decision making model, the angle rule in conjunction with the

time before rule and the width rule, resulted in a significant reduction of these error rates (see Table 4-2).

We created the curves in subsequent sections to graphically characterize our system's performance. The two main relations we modeled were bandwidth versus angle and time versus angle.

Probabilities of Correct Identification with Respect to Bandwidth

We wrote a MATLAB program (see Appendix F: MATLAB Operating Curve Code Example) that could both read in the Excel spreadsheets we recorded our data in and produce a variety of operating curves. The first one we will discuss is Figure 4-1, which illustrates the effect of bandwidth and intersection angle on the probability of correct identification.

The MATLAB program we wrote to build the operating curves strives to maintain statistical significance by ensuring that each angular neighborhood had 120 data points (distributed among the three bandwidths). As a consequence of this, and the fact that the angles were chosen from a Gaussian distribution (see Section 0), the width of each angular neighborhood varies.

It can be seen that the darker neighborhoods correspond to higher probabilities of correct decision-making for their respective parameters. However, aside from seeing a sharp drop-off in probability near the threshold, not much information can be gleaned from Figure 4-1. To make the operating curve more readable we smoothed it by using a five point moving average – taking the mean of the probability of correct detection in one neighborhood and that of the two neighborhoods on either side of it (see Figure 4-2).

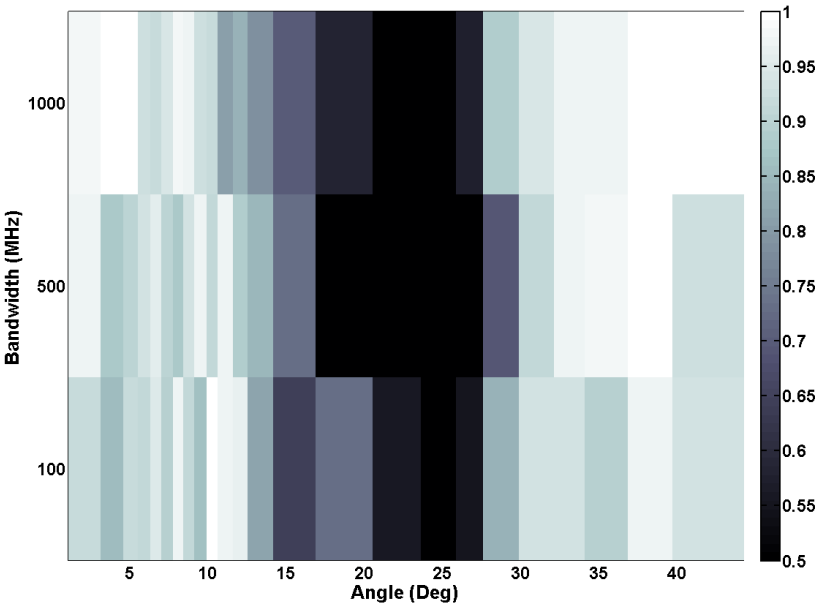

Figure 4-1: System Performance Given Bandwidth and Intersection Angle

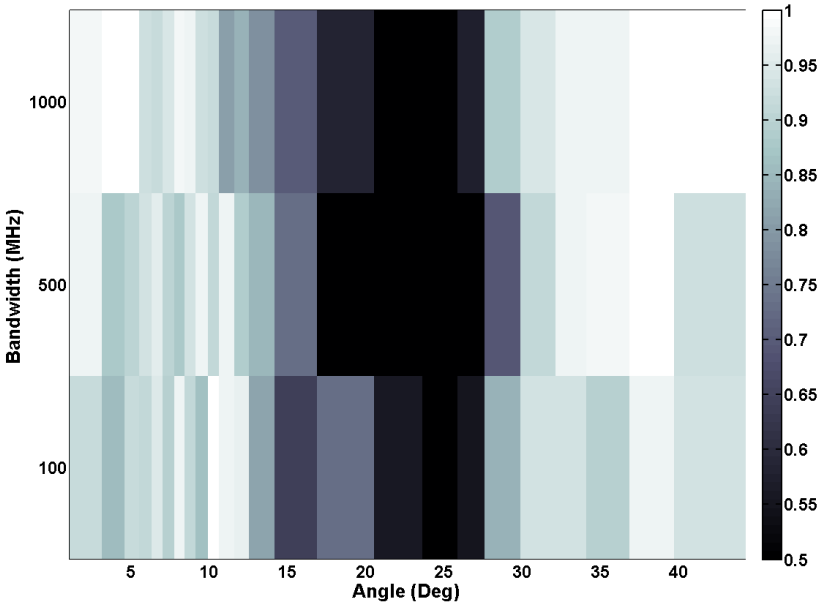

Figure 4-2: Smoothed Curve of System Performance Given Bandwidth and Intersection Angle

Figure 4-2 is a lot more readable. One can clearly see that the probability of correct identification is poorest at the threshold, which is logical because that is the angle where the likelihood of having a split and the likelihood of having a cross are equal. Therefore the probability should be slightly better than a coin flip, assuming that the non-angle rules are able to correctly identify some that the angle rule would have misinterpreted.

As expected, the probability of correct identification approaches unity at very high and low angles of intersection because it unlikely to have a cross at low angles or a split at high angles.

Probabilities of Correct Identification with Respect to Time

The operating curve for time versus angle was constructed a little differently. Our simulation did not have discrete times to choose from as was the case of bandwidth. Since the time distribution was uniform, we had to quantize the y-axis to allow the calculation of probabilities. The unsmoothed and smoothed versions of these curves can be seen in Figure 4-3 and Figure 4-4 respectively.

Again, there is a sharp decrease in probability of correct identification near the angle threshold when there is not sufficient time after the event to interpret the situation.

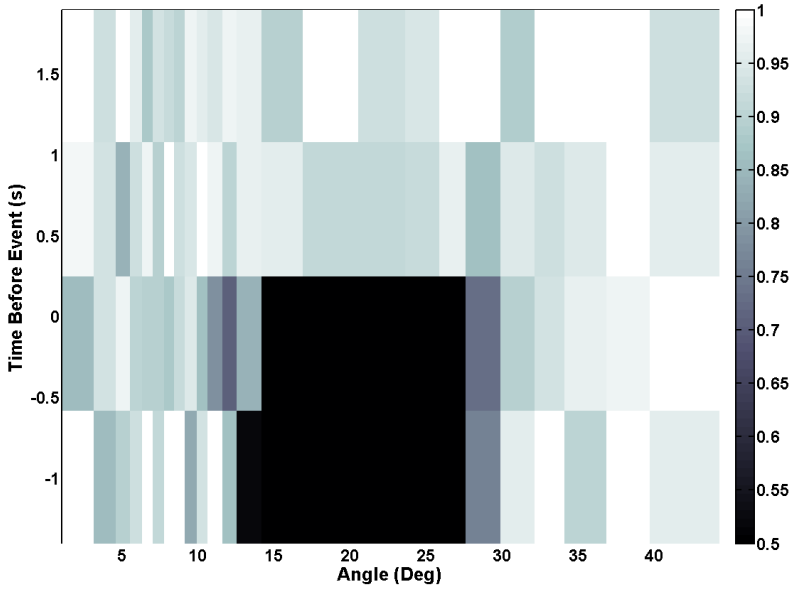

Figure 4-3: System Performance Given Time Before and Intersection Angle

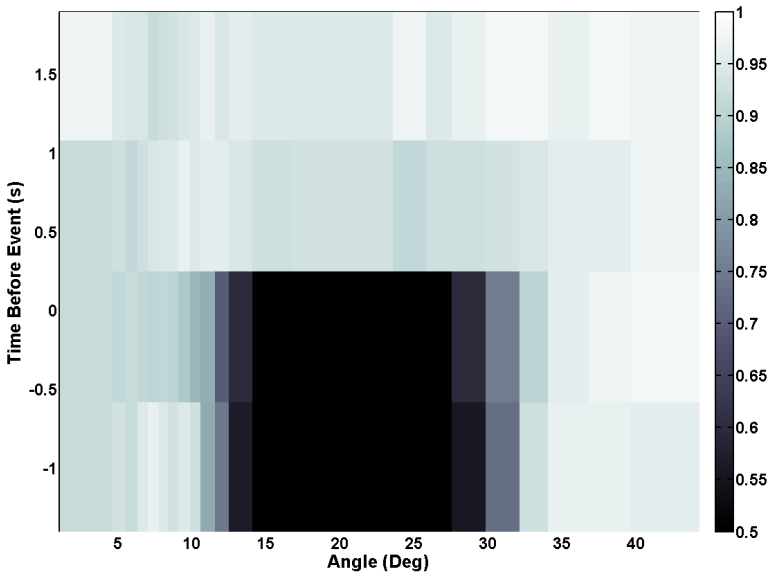

Figure 4-4: Smoothed Performance Given Time Before and Intersection Angle

Threshold Optimization

After building the data set and observing system performance, the group examined what effect varying the threshold would have on the error rates. First, the data set was manipulated to determine the effect the first two rules, Time Before and Width, had on interpretation accuracy. This can be seen in Figure 4-5.

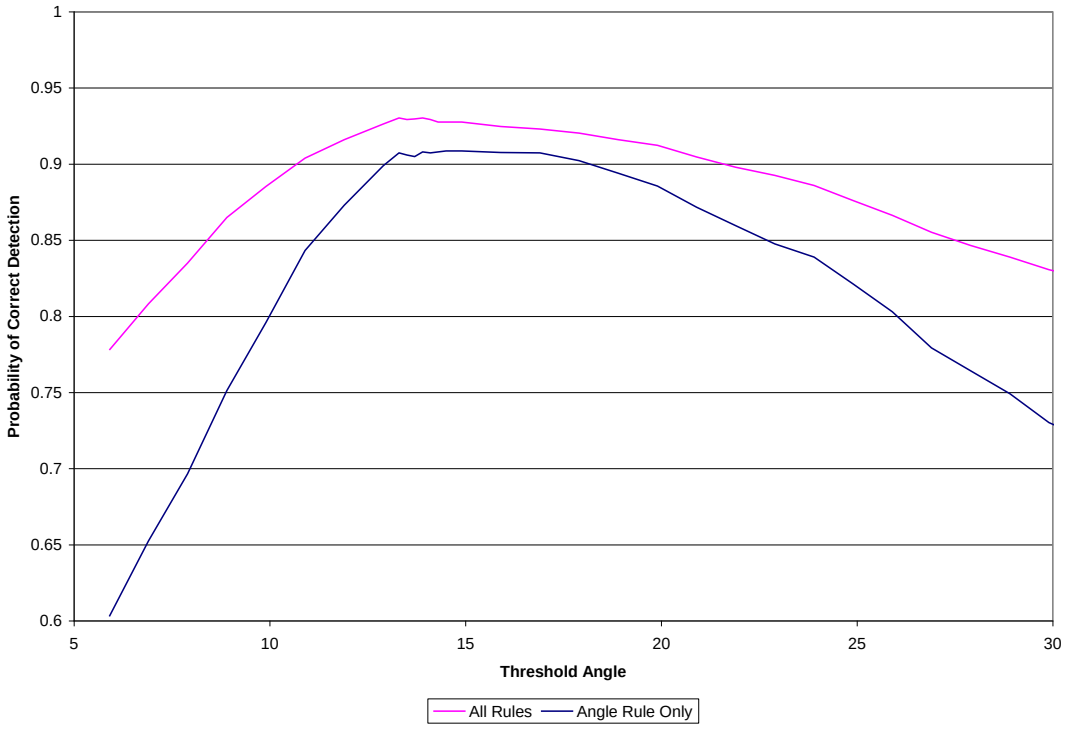

Figure 4-5: Effect of Width and Time Before Rule on System Accuracy

Two important conclusions can be drawn from this result. First, the human decision-making model consistently out-performs an interpretation system that uses only the probabilistic elements. Second, and much more interesting, the threshold for the 'All Rules' curve that yields the highest probability of correct detection, 13.9 degrees, is far from our calculated threshold, 27.9 degrees. To confirm this we examined the components of the error rate, false-cross and false-split, as the threshold was moved. This can be seen in Figure 4-6.

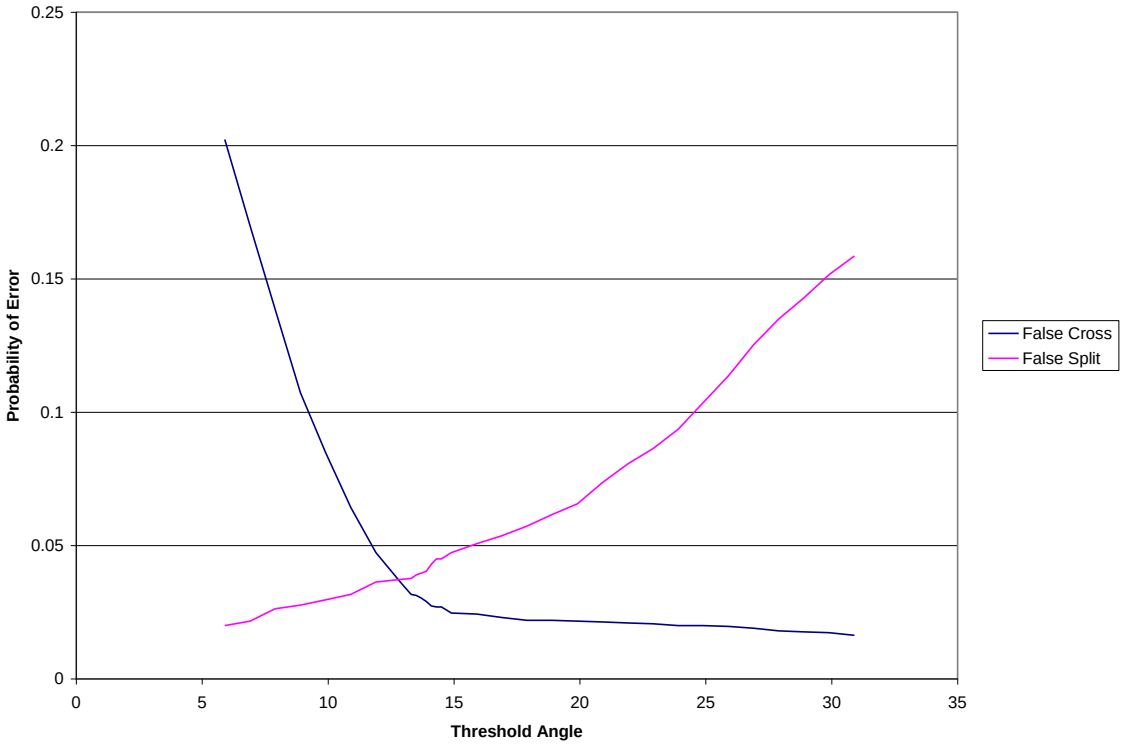

Figure 4-6: Error Rates at Various Thresholds

This second analysis confirms the results of the first: the ideal threshold is different from the calculated threshold.

Faults with Initial System Performance

Further investigation revealed that the system was performing worse than the theoretical maximum of error at and around the threshold; it was performing worse than random guessing. To determine the cause of this poor performance, we examined the distribution of the apparent speed on the RTIs we observed and compared it to the distribution used to generate the RTIs. These can be seen in Figure 4-7 and Figure 4-8, respectively.

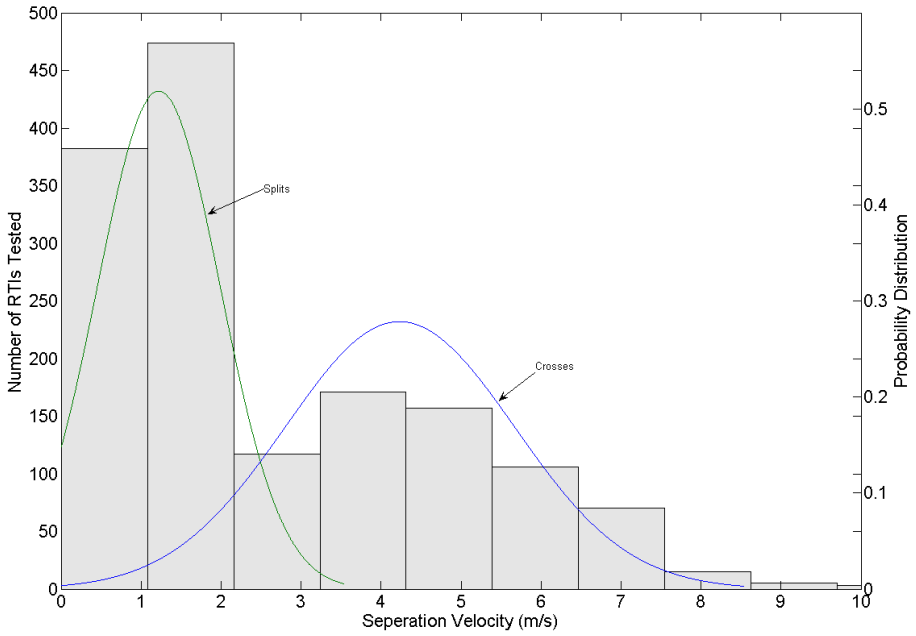

Figure 4-7: Distribution of Sample Velocities with Fitted Normal Curves

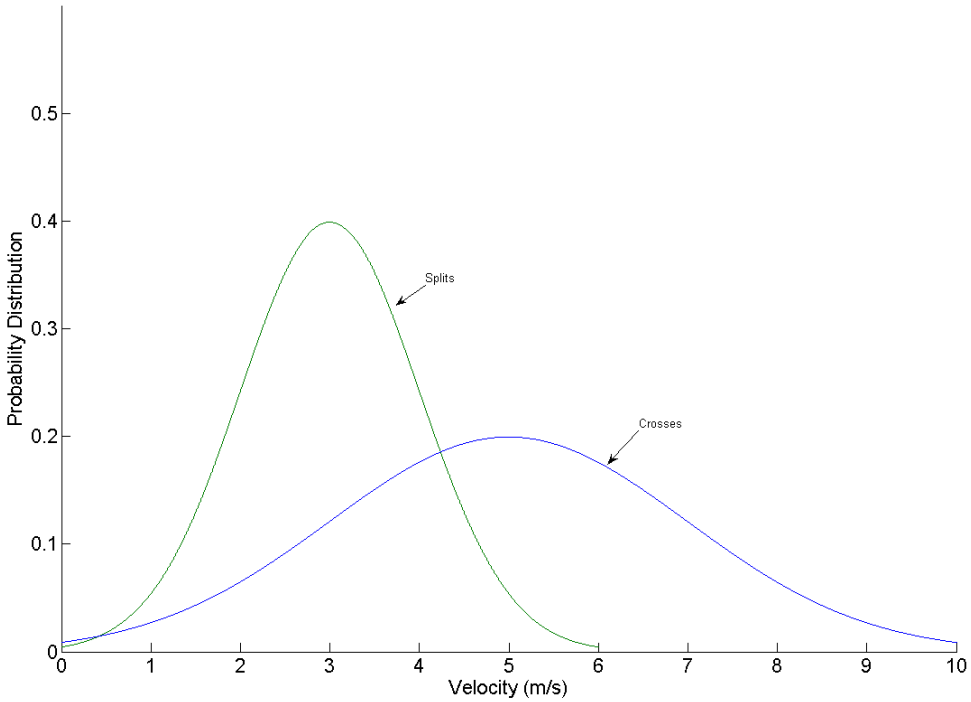

Figure 4-8: Distribution of Velocities for RTI Generation

The distribution for the actual data is significantly different than we expected. The statistics for the actual and expected data are shown in Table 4-4.

Data	Split Mean	Split Standard Deviation	Cross Mean	Cross Standard Deviation
Expected	3.000	1.000	5.000	2.000
Actual	1.222	.769	4.235	1.4328

Table 4-4: Statistics for Expected and Actual Data

The distributions for both crosses and splits in the actual data have lower means and standard deviations than the distribution used to create the RTIs. This explains the optimal threshold being much lower than the calculated threshold.

Effect of Viewing Geometry on Threshold

The cause of this error is due to the difference between the real and apparent relative distances of the objects. For example, see Figure 4-10. d_{A-B} represents the actual distance between the two observed objects. d_A and d_B are the measured distances of the two objects. From the radar's point of view, the relative distance is d_B-d_A, which is less than d_{A-B}.

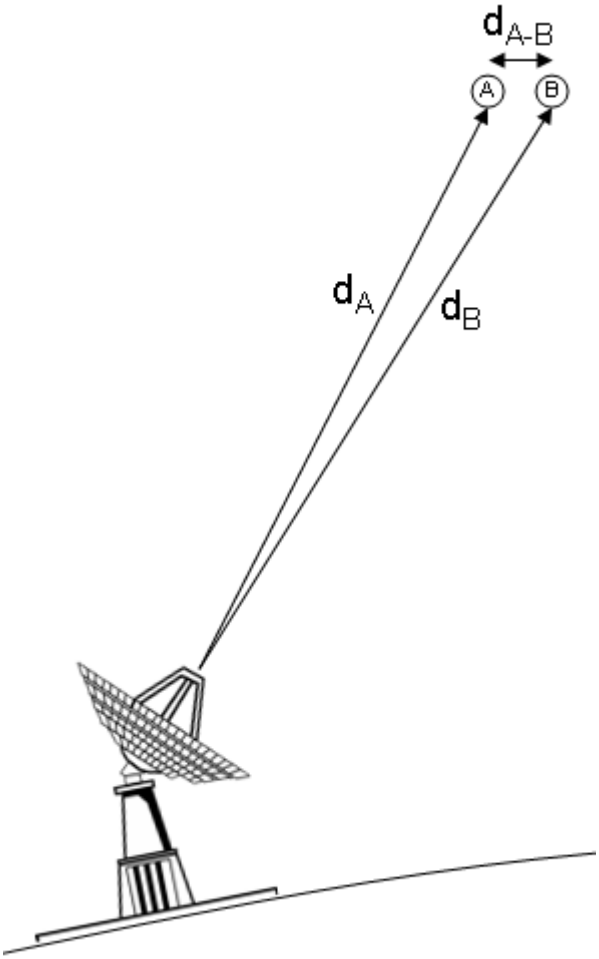

Figure 4-9: Effect of Viewing Geometry on Measured Relative Distance

The degree to which this affects the observations depends on the physical position of the objects. If the objects are collinear with the bore sight of the radar, the measured relative distance is equal to the actual relative distance. If the second object is not located along the same line as the first point, but in the same plane (the plane being formed by the elevation sweep of the radar), the relative distance is $d_{A-B}\cos\theta$, as shown in Figure 4-10.

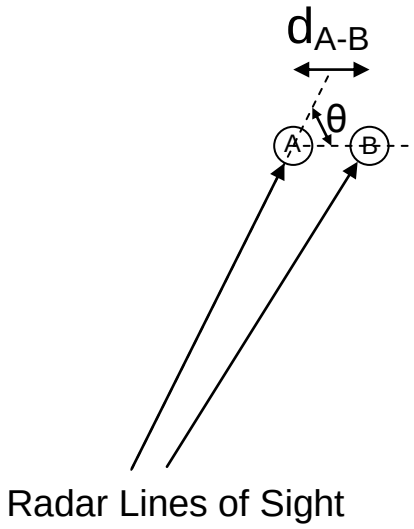

Figure 4-10: Relative Distance Error

When the two objects are not coplanar – as is likely the case in most of the RTIs we observed – the relationship between physical distance and apparent distance becomes more complicated, including the difference in azimuth that the radar must sweep to see both objects. The end result of these viewing geometry issues is that the velocities we have observed have been consistently smaller than those expected, but the relationship between the actual and expected velocities is not linear.

Threshold Modification

Given the nonlinear relationship between the parameters used to generate the scenario and the observed values, the simplest way to determine the new threshold was to find the value which gave us the lowest error. The error rates for all rules and for the angle rule only are presented in Figure 4-11.

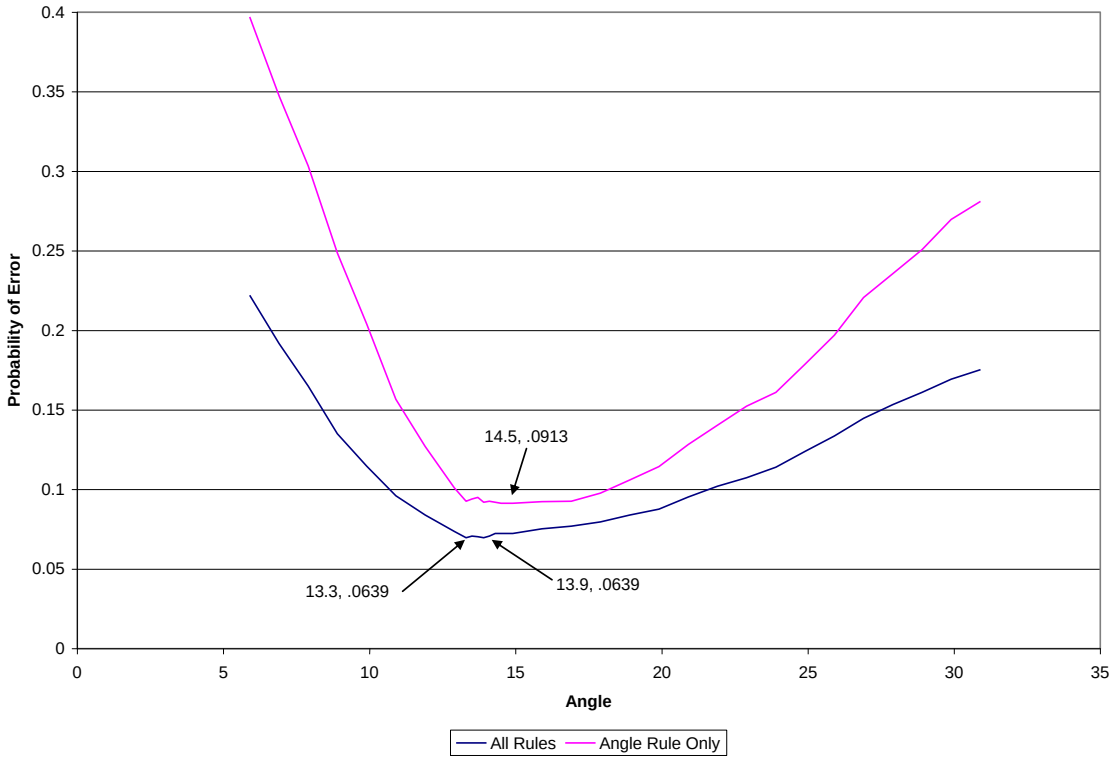

Figure 4-11: Identification Error Rate with Possible Thresholds

For our revised threshold we chose 13.9 degrees, as it gives the lowest error rate using all rules and is also close to the angle rule only threshold.

Revised System Performance

Given the revised threshold, we recalculated the operating curves. Exploring the effect of angle on the probability of correct identification, shown in Figure 4-12, we see performance nearly always over 90%, with an expected sharp decrease about the threshold.

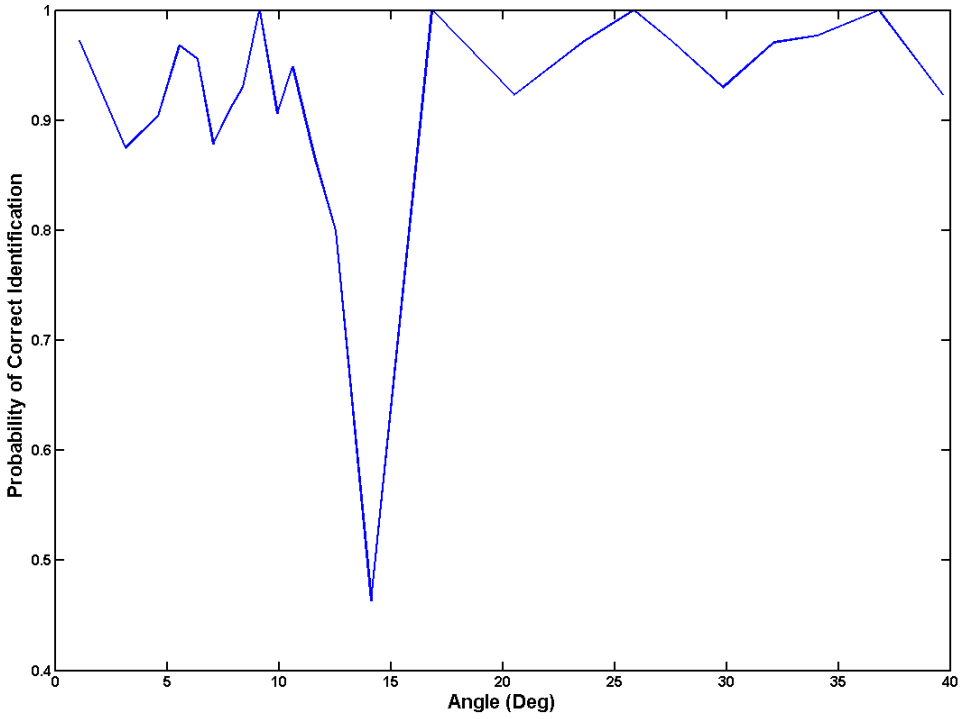

Figure 4-12: Effect of Angle on System Performance

Revised Probabilities of Correct Identification with Respect to Bandwidth

The operating curves presenting system performance with the revised threshold show significant improvements in probability of correct RTI interpretation over those using the original calculated threshold. In Figure 4-13, which displays performance at various bandwidths and intersection angles, system performance is consistently high (over 90% correct identifications) at all bandwidths when the angle is greater than 24°. Performance is also high, though more bandwidth dependant, at lower angles (below 10°). In between 10° and 24°, close to the threshold, performance dips reaching a minimum of approximately 65% correct identifications.

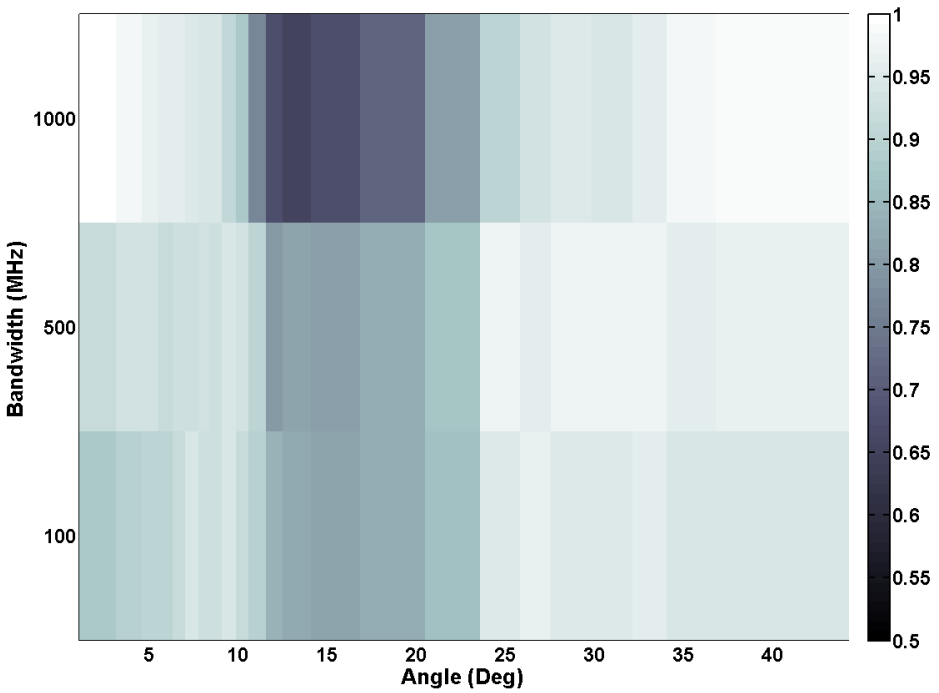

Figure 4-13: Smoothed Curve of System Performance Given Bandwidth and Intersection Angle with Revised Threshold

Revised Probabilities of Correct Identification with Respect to Time

The operating curve presented in Figure 4-14, which displays system performance versus time before the event and intersection angle with the revised threshold, exhibits similar improvements to those detailed in section 0. The new threshold has the band of poor performance transferred to the angle band of 10° to 24°. However, even in this angle band, performance is consistently higher than it was close to the threshold in the original operating curves.

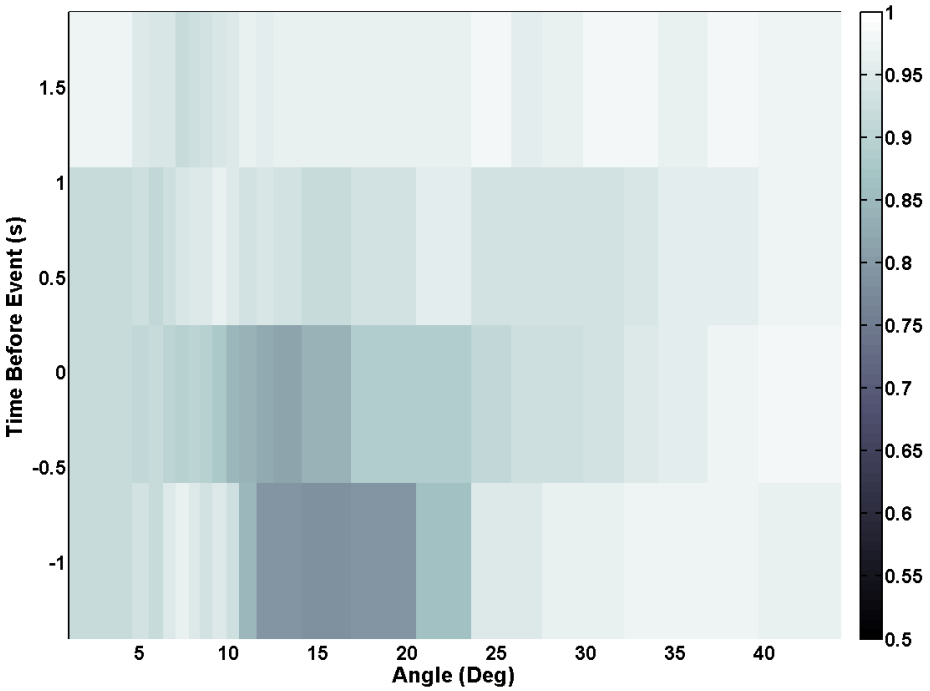

Figure 4-14: Smoothed Curve of System Performance Given Time Before Event and Intersection Angle with Revised Threshold

5. Conclusions and Future Recommendations

The evaluation of our human decision-making model demonstrates that it enhanced our ability to correctly interpret crosses and splits with limited information. By implementing a decision-making model mixing both deterministic and probabilistic rules, we consistently out-performed a solely probabilistic interpretation method. Our analysis found that probability of correct detection was highest when the bandwidth was large, the time before the event was substantial, and the relative velocities of the objects was extreme (either very high or very low). Human error had a small effect on our end results, affecting fewer than 3% of interpretations.

The conclusion of our research leaves open several avenues for additional learning. One variable we would have liked to examine was viewing geometry. Variations in viewing geometry can greatly affect the resulting RTIs, though its effect on our ability to correctly discriminate between splits and crosses is unknown. Another possible follow-up to our research would be to examine scenarios of higher complexity, such as those using multiple objects following more complex motions. The research would be further improved were it conducted using parameters – cross and split velocities, especially - derived from real data rather than best guesses. Finally, the research could be extended by focusing on either the human or machine elements. Research into the effect of the human element could be done by examining error rates across various operators and attempting to minimize or even eliminate human error; for the machine element, it could be through development of a fully automated algorithm for performing cross/split discrimination.

6. Works Cited

Amoozegar, Farid, Vahraz Jamnejad, Timothy Pham, and Robert Cesarone. "Trends in Development of Broad-Band Phased Arrays for Space Applications". *IEEE Explore*. Volume 3-1414. Paper 1335. Accessed September 18, 2007. <http://ieeexplore.ieee.org/iel5/8735/27685/01235257.pdf?arnumber=1235257>

Carpenter, M., and D. Cebula. "MATLAB Radar Scene Generation Toolbox". September 30, 2004.

Hawley, John K., Anna L. Mares, and Cheryl A. Giammanco. "The Human Side of Automation: Lessons for Air Defense Command and Control". March 2005.

Iamaio, Nathaniel. "LL6D V2.4: User Manual". November 2, 2004.

Kaplan, Dr. Laurence M. "Missile Defense: The First Sixty Years". Missile Defense Agency. 27 September 2006. Accessed September 10, 2007. <http://www.mda.mil/mdalink/pdf/first60.pdf>

Missile Defense Agency (MDA). "History of Ballistic Missile Defense." <http://www.mda.mil/mdalink/html/history.html>

Raytheon. Products and Services. < http://www.raytheon.com/products/cobra_dane/> Accessed September 10, 2007.

Skolnik, Merrill L. Introduction to Radar Systems 3rd. New York, NY: Tata McGraw-Hill, 2001.

Spence, Lee. Private conversations.

Taylor, John R. Classical Mechanics. Sausalito, CA: University Science Books, 2005.

Toomay, J.C., and Paul J. Hannen. Radar Principles for the Non-specialist 3rd. Raleigh, NC: SciTech Publishing Inc, 2004.

United States Department of Defense."History of the Missile Defense Organization". Accessed September 18, 2007. <http://www.defenselink.mil/specials/missiledefense/history.html>

Weiner, Stephen D. Private conversations.

Weiner, Stephen D and Sol M. Rocklin. "Discrimination Performance Requirements for Ballistic Missile Defense." *The Lincoln Laboratory Journal*. 7.1. (1994)

Werrel, Kenneth P. "Hitting a Bullet with a Bullet: A History of Ballistic Missile Defense." College of Aerospace Doctrine, Research and Education. Air University. Research Paper 2000-02. (2000)

Appendix A: Scatterer Distance Clarification

As previously mentioned, the range in an RTI is measured with respect to the center of rotation of the object. In Figure A-1 both scatterers and the center of rotation appear to be the same distance away, *d'*, from the radar when in fact the scatterers are slightly further away at a distance *d*.

Since d' is much larger than *s*, the separation between the centres of the spheres at either end of the dumbbell, the difference between *d'* and *d* is negligible. The relationship between these two distances is derived using the Pythagorean Theorem in Equation A- 1.

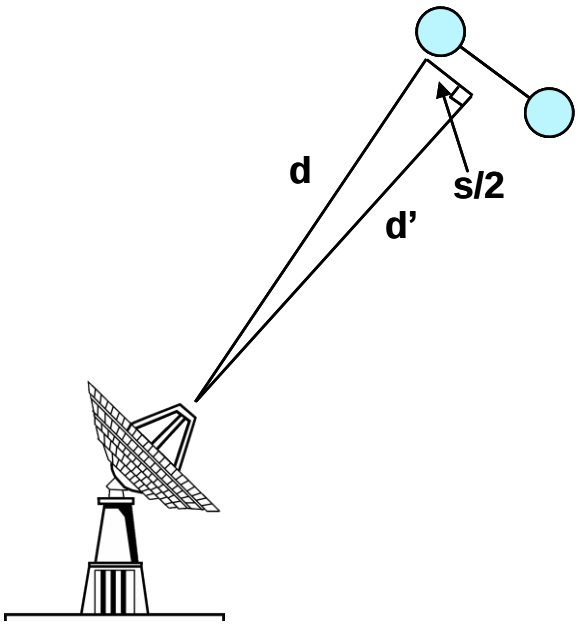

Figure A- 1: Distances from Radar to Object

$$d = \sqrt{d'^2 + \left(\frac{s}{2}\right)^2}$$

A- 1

For example, for a dumbbell with s = 4m and d' =100,000m the difference between d and d' is $2 \cdot 10^{-5}$meters using Equation A- 2.

$$d - d' = \sqrt{\left(10^5\right)^2 + \left(\frac{4}{2}\right)^2} - 100 = 2 \cdot 10^{-5} m$$

A- 2

Appendix B: Reentry Vehicle XML Code

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<RFmodel>
      <Format Version="3.0"/>
      <Name>Basic RV</Name>
      <Background>
            <Author>Cebula</Author>
            <Version>1.0</Version>
            <Date>15 October 2002</Date>
            <Verification>Eyeball comparison to S/X band static
patterns</Verification>
            <Description>This is a model of a simple cone using a
collection of point scatterers</Description>
      </Background>
      <Band>S C X</Band>
      <Requirements>
            pointResponse
      </Requirements>
      <Center>
            <Position Axial="0.0" OffAxis="0.0" Roll="0.0"/>
      </Center>
      <Components>
            <Group Name="nose" Hidden="false">
                  <Component Class="mitll.rcssim.BasicScatterer"
ID="1">
                        <Type>nose</Type>
                        <Response>pointResponse</Response>
                        <Comment>Simple nose scatterer @ -
20dBsm</Comment>
                        <PP>0.1 0.0</PP>
                        <OP>0.001 0.001</OP>
                        <Position Axial="1.0" OffAxis="0.0"/>
                        <Radius>0.05</Radius>
                        <Aspects>0 100 7</Aspects>
                        <Rolls>0 360 7</Rolls>
                  </Component>
            </Group>
            <Group Name="specular" Hidden="false">
                  <Component Class="mitll.rcssim.BasicScatterer"
ID="2a">
                        <Type>slipping</Type>
                        <Response>pointResponse</Response>
                        <Comment>Narrow, strong peak specular</Comment>
                        <PP>10.0 0.0</PP>
                        <OP>5.0 0.01</OP>
                        <Position Axial="0.0" OffAxis="0.33"/>
                        <Aspects>70.3 70.5 0.3</Aspects>
                        <Rolls>0 360 0.3</Rolls>
                  </Component>
                  <Component Class="mitll.rcssim.BasicScatterer"
ID="2b">
                        <Type>slipping</Type>
                        <Response>pointResponse</Response>
```

```
                        <PP>0.5 0.0</PP>
                        <OP>0.6 0.1</OP>
                        <Position Axial="0.0" OffAxis="0.33"/>
                        <Aspects>68 76 3.2</Aspects>
                        <Rolls>0 360 3.2</Rolls>
                  </Component>
            </Group>
            <Group Name="base" Hidden="false">
                  <Component Class="mitll.rcssim.BasicScatterer"
ID="3">
                        <Type>slipping</Type>
                        <Response>pointResponse</Response>
                        <PP>1.0 0.0</PP>
                        <OP>0.5 0.0</OP>
                        <Position Axial="-0.8" OffAxis="0.5"/>
                        <Aspects>150 180 3</Aspects>
                        <Rolls>0 360 3</Rolls>
                  </Component>
                  <Component Class="mitll.rcssim.BasicScatterer"
ID="4">
                        <Type>backslipping</Type>
                        <Response>pointResponse</Response>
                        <PP>0.1 0.0</PP>
                        <OP>0.05 0.0</OP>
                        <Position Axial="-0.8" OffAxis="0.5"/>
                        <Aspects>150 180 3</Aspects>
                        <Rolls>0 360 3</Rolls>
                  </Component>
                  <Component Class="mitll.rcssim.BasicScatterer"
ID="5">
                        <Type>fixed</Type>
                        <Response>pointResponse</Response>
                        <PP>0.3 0.1</PP>
                        <OP>0.1 0.1</OP>
                        <Position Axial="-0.7" OffAxis="0.2"
Roll="30"/>
                        <Aspects>130 180 5</Aspects>
                        <Rolls>0 360 5</Rolls>
                  </Component>
                  <Component Class="mitll.rcssim.BasicScatterer"
ID="6">
                        <Type>slipping</Type>
                        <Response>pointResponse</Response>
                        <PP>0.04 0</PP>
                        <OP>0.03 0</OP>
                        <Position Axial="-0.8" OffAxis="0.5"/>
                        <Aspects>0 130 7</Aspects>
                        <Rolls>0 360 7</Rolls>
                  </Component>
                  <Component Class="mitll.rcssim.BasicScatterer"
ID="2a">
                        <Type>slipping</Type>
                        <Response>pointResponse</Response>
                        <Comment>Narrow, strong peak specular</Comment>
                        <PP>10.0 0.0</PP>
                        <OP>5.0 0.01</OP>
                        <Position Axial="-0.7" OffAxis="0.0"/>
```

```
                        <Aspects>179.7 180.0 0.3</Aspects>
                        <Rolls>0 360 0.3</Rolls>
                    </Component>
                </Group>
            </Components>
</RFmodel>
```

Appendix C: Sample Configuration File

```
#
# Model1Test1718.cfg
#
VerbosityLevel = Silent
Verbosity Stream =
BinaryOutput = false
DragEffectsOn = true
WriteBirthSamples = true
WriteDeathSamples = true
PrintEventSummary = true
GravityModel = GravityJGM3 4 4
FileNameSuffix =
TrajWriteForObjects = all
TrajectoryFileDir = TrajectoryFiles

_OBJECTS
1   RV1          11.0     56.0      56.0        0.234 0.0   0.0
0.001 700
2   RV2          11.0     56.0      56.0        0.234 0.0   0.0
0.001 700
_END_OBJECTS

#

_EVENTS
1 DataRate 0.0 0.1
2 SetState 1 0.0 6678137.0 0.0 0.0 0.0 0.0 2000.0
3 SetState 2 0.0 6678137.0 3.458 0.0 0.0 -4.211 2000.0
4 SetRates 1 0 0 1.817 0
5 SetRates 2 10 0 1.098 0
_END_EVENTS
```

Appendix D: Sample Record File

1. Split DepTime: 79.3408 DepVel: 2.56744 Angle: 7.77827 SepVel: 1.09657 Bwdth: 1000
2. Split-V DepTime: 115.967 DepVel: -1.33442 Angle: 3.58323 SepVel: 0.467833 Bwdth: 100
3. SplitTumble DepTime: 27.5259 DepVel: 3.12533 Tmbl1: 0.770449 Tmbl2: 1.64502 Angle: 8.80514
   SepVel: 1.40003 Bwdth: 100
4. Split DepTime: 45.8459 DepVel: 3.28768 Angle: 12.5155 SepVel: 1.36135 Bwdth: 100
5. CrossTumbleR VelY: 7.29294 DistY: -6.3058 Tmbl1: 1.09779 Tmbl2: 0.48503 Angle: 39.0288
   SepVel: 6.14787 Bwdth: 100
6. Split-V DepTime: 85.3657 DepVel: -4.19092 Angle: 13.7193 SepVel: 1.54913 Bwdth: 1000
7. SplitTumble DepTime: 59.3594 DepVel: 4.18916 Tmbl1: 1.12918 Tmbl2: 0.386659 Angle: 3.09961
   SepVel: 0.353462 Bwdth: 1000
8. SplitTumble DepTime: 136.97 DepVel: 2.96237 Tmbl1: 1.6611 Tmbl2: 1.32533 Angle: 5.40343
   SepVel: 0.834971 Bwdth: 1000
9. CrossTumbleR VelY: 4.34542 DistY: -9.64083 Tmbl1: 0.687651 Tmbl2: 1.69314 Angle: 26.9434
   SepVel: 3.47531 Bwdth: 1000
10. CrossTumbleL VelY: -4.65072 DistY: 10.077 Tmbll: 1.61016 Tmbl2: 1.918 Angle: 26.1443
    SepVel: 4.12691 Bwdth: 500
11. SplitTumble-V DepTime: 133.846 DepVel: -2.81329 Tmbl1: 1.50088 Tmbl2: 0.359386 Angle: 8.62391
    SepVel: 1.0555 Bwdth: 500
12. SplitTumble-V DepTime: 80.9992 DepVel: -3.72579 Tmbl1: 0.223084 Tmbl2: 1.97784 Angle: 14.2975
    SepVel: 1.60956 Bwdth: 100
13. SplitTumble DepTime: 95.3749 DepVel: 2.41168 Tmbl1: 1.54469 Tmbl2: 1.10666 Angle: 8.65242
    SepVel: 1.15797 Bwdth: 500
14. Split DepTime: 93.8177 DepVel: 5.18319 Angle: 15.0427 SepVel: 1.97677 Bwdth: 500
15. CrossL VelY: -5.27279 DistY: 13.3668 Angle: 29.656SepVel: 4.56688 Bwdth: 1000
16. Split-V DepTime: 84.7805 DepVel: -3.11393 Angle: 8.53249 SepVel: 1.36479 Bwdth: 100
17. CrossTumbleR VelY: 2.86646 DistY: -9.15216 Tmbl1: 0.6091 Tmbl2: 0.935856 Angle: 18.0116
    SepVel: 2.36862 Bwdth: 100

Appendix E: MATLAB Code Used For RTI Generation

E.1 RandRTI.m

```
function RandRTI(numIters,recordName,isUniform,clearFiles)

if nargin==2
    isUniform = 0;
    clearFiles = 0;
end

if nargin==3
    clearFiles = 0;
end

tic

cd('C:\Program Files\MATLAB\R2006b\work\');

addpath([pwd '\RFSIG2\']);
addpath([pwd '\RFSIG2\src\']);

mkdir([pwd '\RFSIG2\output\MQP\' recordName '\'])

recfid=fopen([recordName '.txt'],'wt');

numTemplates = 8;
iter = 1;

if clearFiles
    delete([pwd '\RFSIG2\output\MQP\' recordName '\*.jpg']);
end

%%% Template Variables
%% All Cases
BandwidthVals = [100,500,1000];% 100 500 1000
BandwidthLs = [100,1;500,.2;1000,.1];
TimePre = [-1.5,1];
TimePost = [5,15];
SensorPosLat = [0,0];
SensorPosLong = [5, 5];
sensorPos = zeros(2,1);
%% Case 1: MQPCross (L)
C1_Name = 'CrossL';
C1_LIM_VelY = [-6,-.1]; %-.275,-.1
C1_LIM_DistY = [2, 15]; %5,15
C1_LIM_VelY_N = [-5,2];
%% Case 2: MQPSplit
C2_Name = 'Split';
C2_LIM_DepTime = [20, 150];
C2_LIM_DepVel = [.1, 3]; %.1,1
C2_LIM_DepVel_N = [3,1];
%% Case 3: MQPCrossTumble (L)
C3_Name = 'CrossTumbleL';
```

```
C3_LIM_VelY = C1_LIM_VelY;
C3_LIM_VelY_N = C1_LIM_VelY_N;
C3_LIM_DistY = C1_LIM_DistY;
C3_LIM_Tumble = [.1, 2];
%% Case 4: MQPSplitTumble
C4_Name = 'SplitTumble';
C4_LIM_DepTime = C2_LIM_DepTime;
C4_LIM_DepVel = C2_LIM_DepVel;
C4_LIM_DepVel_N = C2_LIM_DepVel_N;
C4_LIM_Tumble = C3_LIM_Tumble;
%% Case 5: MQPCross (R)
C5_Name = 'CrossR';
C5_LIM_VelY = [.1,6]; %.1,.275
C5_LIM_VelY_N = -1.*C1_LIM_VelY_N;
C5_LIM_DistY = [-30, -2]; %-15,-5
%% Case 6: MQPCrossTumble (R)
C6_Name = 'CrossTumbleR';
C6_LIM_VelY = C5_LIM_VelY;
C6_LIM_VelY_N = -1.*C1_LIM_VelY_N;
C6_LIM_DistY = C5_LIM_DistY;
C6_LIM_Tumble = C3_LIM_Tumble;
%% Case 7: MQPSplit (-V)
C7_Name = 'Split-V';
C7_LIM_DepTime = C2_LIM_DepTime;
C7_LIM_DepVel = [-3,-.1]; %-1,-.1
C7_LIM_DepVel_N = -1*C2_LIM_DepVel_N;
%% Case 8: MQPSplitTumble (-V)
C8_Name = 'SplitTumble-V';
C8_LIM_DepTime = C2_LIM_DepTime;
C8_LIM_DepVel = C7_LIM_DepVel;
C8_LIM_DepVel_N = -1*C2_LIM_DepVel_N;
C8_LIM_Tumble = C3_LIM_Tumble;
%% Main Loop
while iter <= numIters

    cfgFile = [recordName num2str(iter)];

    rTemp = ceil(length(BandwidthVals)*rand(1));
    Bandwidth = BandwidthVals(rTemp);

    rTemp = ceil(numTemplates*rand(1));
    writeCFGHeader(cfgFile);

    %interceptAngle = 0;

%    rTemp = [1, 1, 2, 2];
%    rTemp = rTemp(iter);

    tOffStart = randbound(TimePre);
    tOffEnd = randbound(TimePost);

    rScale = 80 * ((tOffEnd)/10)

    sensorPos(1) = randbound(SensorPosLat);
    sensorPos(2) = randbound(SensorPosLong);

    switch rTemp
```

```
        case{1} %Cross
            if(isUniform)
                C1_VelY = randbound(C1_LIM_VelY);
            else
                C1_VelY = randgauss(C1_LIM_VelY_N);
            end
            C1_DistY = randbound(C1_LIM_DistY);
            intersectTime = abs(C1_DistY / C1_VelY);
            writeEventsCross(cfgFile,C1_DistY,C1_VelY);
            [interceptAngle,seperationVel] =
ll6dMATLABnMQP(intersectTime,cfgFile,[tOffStart,tOffEnd],sensorPos,rSca
le);
            fprintf(recfid,'%g. %s \t\t%s:    % 6.6g \t%s:  % 6.6g
\t%s: % 6.6g\t%s: % 6.6g\t%s: %
6.6g\n',iter,C1_Name,'VelY',C1_VelY,'DistY',C1_DistY,'Angle',interceptA
ngle,'SepVel',seperationVel,'Bwdth',Bandwidth);
        case{2} %Split
            if(isUniform)
                C2_DepVel = randbound(C2_LIM_DepVel);
            else
                C2_DepVel = randgauss(C2_LIM_DepVel_N);
            end
            C2_DepTime = randbound(C2_LIM_DepTime);
            intersectTime = C2_DepTime;
            writeEventsSplit(cfgFile,C2_DepTime,C2_DepVel);
            [interceptAngle,seperationVel] =
ll6dMATLABnMQP(C2_DepTime,cfgFile,[tOffStart,tOffEnd],sensorPos,rScale)
;
            fprintf(recfid,'%g. %s \t\t%s: % 6.6g \t%s: % 6.6g \t%s: %
6.6g\t%s: % 6.6g\t%s: %
6.6g\n',iter,C2_Name,'DepTime',C2_DepTime,'DepVel',C2_DepVel,'Angle',in
terceptAngle,'SepVel',seperationVel,'Bwdth',Bandwidth);
        case{3} %Cross w/ Tumble
            if(isUniform)
                C3_VelY = randbound(C3_LIM_VelY);
            else
                C3_VelY = randgauss(C3_LIM_VelY_N);
            end
            C3_DistY = randbound(C3_LIM_DistY);
            intersectTime = abs(C3_DistY / C3_VelY);
            C3_Tumble1 = randbound(C3_LIM_Tumble);
            C3_Tumble2 = randbound(C3_LIM_Tumble);

writeEventsCrossTumble(cfgFile,C3_DistY,C3_VelY,C3_Tumble1,C3_Tumble2);
            [interceptAngle,seperationVel] =
ll6dMATLABnMQP(intersectTime,cfgFile,[tOffStart,tOffEnd],sensorPos,rSca
le);
            fprintf(recfid,'%g. %s \t%s:    % 6.6g \t%s:  % 6.6g \t%s:
% 6.6g \t%s: % 6.6g\t%s: % 6.6g\t%s: % 6.6g\t%s: %
6.6g\n',iter,C3_Name,'VelY',C3_VelY,'DistY',C3_DistY,'Tmbll',C3_Tumble1
,'Tmbl2',C3_Tumble2,'Angle',interceptAngle,'SepVel',seperationVel,'Bwdt
h',Bandwidth);
        case{4} %Split w/ Tumble
            C4_DepTime = randbound(C4_LIM_DepTime);
            if(isUniform)
                C4_DepVel = randbound(C4_LIM_DepVel);
            else
```

```
                C4_DepVel = randgauss(C4_LIM_DepVel_N);
            end
            intersectTime = C4_DepTime;
            C4_Tumble1 = randbound(C4_LIM_Tumble);
            C4_Tumble2 = randbound(C4_LIM_Tumble);

writeEventsSplitTumble(cfgFile,C4_DepTime,C4_DepVel,C4_Tumble1,C4_Tumbl
e2);
            [interceptAngle,seperationVel] =
ll6dMATLABnMQP(C4_DepTime,cfgFile,[tOffStart,tOffEnd],sensorPos,rScale)
;
            fprintf(recfid,'%g. %s \t%s: % 6.6g \t%s: % 6.6g \t%s: %
6.6g \t%s: % 6.6g\t%s: % 6.6g\t%s: % 6.6g\t%s: %
6.6g\n',iter,C4_Name,'DepTime',C4_DepTime,'DepVel',C4_DepVel,'Tmbl1',C4
_Tumble1,'Tmbl2',C4_Tumble2,'Angle',interceptAngle,'SepVel',seperationV
el,'Bwdth',Bandwidth);
        case{5} %Cross
            if(isUniform)
                C5_VelY = randbound(C5_LIM_VelY);
            else
                C5_VelY = randgauss(C5_LIM_VelY_N);
            end
            C5_DistY = randbound(C5_LIM_DistY);
            intersectTime = abs(C5_DistY / C5_VelY);
            writeEventsCross(cfgFile,C5_DistY,C5_VelY);
            [interceptAngle,seperationVel] =
ll6dMATLABnMQP(intersectTime,cfgFile,[tOffStart,tOffEnd],sensorPos,rSca
le);
            fprintf(recfid,'%g. %s \t\t%s:    % 6.6g \t%s:  % 6.6g
\t%s: % 6.6g\t%s: % 6.6g\t%s: %
6.6g\n',iter,C5_Name,'VelY',C5_VelY,'DistY',C5_DistY,'Angle',interceptA
ngle,'SepVel',seperationVel,'Bwdth',Bandwidth);
        case{6} %Cross w/ Tumble
            if(isUniform)
                C6_VelY = randbound(C6_LIM_VelY);
            else
                C6_VelY = randgauss(C6_LIM_VelY_N);
            end
            C6_DistY = randbound(C6_LIM_DistY);
            intersectTime = abs(C6_DistY / C6_VelY);
            C6_Tumble1 = randbound(C6_LIM_Tumble);
            C6_Tumble2 = randbound(C6_LIM_Tumble);

writeEventsCrossTumble(cfgFile,C6_DistY,C6_VelY,C6_Tumble1,C6_Tumble2);
            [interceptAngle,seperationVel] =
ll6dMATLABnMQP(intersectTime,cfgFile,[tOffStart,tOffEnd],sensorPos,rSca
le);
            fprintf(recfid,'%g. %s \t%s:    % 6.6g \t%s:  % 6.6g \t%s:
% 6.6g \t%s: % 6.6g\t%s: % 6.6g\t%s: % 6.6g\t%s: %
6.6g\n',iter,C6_Name,'VelY',C6_VelY,'DistY',C6_DistY,'Tmbl1',C6_Tumble1
,'Tmbl2',C6_Tumble2,'Angle',interceptAngle,'SepVel',seperationVel,'Bwdt
h',Bandwidth);
        case{7} %Split % Intersect time = DepTime
            C7_DepTime = randbound(C7_LIM_DepTime);
            if(isUniform)
                C7_DepVel = randbound(C7_LIM_DepVel);
            else
```

```
                C7_DepVel = randgauss(C7_LIM_DepVel_N);
            end
            intersectTime = C7_DepTime;
            writeEventsSplit(cfgFile,C7_DepTime,C7_DepVel);
            [interceptAngle,seperationVel] =
ll6dMATLABnMQP(C7_DepTime,cfgFile,[tOffStart,tOffEnd],sensorPos,rScale)
;
            fprintf(recfid,'%g. %s \t\t%s: % 6.6g \t%s: % 6.6g \t%s: %
6.6g\t%s: % 6.6g\t%s: %
6.6g\n',iter,C7_Name,'DepTime',C7_DepTime,'DepVel',C7_DepVel,'Angle',in
terceptAngle,'SepVel',seperationVel,'Bwdth',Bandwidth);
        case{8} %Split w/ Tumble
            C8_DepTime = randbound(C8_LIM_DepTime);
            if(isUniform)
                C8_DepVel = randbound(C8_LIM_DepVel);
            else
                C8_DepVel = randgauss(C8_LIM_DepVel_N);
            end
            intersectTime = C8_DepTime;
            C8_Tumble1 = randbound(C8_LIM_Tumble);
            C8_Tumble2 = randbound(C8_LIM_Tumble);

writeEventsSplitTumble(cfgFile,C8_DepTime,C8_DepVel,C8_Tumble1,C8_Tumbl
e2);
            [interceptAngle,seperationVel] =
ll6dMATLABnMQP(C8_DepTime,cfgFile,[tOffStart,tOffEnd],sensorPos,rScale)
;
            fprintf(recfid,'%g. %s \t%s: % 6.6g \t%s: % 6.6g \t%s: %
6.6g \t%s: % 6.6g\t%s: % 6.6g\t%s: % 6.6g\t%s: %
6.6g\n',iter,C8_Name,'DepTime',C8_DepTime,'DepVel',C8_DepVel,'Tmbl1',C8
_Tumble1,'Tmbl2',C8_Tumble2,'Angle',interceptAngle,'SepVel',seperationV
el,'Bwdth',Bandwidth);
        otherwise
            disp(['Case error, rTemp = ' num2str(rTemp)])
    end

    load('C:\program files\MATLAB\R2006b\work\RFSIG2\MQP.mat');
    guioutput.record = recordName;
    guioutput.ident = cfgFile;
    guioutput.WBinfo.bandwidthMHz = Bandwidth;
    guioutput.time.start = intersectTime + tOffStart;
    if guioutput.time.start < 0
        guioutput.time.start = 0;
    end
    guioutput.time.end = intersectTime + tOffEnd;
    guioutput.WBinfo.maxrange = rScale;
    guioutput.angle = interceptAngle;
    guioutput.BWTable = BandwidthLs;
    guioutput.eventTime = intersectTime;
    save('C:\program
files\MATLAB\R2006b\work\RFSIG2\MQP.mat','guioutput');

    cleansingleMQP;
    cd('C:\Program Files\MATLAB\R2006b\work\');
    iter = iter + 1;

    delete([cfgFile '.cfg']);
```

```
end

toc
```

E.2 ll6dMATLABnMQP.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%
%   UNCLASSIFIED
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%
%   LL6dMATLAB.m
%
%   Basic calculation of random generating threat variations for
analysis
%
%   version 0  :  MATLABClientCode.m  N. Iamaio June 28, 2004
%   version 0.1:  LL6DMATLAB.m  B.Tipton June 29,2004
%        Added more comments and time history functions
%
%   +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
%   BEFORE USING THIS CODE IN MATLAB VERIFY YOU HAVE INSTALLED LL6D
%   CORRECTLY
%    1)  downloaded and installed LL6D libraries according to
README.txt
%    2)  Created a "classpath.txt" file in a directory in which this
%        MATLAB is running.  Inside the classpath.txt are lines with
the
%        paths to all the LL6D libraries.
%    3)  Run MATLAB in the same directory as the classpath.txt
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%
%  I)  Create the trajectories
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%

%
%   I.1 Create flight manager object
%       The second argument is the LL6D threat configuration file
%       The third argument is 0 for write to file or 1 for buffer
output to memory
%       Be sure to watch memory usage when buffering memory,
trajectories can
%       10s of megabytes.
%
%flightMgr = javaObject('timehistory.FlightManager',...
%    'Single.cfg');
```

```
function
[sepAng,sepVel]=ll6dMATLABnMQP(intersectTime,config,timeRange,sensorPos
,rScale)

%Globals
global randGen;
global props;
global jnull;
global sensor;
global NBinfo;
global WBinfo;

%Filename and path definitions
%Program will create and look for trajectory and time history files in
the
%data folder, which must be a subdir of the working directory.
%Config, scatterer file, and APSM file must be in the working
directory.
wkDir                       = 'C:\Program Files\MATLAB\R2006b\work\';
dataDir                     = 'TrajectoryFiles\';
%scenarioName                = 'MQPSplitTumble';
objNames                    = {'RV1','RV2'};
scattererFiles              = {'basicrv.xml','basicrv.xml'}; %should be
an xml file
alignIndex                  = 1; %Index of object to align on
apsmFile                    = 'SimdefXML.properties'; %properties file
name
range_scale                 = 200; %specifies how wide or narrow of a
range you want the rti to cover
gen_files                   = 1;
gen_rti                     = 0;
cfgIsBinary                 = 0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
cfgFile                     = [config '.cfg'];
outputDir                   = 'RFSIG2\trajectories\RandRTI';

%%%%%%%%%%% DEFINE RTI DURATION
rti_time=100:.25:101;

%or if you want an automatically set time window, just uncomment the
%getTimeLimits line before the call to form_rti

num_objects                 = length(objNames);
%arr_ind                     = 1:num_objects;

trajobj                     = cell(num_objects,1);
thf                         = cell(num_objects,1);
trks                        = cell(num_objects,1);

props                       = java.util.Properties;
jnull                       = props.get('Junk');
rfsig_dir                   = '.\';

radar.f0                    = 10;         % Center frequency (GHz)
radar.bw                    = 300;       % Bandwidth (MHz)
```

```
radar.rw                    = range_scale;% Range window (m) for
combined response from all objects
radar.rg                    = .1;        % Range gate spacing (m)
radar.brf                   = 1;          % Burst frequency (hz)
radar.prf                   = 100;        % Instantaneous PRF (Hz)
within each burst
radar.t_burst               = 1;          % Time duration (s) of each
burst
radar.t_offset              = 100;        % Time offset (s) of each
burst
radar.n_p_burst             = radar.prf;  % Total number of bursts in
each pulse
radar.n_p_int               = 1;

WBinfo.responseFilename     = [rfsig_dir,'exresp_taylor.txt'];
WBinfo.respTab              =
javaObject('mitll.rcssim.ResponseTable',WBinfo.responseFilename);
WBinfo.maxrange             = radar.rw;
%WBinfo.snroff               = [-15 -12];
WBinfo.snroff               = [-50 -50];
WBinfo.pulsetimes           = 250;
isWB                        = true;
WBinfo.rangeGateSize        = radar.rg;
WBinfo.bandwidthMHz         = radar.bw;
WBinfo.freqGHz              = radar.f0;
WBinfo.windowDefinition     = 'taylor 40 6';
WBinfo.noiseFloor           = -60;
WBinfo.pol                  = 'PP';
WBinfo.polarization         = WBinfo.pol;

NBinfo.rangeGateSize        = radar.rg;
NBinfo.bandwidthMHz         = radar.bw;
NBinfo.freqGHz              = radar.f0;
NBinfo.polarization         = WBinfo.pol;
NBinfo.windowDefinition     = 'taylor 40 6';

scenario.random_seed        = sum(100*clock);
scenario.random_seed        = 1;

randSeed                    = scenario.random_seed;
randGen                     = javaObject('java.util.Random',randSeed);

if gen_files
    !del/Q TrajectoryFiles
    FILEBUFFERMODE=0;
    flightMgr = javaObject('timehistory.FlightManager',...
        cfgFile,FILEBUFFERMODE);

    flightMgr.runSim;
end

%
% I.5 Retrieve the output.  These are trajectories
%
%      trajarray is an Java array Object of type java.util.Arraylist
%      Retrieve an individual trajectory with the syntax
traj=trajarray.get(#);
```

```
%      Remember Java uses "C" style array indexing.  The first object
has
%       index 0! This is opposed to MATLAB which indexes from 1.

%%%%%%%%%%%%%%%%%%%%%%%%%%% UPDATE FILENAME/FILEPATH TO FIT CURRENT
OBJECT
trajio = mitll.sixd.TrajectoryIO;
iter=1;
if cfgIsBinary
    while(iter<=length(objNames))
        trajobj(iter) = trajio.readBin([wkDir dataDir objNames{iter}
'.bin']);
        iter = iter+1;
    end
else
    while(iter<=length(objNames))
        trajobj(iter) = trajio.readAscii([wkDir dataDir objNames{iter}
'.dat']);
        iter = iter+1;
    end
end



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%
%  II) "TimeHistory" Coversion.
%     Convert the Truth Trajectory in ECI coordinates to sensor/radar
%     reference frame measurements (range, azimuth, elevation).  Also,
%     make basic calculations of radar coverage with respect to
%     Field-of-View and a very basic "noise floor" calculation
%  (  Noise Floor is defined as the negative SNR on a 0dBsm object.
%     e.g. use as SNR in dB = RCS - NoiseFloor
%     where  NoiseFloor= 40log10(Range) - SNR_REFERENCE.  See Below)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%

%
%  II.1 Define the sensor
%
sensor = javaObject('mitll.sixd.SensorInfo');
sensor.m_name='Bogus Radar';
% place radar at latitude, longitude, altitude = 33 deg, 130 deg, 0m
%      (somewhere around Japan?)

%%%%% 'Perfect' radar - 36
sensor.m_location=javaObject('mitll.metric.LatLonAltPoint',
sensorPos(1), sensorPos(2), 0.0 );
sensor.m_elevLow = 1;  % 1 deg min elevation
sensor.m_elevHigh=90;  % 80 deg max elevation
sensor.m_elev0 = 40;   % sensor array, with boresight at 40 deg from
horizontal
sensor.m_elevKValue = 2; % ave. scanloss exponent.  Scanloss(dB)= -
10log10( cos(theta)^K)
sensor.m_azimLow = 0;  % min azimuth
sensor.m_azimHigh = 360;  % max azimuth
```

```
sensor.m_azim0=0;  % sensor array facing north. Note 0/360 wrapping
accounted for.
sensor.m_azimKValue=2; % ave. scanloss exponent for azimuth.
sensor.m_SNR_REF=-350; % factor accounting for sensor power, area etc.
                       % Noise floor is calculated with this offset via
                       %  NoiseFloor=-40log10 (Range) - SNR_REFERENCE
                       % Note:  -350 is for a ridiculously powerful
radar


%
% II.2  Calculate what this sensor sees of the LL6D trajectories
%
traj2th=javaObject('mitll.sixd.Trajectory2TimeHistory');

iter=1;
while(iter<=length(trajobj))
    thf(iter) = javaMethod('generate',traj2th, trajobj{iter}, sensor);
    iter = iter+1;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
iter=1;
thfio = mitll.sixd.TimeHistoryFileIO;
while(iter<=length(thf))
    thfio.writeAscii(thf{iter},[dataDir objNames{iter} '_THF.dat']);
    iter = iter+1;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%
%  III) "GenSig" Coversion.
%     Convert radar observations into a "TrackedObjectLL6dGenSig" class
%     which uses the Augmented Point Scatterer Model (APSM) to produce
%     radar signatures and pulse-by-pulse data products (RTI,DTI,RDI,
etc).
%
%
%  NOTE: Code only works if rfsig/mitll/APSM are distributed in
addition to
%         LL6D
%
%   This section still under construction.  --June 29, 2004 BT
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%

%Copy moves THF files to appropriate place for SIMGUI
copyfile([wkDir dataDir '*thf.dat'],[wkDir outputDir])

props=javaObject('java.util.Properties');
iter = 1;

while(iter<=length(trajobj))
    trks{iter} = create_track([wkDir outputDir '\' objNames{iter}
'_THF.dat'],0,[wkDir scattererFiles{iter}],[wkDir
apsmFile],objNames(iter),objNames(iter),'b--',props);
```

```
    iter = iter+1;
end

sepDist = abs((trks{2}.getRanges(intersectTime+timeRange(2),0)-
trks{1}.getRanges(intersectTime+timeRange(2),0)));
sepAng = atand( (sepDist*1000/rScale) / (timeRange(2)/(timeRange(2)-
timeRange(1))) );% * ((timeRange(2)-timeRange(1))/10) )%/timeRange(2))
sepVel = 1000*sepDist/(timeRange(2)-timeRange(1));
%
%  III.3  Form a range-time-intensity plot
%   (form_rti from rfsig)


%timetemp = trk.getTimeLimits;
%rti_time = timetemp(1):(timetemp(2)-timetemp(1))./1000:timetemp(2);

if gen_rti
    rti_out=form_rti_backup(rti_time,trks,isWB,trks{alignIndex},250,[-
15 -12],radar.rw,true,WBinfo.noiseFloor,0);

% RTI Plotter

    figure;
    plotrti(rti_out.amps,rti_out.r,rti_out.t);

end



'RTI completed.' %#ok<NOPRT>
```

E.3 runsimMQP.m

```
function [] = runsim(guiinput)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%%
%% runsim.m: Driver for RFSIG generation
%%
%% Driver program that generates radar data using APSM scatterer files
and LL6D
%% time history files.
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%% runsim.m: modified version of testsim.m, changed to work with the
GUI
%% as input.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%%
%% $Id: testsim.m,v 1.6 2005/05/25 19:15:16 bkate Exp $
%%
%% AUTHOR: David Cebula
%%         MIT LINCOLN LABORATORY
%%         April 7, 2003
%%
%% Copyright (c) 2005 MIT/Lincoln Laboratory.
%% All Rights Reserved.
%%
%% THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF
%% MIT/Lincoln Laboratory. The copyright notice does
%% not evidence any actual or intended publication of
%% such source code.
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%

javaaddpath([pwd '/jars/mitll_external-3.1.2.jar']);
javaaddpath([pwd '/jars/jnl.jar']);

global DIR_ROOT;
global jnull;
global props;
global NBinfo;
global WBinfo;
global pcparams;
global randGen;

%% create java null
props = javaObject('java.util.Properties');
jnull = props.get('Junk');

%% make a directory root
DIR_ROOT = [pwd '/'];
addpath src;

time = guiinput.time;
ts = time.start:time.step:time.end;

%% Pulls structs out of input var
WBinfo = guiinput.WBinfo;
NBinfo = guiinput.NBinfo;
DTIinfo = guiinput.DTIinfo;
IMinfo = guiinput.IMinfo;
NBIMinfo = guiinput.NBIMinfo;
WBIMinfo = guiinput.WBIMinfo;
pcparams = guiinput.pcparams;
opts = guiinput.opts;
defComplex = guiinput.defComplex;
staticRange = guiinput.staticRange;

record = guiinput.record;
cfgFile = guiinput.ident;
```

```
randSeed = 1;
randGen = javaObject('java.util.Random',randSeed);
props.put('RandomGenerator',randGen);

%% Sets up default file paths and makes neccessary formatting changes
if guiinput.defaultFilepaths
  defComplex.thfloc = [DIR_ROOT 'trajectories/' defComplex.cfgFile];
  defComplex.apsmdefloc = [DIR_ROOT 'simdefAPSM.properties'];
  defComplex.scattdefloc = [DIR_ROOT 'targets/'];
  WBinfo.responseFilename = [DIR_ROOT
'targets/responses/exresp_taylor.txt'];
  opts.saveLoc = regexprep([DIR_ROOT 'output\MQP\' record
'\'],'\','/');
  %% saveLoc character replacement neccessary to avoid errors in
sprintf
else
  defComplex.thfloc = [defComplex.thfloc defComplex.cfgFile];
  opts.saveLoc = regexprep(opts.saveLoc,'\','/');
  %% saveLoc character replacement neccessary to avoid errors in
sprintf
end

WBinfo.respTab =
javaObject('mitll.rcssim.ResponseTable',WBinfo.responseFilename);

%% Generates tracks as mitll.architecture.ITrackedObjects
trks = gen_tracks(defComplex); %
trkAlign = trks{defComplex.trkAlign};


%%%%%%%%%%%%%%%%%%%%%%%%% Computation and Output
%%%%%%%%%%%%%%%%%%%%%%%%

%% open file to facilitate output
if opts.saveEMF
    fout = fopen([opts.saveLoc 'hdrALL.txt'], 'wt');
    fprintf(fout,'%.1f - %.1f\n', min(ts), max(ts));
    fprintf(fout,'define_complex.m runsim.m\n');
end


%% Show tracks
if opts.showRanges

    figure;
    hold on;
    ranges0 = trkAlign.getRanges(ts,0);

    for k=1:length(trks)
        trk = trks{k};
        rs = trk.getRanges(ts,0);
        iix = find(rs > 0);
        plot(rs(iix)-ranges0(iix),ts(iix),char(trk.getPlotStyle));
    end

    xlabel('Range (km)');
    ylabel('Time (s)');
```

```
    fmakep5;

    if opts.saveEMF
        str = ['print -dmeta ' '''' opts.saveLoc '''' 'range.emf'];
        eval(str);
        fprintf(fout,'range.emf Relative Ranges\n');
    end

    if opts.saveIMG
        str = sprintf(['print -d' opts.imgType ' ' '''' opts.saveLoc
'''' '%s_range_%04.3f-%04.3f.' opts.imgXtn], cfgFile,ts(1),
ts(length(ts)))
        eval(str);
    end
end

%% WB RTI - Combo
if opts.showWBRTI

    if opts.returnAmps
        [amps,xs] =
form_rti_F(ts,trks,true,trkAlign,WBinfo.snroff,WBinfo.maxrange,true,WBi
nfo.noiseFloor);
        pows = 20*log10(abs(amps));
    else
        [pows,xs] =
form_rti_F(ts,trks,true,trkAlign,WBinfo.snroff,WBinfo.maxrange,false,WB
info.noiseFloor);
    end

    timeScale = 0:time.step:length(ts);
    timeScale = timeScale.*time.step;
    figure;
    imagesc(xs,timeScale,pows,[-40 30]);
    axis xy;
    axis square;
    colorbar;
    xlabel('Range (m)');
    ylabel('Time (s)');
    fmakep5;

    if opts.saveEMF
        str = ['print -dmeta ' '''' opts.saveLoc ''''
'wb_combo_rti.emf'];
        eval(str);
        fprintf(fout,'wb_combo_rti.emf Wideband RTI\n');
    end

%%MQP Output Vars
angle = guiinput.angle;
Ls = guiinput.BWTable;
obsL =
interp1q(Ls(1:size(Ls,1),1),Ls(1:size(Ls,1),2),WBinfo.bandwidthMHz);
obsTime = quant((obsL/tand(angle))*length(timeScale)/length(xs),.2);
if obsTime == 0
    obsTime = .2;
end
```

```
eventTime = guiinput.eventTime;

    if opts.saveIMG
        str = sprintf(['print -d' opts.imgType ' ' '''' opts.saveLoc
'''' '%s_BW%04.3f_AN%04.3f_OT%04.3f_L%04.3f_TB%04.3f.' opts.imgXtn],
cfgFile, WBinfo.bandwidthMHz, angle, obsTime, obsL, eventTime-
time.start)
        eval(str);
    end
end

pause(1);

%% NB RTI - Combo
if opts.showNBRTI

    if opts.returnAmps
        [amps,xs] =
form_rti_F(ts,trks,false,trkAlign,NBinfo.snroff,NBinfo.maxrange,true,NB
info.noiseFloor);
        pows = 20*log10(abs(amps));
    else
        [pows,xs] =
form_rti_F(ts,trks,false,trkAlign,NBinfo.snroff,NBinfo.maxrange,false,N
Binfo.noiseFloor);
    end

    figure;
    imagesc(xs,ts,pows,[-40 30]);
    axis xy;
    colorbar;
    xlabel('Range (m)');
    ylabel('Time (s)');
    fmakep5;

    if opts.saveEMF
        str = ['print -dmeta ' '''' opts.saveLoc ''''
'nb_combo_rti.emf'];
        eval(str);
        fprintf(fout,'nb_combo_rti.emf Narrowband RTI\n');
    end

    if opts.saveIMG
        str = sprintf(['print -d' opts.imgType ' ' '''' opts.saveLoc
'''' 'nb_combo_rti_%04.3f-%04.3f.' opts.imgXtn], ts(1),
ts(length(ts)));
        eval(str);
    end
end


%% WB RTI - singles
if opts.showWBRTIsing

    for k=1:length(trks)

        trk = trks{k};
```

```
        name = char(trk.getIdentifier);

        if opts.returnAmps
            [amps,xs] =
form_rti_sing(ts,trk,true,WBinfo.snroff,true,WBinfo.noiseFloor);
            pows = 20*log10(abs(amps));
        else
            [pows,xs] =
form_rti_sing(ts,trk,true,WBinfo.snroff,false,WBinfo.noiseFloor);
        end

        figure;
        imagesc(xs,ts,pows,[-40 30]);
        axis xy;
        colorbar;
        xlabel('Range (m)');
        ylabel('Time (s)');
        title(sprintf('Object = %s', name));
        fmakep5;

        if opts.saveEMF
            fnam = sprintf('wb_rti_%s.emf', name);
            str = sprintf(['print -dmeta ' '''' opts.saveLoc ''''
'%s'],fnam);
            eval(str);
            fprintf(fout,'%s Wideband RTI; ID = %s\n', fnam, name);
        end

        if opts.saveIMG
            str = sprintf(['print -d' opts.imgType ' ' ''''
opts.saveLoc '''' 'wb_rti_%s_%04.3f-%04.3f.' opts.imgXtn], name, ts(1),
ts(length(ts)));
            eval(str);
        end
    end
end


%% NB RTI - singles
if opts.showNBRTIsing

    for k=1:length(trks)

        trk = trks{k};
        name = char(trk.getIdentifier);
        if opts.returnAmps
            amps =
form_rti_sing(ts,trk,false,NBinfo.snroff,true,NBinfo.noiseFloor);
            pows = 20*log10(abs(amps));
        else
            pows =
form_rti_sing(ts,trk,false,NBinfo.snroff,false,NBinfo.noiseFloor);
        end

        figure;
        plot(ts,pows,char(trk.getPlotStyle));
        axis xy;
```

```
        colorbar;
        ylabel('RCS (dBsm)');
        xlabel('Time (s)');
        title(sprintf('Object = %s', name));
        fmakep5;

        if opts.saveEMF
            fnam = sprintf('nb_rti_%s.emf', name);
            str = sprintf(['print -dmeta ' '''' opts.saveLoc ''''
'%s'],fnam);
            eval(str);
            fprintf(fout,'%s Narrowband RCS History; ID = %s\n', fnam,
name);
        end

        if opts.saveIMG
            str = sprintf(['print -d' opts.imgType ' ' ''''
opts.saveLoc '''' 'nb_rti_%s_%04.3f-%04.3f.' opts.imgXtn], name, ts(1),
ts(length(ts)));
            eval(str);
        end
    end
end


%% Combined DTI
if opts.showDTI

    if opts.returnAmps
        [amps,fs] =
form_dti_mult(ts,trks,trkAlign,DTIinfo.prf,DTIinfo.npuls,DTIinfo.snroff
,DTIinfo.maxrange,true,DTIinfo.noiseFloor);
        pows = 20*log10(abs(amps));
    else
        [pows,fs] =
form_dti_mult(ts,trks,trkAlign,DTIinfo.prf,DTIinfo.npuls,DTIinfo.snroff
,DTIinfo.maxrange,false,DTIinfo.noiseFloor);
    end

    figure;
    imagesc(fs,ts,pows,[-40 30]);
    axis xy;
    colorbar;
    xlabel('Frequency (Hz)');
    ylabel('Time (s)');
    fmakep5;

    if opts.saveEMF
        str = ['print -dmeta ' '''' opts.saveLoc '''' 'dti_combo.emf'];
        eval(str);
        fprintf(fout,'dti_combo.emf Combined DTI\n');
    end

    if opts.saveIMG
        str = sprintf(['print -d' opts.imgType ' ' '''' opts.saveLoc
'''' 'dti_combo_%04.3f-%04.3f.' opts.imgXtn], ts(1), ts(length(ts)));
        eval(str);
```

```
    end
end


%% DTI - singles
if opts.showDTIsing

    for k=1:length(trks)

        trk = trks{k};
        name = char(trk.getIdentifier);

        if opts.returnAmps
            [amps,fs] =
form_dti_sing(ts,trk,DTIinfo.prf,DTIinfo.npuls,DTIinfo.snroff,true,DTIi
nfo.noiseFloor);
            pows = 20*log10(abs(amps));
        else
            [pows,fs] =
form_dti_sing(ts,trk,DTIinfo.prf,DTIinfo.npuls,DTIinfo.snroff,false,DTI
info.noiseFloor);
        end

        figure;
        imagesc(fs,ts,pows,[-40 30]);
        axis xy;
        colorbar;
        xlabel('Frequency (Hz)');
        ylabel('Time (s)');
        title(sprintf('Object = %s', name));
        fmakep5;

        if opts.saveEMF
            fnam = sprintf('dti_%s.emf', name);
            str = sprintf(['print -dmeta ' '''' opts.saveLoc ''''
'%s'],fnam);
            eval(str);
            fprintf(fout,'%s DTI; ID = %s\n', fnam, name);
        end

        if opts.saveIMG
            str = sprintf(['print -d' opts.imgType ' ' ''''
opts.saveLoc '''' 'dti_%s_%04.3f-%04.3f.' opts.imgXtn], name, ts(1),
ts(length(ts)));
            eval(str);
        end
    end
end


%% Single range doppler images
if opts.showSingRDImages

    for itrk = 1:length(trks);

        if (IMinfo.makeplot ~= -99)
```

```
            iix = find(itrk == IMinfo.makeplot);

            if length(iix) < 1
                continue;
            end
        end

        trk = trks{itrk};
        name = char(trk.getIdentifier);

        if opts.saveAVI
            fname = sprintf([opts.saveLoc 'rd_img_%s_%04.3f-
%04.3f.avi'], name, ts(1), ts(length(ts)));
            aviobj = avifile(fname,'FPS',10);
            aviobj.quality = 100;
        end

        for k=1:length(ts)

            t = ts(k);
            imparms = trk.getImageParameters(t, 0.05, 1.0);
            IMinfo.prf = imparms(1);

            if (IMinfo.prf < 20)
                IMinfo.prf = 20;
            end

            IMinfo.npuls = ceil(imparms(2)*IMinfo.prf);
            IMinfo.snroff = 90 - 10*log10(IMinfo.npuls);
            asp = imparms(3);

            if opts.returnAmps
                [amps,xs,fs] =
form_image_sing(t,trk,IMinfo.prf,IMinfo.npuls,IMinfo.snroff,true,IMinfo
.noiseFloor);
                pows = 20*log10(abs(amps));
            else
                [pows,xs,fs] =
form_image_sing(t,trk,IMinfo.prf,IMinfo.npuls,IMinfo.snroff,false,IMinf
o.noiseFloor);
            end

            fig = figure(99);
            set(fig,'DoubleBuffer','on');
            imagesc(fs,xs,pows,[-40 30]);
            axis xy;
            colorbar;
            xlabel('Frequency (Hz)');
            ylabel('Range (m)');
            vv = axis;
            str = sprintf('PRF = %.0f Hz; N =
%d',IMinfo.prf,IMinfo.npuls);
            ss=text(0.98*vv(1)+0.02*vv(2),0.95*vv(3)+0.05*vv(4),str);
            set(ss,'Color',[1 1 1]);
            title(sprintf('Object = %s; Time = %.3f; Aspect =
%.0f',char(trk.getIdentifier),t,asp));
            fmakep5;
```

```
            if opts.saveIMG
                str = sprintf(['print -d' opts.imgType ' ' '''' 
opts.saveLoc '''' 'rd_img_%s_%04.3f.' opts.imgXtn], name, t);
                eval(str);
            else
                pause(0.1);
            end

            if opts.saveAVI
                set(fig,'Color',[1 1 1]);
                    aviobj = addframe(aviobj,fig);
            end
        end

        if opts.saveAVI
            aviobj = close(aviobj);
        end
    end
end


%% Combined NB images
if opts.showMultNBImages

    if opts.saveAVI
        fname = sprintf([opts.saveLoc 'nb_combo_img_%04.3f-
%04.3f.avi'], ts(1), ts(length(ts)));
        aviobj = avifile(fname,'FPS',10);
        aviobj.quality = 100;
    end

    for k=1:length(ts)

        t = ts(k);

        if opts.returnAmps
            [amps,xs,fs] =
form_image_mult_F(t,trks,false,trkAlign,NBIMinfo.prf,NBIMinfo.npuls,NBI
Minfo.snroff,NBIMinfo.maxrange,true,NBIMinfo.noiseFloor);
            pows = 20*log10(abs(amps));
        else
            [pows,xs,fs] =
form_image_mult_F(t,trks,false,trkAlign,NBIMinfo.prf,NBIMinfo.npuls,NBI
Minfo.snroff,NBIMinfo.maxrange,false,NBIMinfo.noiseFloor);
        end

        fig = figure(100);
        set(fig,'DoubleBuffer','on');
        imagesc(fs,xs,pows,[-40 30]);
        axis xy;
        colorbar;
        xlabel('Frequency (Hz)');
        ylabel('Range (m)');
        title(sprintf('Time = %.3f',t));
        fmakep5;
```

```
        if opts.saveIMG
          str = sprintf(['print -d' opts.imgType ' ' '''' opts.saveLoc
'''' 'nb_combo_img_%04.3f.' opts.imgXtn], t);
          eval(str);
        else
            pause(0.1);
        end

        if opts.saveAVI
            set(fig,'Color',[1 1 1]);
            aviobj = addframe(aviobj,fig);
        end
    end

    if opts.saveAVI
        aviobj = close(aviobj);
    end
end


%% Combined WB images
if opts.showMultWBImages

    if opts.saveAVI
        fname = sprintf([opts.saveLoc 'wb_combo_img_%04.3f-
%04.3f.avi'], ts(1), ts(length(ts)));
        aviobj = avifile(fname,'FPS',10);
        aviobj.quality = 100;
    end

    for k=1:length(ts)

        t = ts(k);

        if opts.returnAmps
            [amps,xs,fs] =
form_image_mult_F(t,trks,true,trkAlign,WBIMinfo.prf,WBIMinfo.npuls,WBIM
info.snroff,WBIMinfo.maxrange,true,WBIMinfo.noiseFloor);
            pows = 20*log10(abs(amps));
        else
            [pows,xs,fs] =
form_image_mult_F(t,trks,true,trkAlign,WBIMinfo.prf,WBIMinfo.npuls,WBIM
info.snroff,WBIMinfo.maxrange,false,WBIMinfo.noiseFloor);
        end

        fig = figure(101);
        set(fig,'DoubleBuffer','on');
        imagesc(fs,xs,pows,[-40 30]);
        axis xy;
        colorbar;
        xlabel('Frequency (Hz)');
        ylabel('Range (m)');
        title(sprintf('Time = %.3f',t));
        fmakep5;

       if opts.saveIMG
```

```
          str = sprintf(['print -d' opts.imgType ' ' '''' opts.saveLoc
'''' 'wb_combo_img_%04.3f.' opts.imgXtn], t);
          eval(str);
        else
            pause(0.1);
        end

        if opts.saveAVI
            set(fig,'Color',[1 1 1]);
            aviobj = addframe(aviobj,fig);
        end
    end

    if opts.saveAVI
        aviobj = close(aviobj);
    end
end

if opts.saveEMF
    fclose(fout);
end

%% Static Range RTI - singles
if opts.showStaticRangeRTI


staticRangeTHF(time.start,time.step,time.end,staticRange.range,staticRa
nge.aspect,staticRange.orientation,staticRange.period,staticRange.noise
Floor)

    for k=1:length(trks)

        tempComplex = struct('thfloc',[pwd
'\sim_gui\settings\'],'apsmdefloc',defComplex.apsmdefloc,'scattdefloc',
defComplex.scattdefloc,'identList',{defComplex.identList(k)},'graphList
',{defComplex.graphList(k)},'typeList',{defComplex.typeList(k)},'scattL
ist',{defComplex.scattList(k)},'isBinary',0,'cfgFile','','sufext','_THF
.dat');
        tempComplex.objNameList=cellstr('staticrange');

        trktemp = gen_tracks(tempComplex);
        trk = trktemp{1};

        name = char(trk.getIdentifier);

        if opts.returnAmps
            [amps,xs] =
form_rti_sing(ts,trk,true,staticRange.snroff,true,staticRange.noiseFloo
r);
            pows = 20*log10(abs(amps));
        else
            [pows,xs] =
form_rti_sing(ts,trk,true,staticRange.snroff,false,staticRange.noiseFlo
or);
        end

        figure;
```

```
        imagesc(xs,ts,pows,[-40 30]);
        axis xy;
        colorbar;
        xlabel('Range (m)');
        ylabel('Time (s)');
        title(sprintf('Object = %s', name));
        fmakep5;

        if opts.saveEMF
            fnam = sprintf('wb_rti_%s.emf', name);
            str = sprintf(['print -dmeta ' '''' opts.saveLoc ''''
'%s'],fnam);
            eval(str);
            fprintf(fout,'%s Static Range RTI; ID = %s\n', fnam, name);
        end

        if opts.saveIMG
            str = sprintf(['print -d' opts.imgType ' ' ''''
opts.saveLoc '''' 'sr_rti_%s_%04.3f-%04.3f.' opts.imgXtn], name, ts(1),
ts(length(ts)));
            eval(str);
        end
    end
end

%msgbox('Simulation complete.');

% $Log: testsim.m,v $
% Revision 1.6  2005/05/25 19:15:16  bkate
% updated method for loading classes and libraries
%
% Revision 1.5  2005/05/25 17:13:09  bkate
% made MATLAB 6 friendly
%
% Revision 1.4  2005/05/25 11:38:19  bkate
% modified path to mitll jar
%
% Revision 1.3  2005/05/24 19:47:56  bkate
% updated for re-distribution
%
% Revision 1.2  2004/08/04 19:41:22  npiamaio
% added and addpath command to specifiy the src dir.  These files
should not
% be modified.  They serve as an example as to how to call teh code.
If new
% client code is written and the author thinks it useful for others
check
% those in.
%
% Revision 1.1  2004/08/04 19:10:38  npiamaio
% initial
%
```

Appendix F: MATLAB Operating Curve Code Example

```
%%% Operating Curve: Bandwidth, Revised Threshold
xlsFile = 'C:\Documents and Settings\chris\My Documents\final.xls';
samples = 120;

bwvals = [100 500 1000];
tbquants = 4;

[angles,blah1,blah2]=xlsread(xlsFile,'Raw Data','C2:C5000');
%#ok<NASGU>
[bwith,blah1,blah2]=xlsread(xlsFile,'Raw Data','D2:D5000'); %#ok<NASGU>
[tbefore,blah1,blah2]=xlsread(xlsFile,'Raw Data','E2:E5000');
%#ok<NASGU>
[correct,blah1,blah2]=xlsread(xlsFile,'Angle2','I2:I5000');

clear blah1
clear blah2

data=[angles bwith tbefore correct];
data=sortrows(data);
%[angles,i]=sort(angles);
ind = 1:samples:size(data,1);
ind = ind(1:end-1);
limits = ((data(ind+samples/2,1)));
limits(end)=limits(end)+1;

j=1;
bwout = cell(3,length(limits)-1);
sampSize = zeros(3,length(limits)-1);
while j<=length(bwvals)
    i=1;
    bwtest = bwvals(j);
    a=find(data(1:end,2)==bwtest);
    while i<=length(limits)-1
        b=find(limits(i)<=data(a,1) & data(a,1)<limits(i+1));
        bwout{j,i}=data(a(b),1:4);
        sampSize(j,i)=length(b);
        i=i+1;
    end
    j=j+1;
end

bwlevel = zeros(3,100*length(limits)-1);
bwlevel2 = zeros(3,length(limits)-1);
j=1;
while j<=length(bwvals)
    i=1;
    loc = 1;
    while i<=length(limits)-1
%         k=1;
%         while k<=size(bwout{j,i},1);
        k=size(bwout{j,i},1);
        scale = floor((limits(i+1)-limits(i))*100);
```

```
        if k>0
            bwlevel(j,loc:loc+scale)=sum(bwout{j,i}(1:k,4))/k;
            bwlevel2(j,i)=sum(bwout{j,i}(1:k,4))/k;
            loc=loc+scale+1;
        end
        i=i+1;
    end
    j=j+1;
end

load OperatingCurveColormap;

i=1;
finalout = zeros(length(bwvals),100*length(limits)-1);
movAve = 2;
while i<=length(bwvals)
    j=1;
    loc = 1;
    while j<=length(limits)-1
        a=j-movAve;
        b=j+movAve;
        scale = floor((limits(j+1)-limits(j))*100);
        if a<=0
           a=1;
        end
        if b>length(bwlevel2)
            b=length(bwlevel2);
        end
        finalout(i,loc:loc+scale) = mean(bwlevel2(i,a:b));
        j=j+1;
        loc = loc + scale;
    end
    i=i+1;
end


figure
hold on;
imagesc(limits,1:length(bwvals),bwlevel,[.5,1]);
colormap(mycmap);
colorbar;
axis([limits(1) limits(end)-.5 .5 length(bwvals)+.5]);
set(gca,'ytick',[1 2 3],'yticklabel',{'100';'500';'1000'})
xlabel('Angle (Deg)','FontSize',16);
ylabel('Bandwidth (MHz)','FontSize',16);
fmakep5;
hold off;

figure
hold on;
imagesc(limits,1:length(bwvals),bwlevel,[.75,1]);
colormap(mycmap);
colorbar;
axis([limits(1) limits(end)-.5 .5 length(bwvals)+.5]);
set(gca,'ytick',[1 2 3],'yticklabel',{'100';'500';'1000'})
xlabel('Angle (Deg)','FontSize',16);
```

```
ylabel('Bandwidth (MHz)','FontSize',16);
fmakep5;
hold off;

figure
hold on;
imagesc(limits,1:length(bwvals),finalout,[.5,1]);
colormap(mycmap);
colorbar;
axis([limits(1) limits(end)-.5 .5 length(bwvals)+.5]);
set(gca,'ytick',[1 2 3],'yticklabel',{'100';'500';'1000'})
xlabel('Angle (Deg)','FontSize',16);
ylabel('Bandwidth (MHz)','FontSize',16);
fmakep5;
hold off;

figure
hold on;
imagesc(limits,1:length(bwvals),finalout,[.75,1]);
colormap(mycmap);
colorbar;
axis([limits(1) limits(end)-.5 .5 length(bwvals)+.5]);
set(gca,'ytick',[1 2 3],'yticklabel',{'100';'500';'1000'})
xlabel('Angle (Deg)','FontSize',16);
ylabel('Bandwidth (MHz)','FontSize',16);
fmakep5;
hold off;
```

Distribution Statement

A. Approved for public release; distribution is unlimited.