Target Breakup Detection in the Multiple Hypothesis Tracking Formulation

by

Maurice Kyojin Chu

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degrees of

Bachelor of Science in Electrical Science and Engineering

and

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1996

Author .....
Department of Electrical Engineering and Computer Science
May 17, 1996

Certified by .....
Sanjoy K. Mitter
Professor
Thesis Supervisor

Certified by ...
Keh-Ping Dunn
Group Leader, Group 42 Systems Testing and Analysis
Thesis Supervisor

Accepted by...
F. R. Morgenthaler
Chairman, Departmental Committee on Graduate Theses

Target Breakup Detection in the Multiple Hypothesis Tracking Formulation

by

Maurice Kyojin Chu

Submitted to the Department of Electrical Engineering and Computer Science
on May 17, 1996, in partial fulfillment of the
requirements for the degrees of
Bachelor of Science in Electrical Science and Engineering
and
Master of Engineering in Electrical Engineering and Computer Science

Abstract

The multiple hypothesis tracking (MHT) formulation has been extended to include target breakup detection. Target breakup detection entails finding the time of breakup of a target and tracking its breakup pieces. Incorporating this into the MHT formulation requires modifying the score function used for ranking hypotheses. This problem eventually turns into one of identifying the statistical model generating the data in each scan, and Akaike's criterion was found to be relevant to solving this problem. Thus, the application of Akaike's criterion to target breakup detection in a multiple hypothesis tracker is formulated.

Applications of a target breakup incorporated MHT formulation lie in ballistic missile defense where the offense may intentionally break up a reentry vehicle (RV) or where the defense intercepts RV's.

Thesis Supervisor: Sanjoy K. Mitter
Title: Professor

Thesis Supervisor: Keh-Ping Dunn
Title: Group Leader, Group 32 Systems Testing and Analysis

Acknowledgments

I would like to thank Professor Sanjoy Mitter and Dr. Keh-Ping Dunn for their help and support in this thesis and Dr. Ming-Jer Tsai for bringing the multiple hypothesis tracking problem with target breakup detection to my attention. I also especially thank my parents whose undying faith in my abilities has allowed me to achieve this stage of my life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Multiple target tracking is an interesting problem theoretically due to the inherent data association problem not found in single target tracking. However, research for the sake of interest alone cannot justify a theory's value because it is the theory's utility to practical applications that gives it value. Multiple target tracking has many applications in both military and civilian areas. Regarding military applications, multiple target trackers are used in *ballistic missile defense* to track potentially dangerous reentry vehicles and in *air defense* to track enemy aircraft. One important civilian application is *air traffic control.*

The major contribution of this thesis is the incorporation of *target breakup detection* into the multiple hypothesis tracking formulation. The usefulness of detecting target breakups is limited by its applications. The applications to which the work of this thesis applies lie in ballistic missile defense. All targets that are tracked are assumed to be of the ballistic type which are constrained to follow the dynamic equations of motion due to gravity and which do not exhibit any maneuvers. Thus, all targets are assumed to be passive without actuators, such as rockets, etc. With these assumptions, target breakup comes in two scenarios:

1. **Offensive Breakup** The offense (i.e. the party that launched the missile or, generically, the reentry vehicle (RV)) intentionally breaks up the RV into several pieces to complicate the defense's task in executing a counterattack as

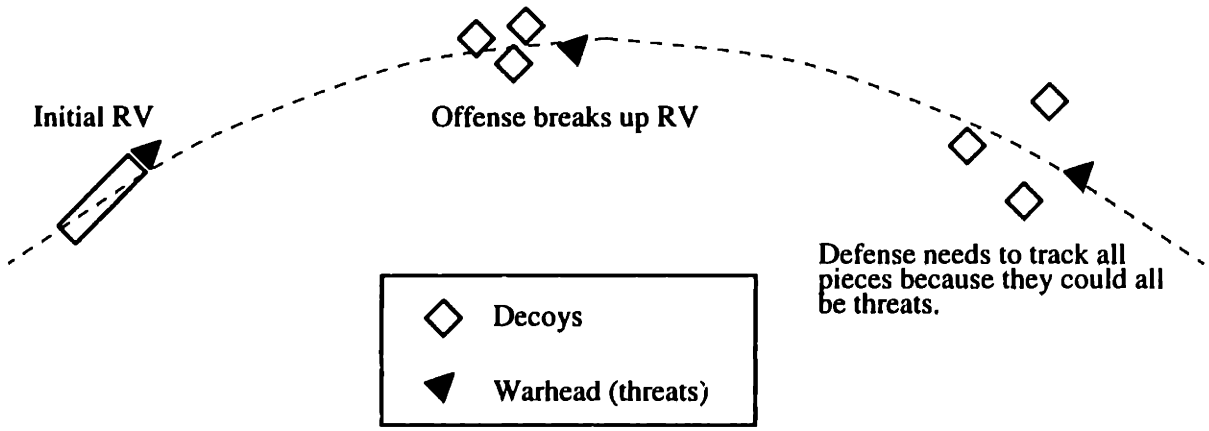


Figure 1-1: Offensive Breakup

in Figure 1-1. The defense is interested in tracking each of these breakup pieces because one or several of them may contain warheads. Furthermore, if the defense knew that only one out of several RV's was a threat before any breakup occurred, the ability to identify from which target the breakup pieces originated would help the defense's counterattack.

2. **Defensive Interception** The defense has intercepted the offense's RV, and assuming that the RV was hit, the RV should be blown into hundreds of pieces. Tracking these pieces seems superfluous since they are no longer a threat; however, it is not always the case that a hit to the RV is a successful interception because the tail of the RV may have been hit, and the warhead, which is commonly found in the nose of the RV, is still a threat (See Figure 1-2). Tracking the breakup pieces of the RV then becomes important.

Target breakup detection is essentially the identification of the origin of a track or, in other words, the determination of whether or not a track originated from the breakup of some existing track. Therefore, a good understanding of multiple target tracking is essential. Chapter 2 describes the problem of multiple target tracking and several approaches to solving the problem and defines much of terminology used in proceeding chapters. In particular, this chapter will focus on describing and discussing the multiple hypothesis tracking (MHT) approach.

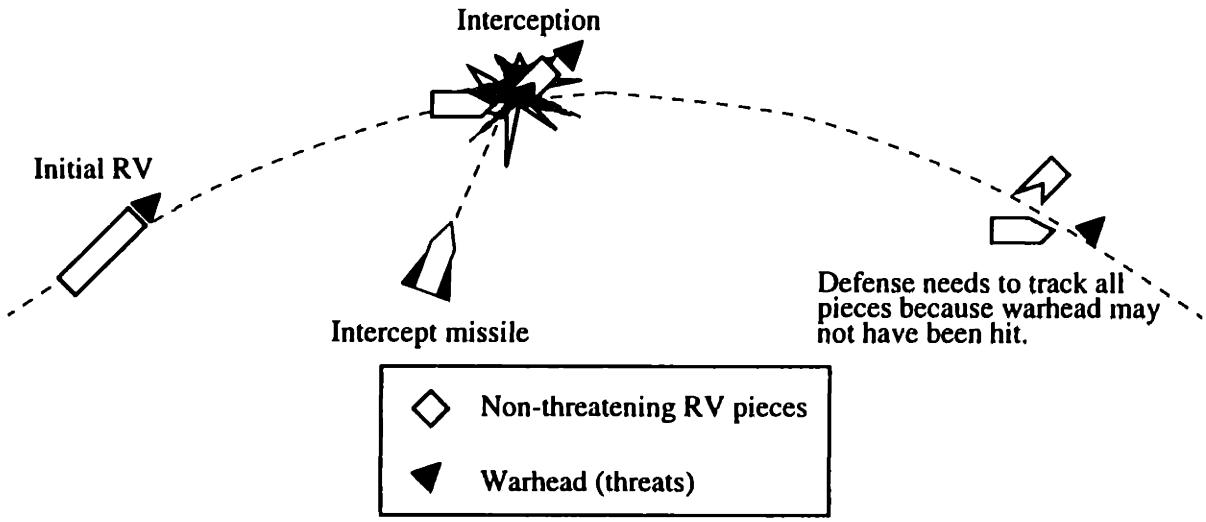


Figure 1-2: Defensive Interception

Chapter 3 defines the problem of target breakup detection more rigorously and proposes a solution which uses Akaike's criterion. Discussions of why the multiple hypothesis tracking formulation was utilized rather than other tracking approaches are given, along with pros and cons of the proposed solution to target breakup detection. This chapter is organized in a methodological manner, steadily building up to the solution for target breakup detection. Discussion is inserted wherever appropriate, and thus, a chronological reading is recommended.

The theory for target breakup detection is presented in Chapter 3. Chapter 4, on the other hand, discusses the implementation of the proposed target breakup detection solution. Implementation issues are discussed, especially the tradeoffs between speed, accuracy, and memory requirements, since the applications of target breakup detection necessitate real-time execution.

Execution results on simulated data are given in Chapter 5 along with discussion on them, and Chapter 6 proposes some possible additional work for the future.

Appendix A presents the derivation of Akaike's criterion, taken from [1].

Appendix B prints a listing of the FORTRAN code which implements the multiple hypothesis tracker with target breakup detection.

Chapter 2

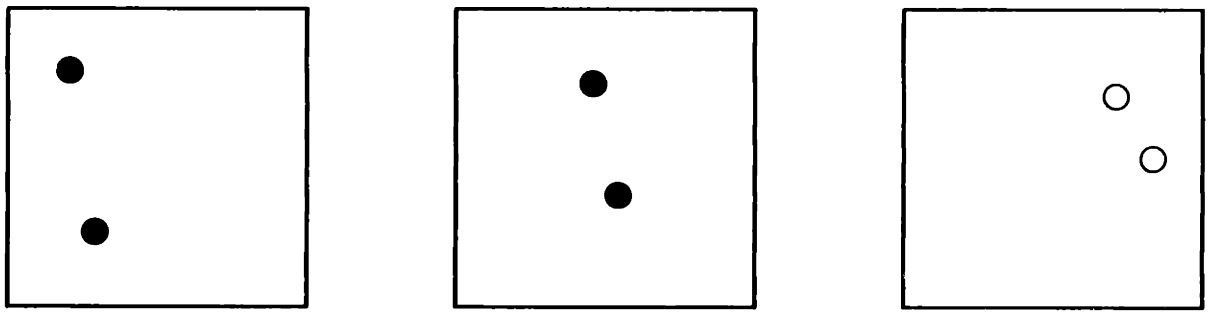
Background

Before tackling multiple target tracking, single target tracking will be discussed first to lay down some fundamental ideas of the kinds of estimation occuring in tracking. Then, multiple target tracking will be presented followed by a detailed presentation of the multiple hypothesis tracking approach to multiple target tracking.

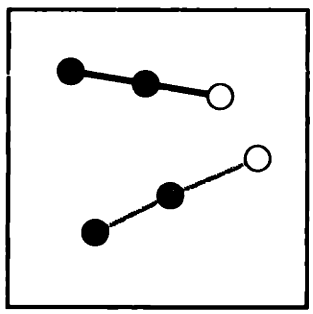
2.1 Single Target Tracking

The purpose of tracking is to discriminate which measurements originate from which targets in a given volume of space, known as the *scan volume*. *Targets* are the actual objects moving through the scan volume whose positions can be measured at several instants of time. These instants of time are called *scans* and contain several measurements depending on the number of targets present in the scan volume. Figure 2-1a shows three scans each with two measurements, which are represented by the shaded dots. Dots of the same shade represent measurements obtained in the same scan. Darker dots represent measurements obtained earlier than measurements represented by lighter dots, which are the more recently obtained measurements. The tracker will collect measurements from different scans into *tracks* as in Figure 2-1b; thus, each track represents a possible target. It is then the tracker's job to determine which tracks are more credible than others.

To simplify this problem, first consider the case where only one measurement is



a) Three scans with two measurements in each.



b) Two possible tracks from these measurements.

Figure 2-1: Pictorial representation of tracks.

obtained at each scan and that measurement actually originated from the target. This simplified problem of tracking is essentially the estimation of the state variables of the moving target through the scan volume. The state variables, which in tracking are the position and velocity components of the target, follow a certain transition function as time progresses dependent on the dynamics of the moving target. Estimation of the state variables is based on the measurements, or observations, of the position of the target taken at different scans. These measurements may be corrupted by noise from disturbances in the environment or from limitations of the sensors. Thus, putting this description into mathematical equations,

$$\dot{\underline{x}}(t) = \underline{f}(\underline{x}(t), t) + \underline{w}(t) \tag{2.1}$$

$$\underline{z}(t) = \underline{h}(\underline{x}(t), t) + \underline{v}(t) \tag{2.2}$$

where

- the underscore represents vectors,
- $\underline{x}(t)$ is the vector of state variables of the target,
- $\underline{z}(t)$ is the vector of measurements at time t,
- the function $\underline{f}(\underline{x}(t), t)$ describes the dynamics of the target,
- the function $\underline{h}(\underline{x}(t), t)$ converts the state vector into a measurement vector,
- $\underline{w}(t)$ is the process noise in the model to account for any model errors, and
- $\underline{v}(t)$ is any noise that corrupts the measurements.

Thus, the sought estimate is the estimate of $\underline{x}(t)$ based on the measurements $\underline{z}(t)$. Note that the dynamics of the target and the measurements received are continuous; however, in any real implementation of a tracker, measurements are received in scans, which are discrete snapshots of the continuous measurements, $\underline{z}(t)$. The times for these scans will be denoted by subscripting the time variable t with an integer which

represents the scan that the time represents. Therefore, t_0 represents the time for the initial scan of measurements, and $t_1, t_2, \ldots$ represent subsequent scan times.[1] In terms of radar tracking of targets with *no drag effect* and *no maneuverability* which is the scope of this thesis, the variables and functions of Eqn. 2.1 and Eqn. 2.2 are defined as below.

- $\underline{x}(t)$ is a six component vector. The first three components are of position; the last three are of velocity. All coordinates are in terms of the earth-centered inertial (ECI) coordinate system. [6] describes coordinate systems in detail.
- $\underline{z}(t)$ is a three component vector. Since this is the return from a radar, the components are range, azimuth, and elevation (RAE) measurements of position.
- $\underline{f}(\underline{x}(t), t)$ describes the dynamics of a target above the earth's surface without drag effect. This is the model of the moving target.
- $\underline{h}(\underline{x}(t), t)$ converts the position part of $\underline{x}(t)$ from ECI to RAE coordinates. This function implicitly takes as input the position of the radar to compute this conversion.
- $\underline{w}(t)$ is the process noise that accounts for model errors.
- $\underline{v}(t)$ is the noise caused by the environment and radar imperfections.

With such a state-space realization of the dynamics and measurements of the target as shown in Eqn. 2.1 and Eqn. 2.2, estimation by the extended Kalman filter, as described in [5], is a natural choice. This single target tracking is not much more than a nonlinear estimation problem, and its solution has been studied extensively in classic estimation and detection theory.

[1] t_0 does not necessarily represent *the* first scan of measurements. It could represent the time for the beginning of a track, as in the maximum likelihood derivations of Chapter 3. Look in the context to see exactly what t_0 represents.

2.2 Multiple Target Tracking (MTT)

The above described simplified problem of tracking masks the real problem of tracking. If several targets are allowed to be in the scan volume, not only must estimation be performed for each track, the origin of the measurements must be determined. In the single target tracking case, the measurement was known to have come from the one target in the scan volume; however, once more than one target exists in the scan volume, it is no longer certain which target generated which measurement. Thus, the fundamental problem of multiple target tracking is *data association* of measurements to tracks.

2.2.1 Difficulties in MTT

Assume that there are exactly S measurements in each scan that correspond to exactly S targets that exist in the scan volume. Even with the assumptions that the number of targets is known and each target present in the scan volume generates exactly one measurement and nothing else generates a measurement, the data association problem is a difficult one. Relaxing these assumptions, the following three *extra* difficulties can occur:

- false reports,
- missing reports, and
- unknown targets.

Background clutter could generate a measurement located randomly in the scan volume; this is a false alarm, or *false report.* These measurements need to be identified so that they do not cause misassociations. Figure 2-2 shows that a miscorrelation to a false alarm (dotted line) could mislead the tracker to stray from the real target (solid line).

If the probability of detection of the sensor is less than unity, then it is possible that a target may not generate a measurement for some scans. In this case, data

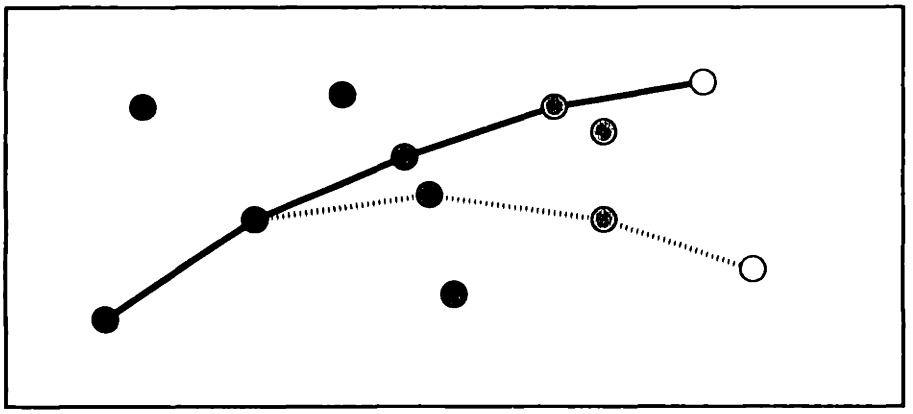


Figure 2-2: False Reports

association must be able to ignore the scan and continue association afterwards to account for these *missing reports* (See Figure 2-3). Missing reports may arise for a variety of reasons. Noise in the background could have disturbed the signal to the sensor, or the sensor may have malfunctioned briefly.

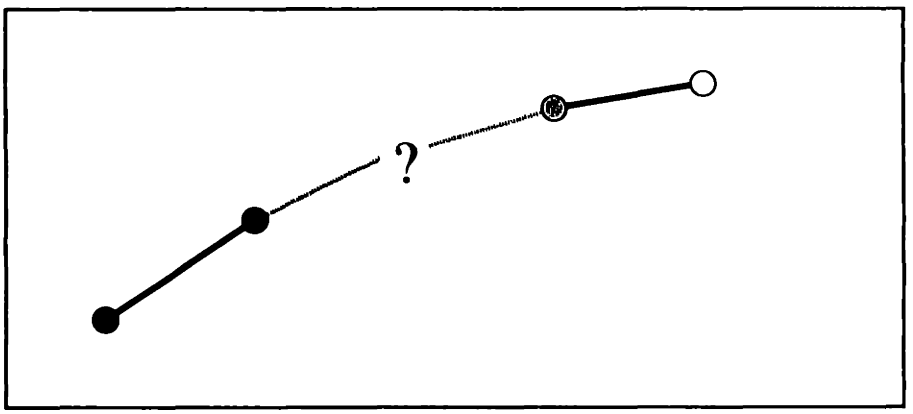
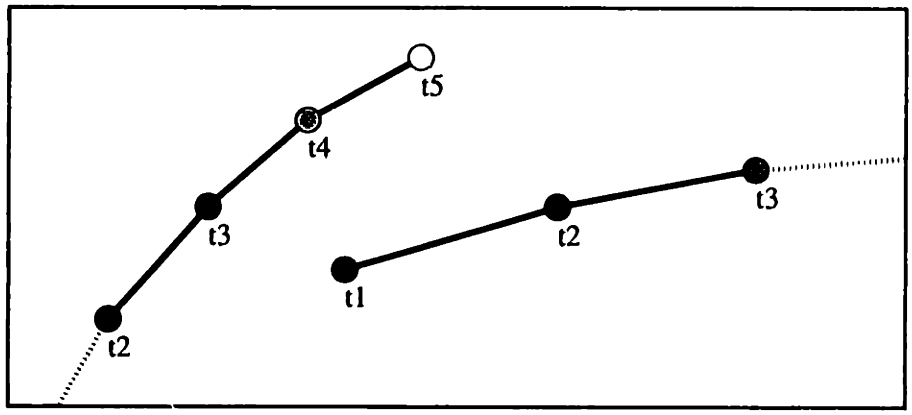


Figure 2-3: Missing Reports

Targets may enter or leave the scan volume during the tracking period. Thus, the number of targets and their initial states are unknown *a priori* which necessitates the tracker's ability to initiate and drop tracks at any time. These *unknown targets* seem to be the most difficult problem in multiple target tracking. Figure 2-4 shows a target that was initiated at time t_2 and a target that terminated at time t_3. The number



Times t1-t5 refer to the scan that measurements appeared in the scan volume.

Figure 2-4: Unknown Targets

of targets can change during the course of tracking. At times t_1, t_4, and t_5, only one target is present in the scan volume while at times t_2 and t_3, two targets are present. Since the number of targets can change at any time, all existing tracks could terminate at any time, and every new measurement could be the beginning of a new track. Thus, the number of data association possibilities from not knowing the number of targets *a priori* is tremendous. Regarding target breakup, since breakup initiates breakup pieces in the scan volume which is similar to track initiation, detecting target breakup will necessitate the tracker's ability to handle this unknown target difficulty.

A good solution approach should be able to handle these three difficulties. To aid in dealing with these difficulties, the concepts of gating and clustering will be presented in the next section.

2.2.2 Gating and Clustering

Since a track's state vector includes a velocity part, the position of the target at the next scan can be estimated. A region is defined around this next predicted position called a *gate* as shown by circles around the x's in Figure 2-5.

The exact shape of the gate can be rectangular or ellipsoidal depending on the particular tracking application. The size can either be set to an *a priori* value or determined by the covariance of the position estimate. In particular to the implementation

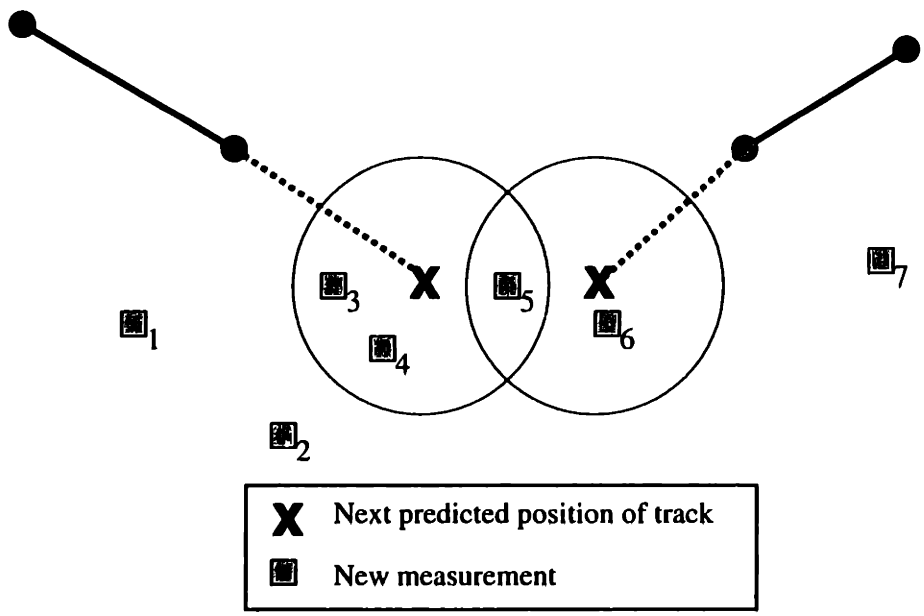


Figure 2-5: Gating

used in this thesis, the gate is an ellipsoidal one set to an *a priori* size.

The purpose of gating is to separate those measurements that are candidates for track continuation from those that are not. In this way, any measurements that do not lie in an existing track's gate could either be false alarms or the start of a new track but cannot continue an existing track, such as measurements 1, 2, and 7 in Figure 2-5. Only those measurements that do lie in a track's gate can continue the track, but they are not limited to only continue the track since those measurements can also be false alarms or the start of a new track (measurements 3, 4, 5, 6). It is possible for more than one measurement to lie in a track's gate. This is the case of an ambiguous correlation, and more information would be useful in determining the correct measurement to continue this track. Also, notice that it is possible for two different tracks' gates to overlap. If a measurement falls in this overlapped region (measurement 5), it is uncertain as to which track this measurement should correlate, if any. Several approaches have been proposed to deal with these ambiguous correlations, three of which are briefly described in Section 2.2.4.

Taking the basic premise of gating, separating measurements into different groups,

to a macro level results in *clustering*, which divides the scan volume into several regions where tracking is done independently of each other. The greatest advantage of clustering is reducing the computation required for data correlation. Thus, clustering is an optimizing technique for tracking solutions.

2.2.3 Orientations for solution approaches

Three general approaches to tracking which mainly differ in how they go about analyzing a scan of data are:

Target-oriented Approach This approach is the simplest of the three general approaches. The number of targets and the initial velocities of the targets are known *a priori*, and it is assumed that the number of tracks never changes. Thus, the number of tracks and their initial state vectors are given. For each track, the best measurement or some combination of the measurements in the current scan is selected to continue the track.

Track-oriented Approach Like the target-oriented approach above, the number of targets and their initial velocities are known beforehand in the track-oriented approach, along with the assumption that the number of targets never changes. For each track, rather than selecting the best measurement or some combination of the measurements to continue the track immediately, a new track is created for each measurement that lies in the original track's gate; this is called *track-splitting*. Only a certain number of the best tracks is kept after each scan. In general, the number of tracks kept after each scan is greater than the number of targets assumed to exist in each scan. This allows delaying some decisions about ambiguous correlations until more scans of measurements are obtained. Note that the track-oriented approach is the same as the target-oriented approach if the number of tracks kept after each scan is the same as the number of targets assumed to be in the scan.

Measurement-oriented Approach For each new scan of measurements, each measurement is hypothesized to either be the continuation of an existing track if

it lies in an existing track's gate, a false alarm, or the start of a new track. Therefore, this approach makes no assumptions on the number of targets or their initial velocities beforehand.

The target-oriented and track-oriented approaches inherently look only for track continuation because they consider only those measurements that lie in existing tracks' gates. Measurements that do not lie in a track's gate are not analyzed which is not necessarily faulty since those measurements are most likely false alarms. However, by ignoring these measurements, these two approaches cannot handle track initiation.

The measurement-oriented approach analyzes each measurement for every possible kind of association. It explicitly considers each measurement to be continuation of an existing track, a false alarm, or a new track. Since this approach does look at all possibilities, it can enumerate all possible data associations, and thus, theoretically, it is an optimal approach. On the other hand, practically, computation is the most complex for the measurement-oriented approach.

2.2.4 Some MTT Schemes

In considering the difficulties of multiple target tracking, several schemes have been devised in the literature.

Nearest-Neighbor (NN) Tracking The measurement closest to the predicted position of the track is considered to be the measurement generated by the target. This approach is the most prone to incorrect data association since recovery from one misassociation is virtually impossible. Furthermore, nearest-neighbor tracking fails when there is a lot of clutter or when tracks cross.

Joint Probabilistic Data Association (JPDA) This method was developed by Bar-Shalom in [2] which is a multiple-target extension to his probabilistic data association filter developed in [3]. Data association is performed by calculating the probability that each measurement was generated by each track. Then, each track is updated by combining all the measurements in proportion to the

probability that the measurement was generated by the track. This scheme accounts for clutter and crossing tracks well; however, since it is a target-oriented approach, there is no natural way to handle track initiation.

Multiple Hypothesis Tracking (MHT) The fundamental idea that lies in multiple hypothesis tracking is to delay determining ambiguous correlations when more information is obtained. This amounts to keeping several *hypotheses* of data associations, and as more scans are obtained, the less likely hypotheses are dropped. This approach handles clutter and crossing tracks much better than the JPDA or the NN approach; furthermore, since this is a measurement-oriented approach, track-initiation is handled automatically. How to go about enumerating all of these hypotheses was a problem until Reid suggested an algorithm in [8].

There is a tradeoff between performance and computational complexity in the three algorithms described above. The NN approach is minimal in terms of computation, but it fails too easily for most practical uses. The JPDA approach yields better tracking at the cost of computational complexity since it must calculate probabilities to perform data association; however, without the ability to initiate tracks, this approach must be accompanied by a seperate track initiator. The MHT approach can handle cases where the NN and JPDA approaches fail as well as initiate tracks; however, this approach requires a tremendous memory capacity, and it is computationally the most complex of the three approaches.

It is the MHT's ability to initiate tracks that has led it to being chosen for incorporating target breakup detection. Chapter 3 discusses these reasons in more detail.

2.3 Multiple Hypothesis Tracking

In order to handle the difficulties of multiple target tracking, multiple hypothesis tracking keeps several hypotheses between scans so that later scans can help determine any ambiguous correlations between tracks and measurements. By delaying the

correlations of the tracks with the measurements, the information from future scans can aid in past correlations. This capability to use later measurements to aid in prior correlations, called *multiple-scan correlation*, makes the handling of false reports, missing reports, and unknown targets easier.

A hypothesis, in terms of MHT, is one possible data association of all the measurements ever encountered up to that scan. Thus, in more concrete terms, a *hypothesis* is defined as a collection of tracks such that all measurements ever encountered are a part of some track of the hypothesis. In this way, the set of hypotheses span all possible data associations.[2] Furthermore, the assumption will be made that any target in the scan volume will produce at most one unique measurement in any scan. Thus, a target will never generate multiple measurements in the same scan. Considering this *uniqueness* of measurements, no two tracks in the same hypothesis can have any measurements in common.

It then becomes apparent what the MHT formulation must do. It must:

- generate all possible hypotheses of all the measurements encountered thus far.
- rank the hypotheses based on the likelihood of each hypothesis, which is based on the *quality* of the tracks contained in the hypothesis.

The generation of all hypotheses was an unsolved problem until Reid proposed an algorithm in [8]. Reid's method and a slightly modified version of it, which is more in the likeness of Blackman's implementation method proposed in [4], will be presented first. Then, the ranking of the hypotheses will be discussed.

2.3.1 Hypothesis Generation

As mentioned before, each measurement represents one of the following three possibilities:

- continuation of an existing track,

[2]Note that false alarms are the special case of a one point track.

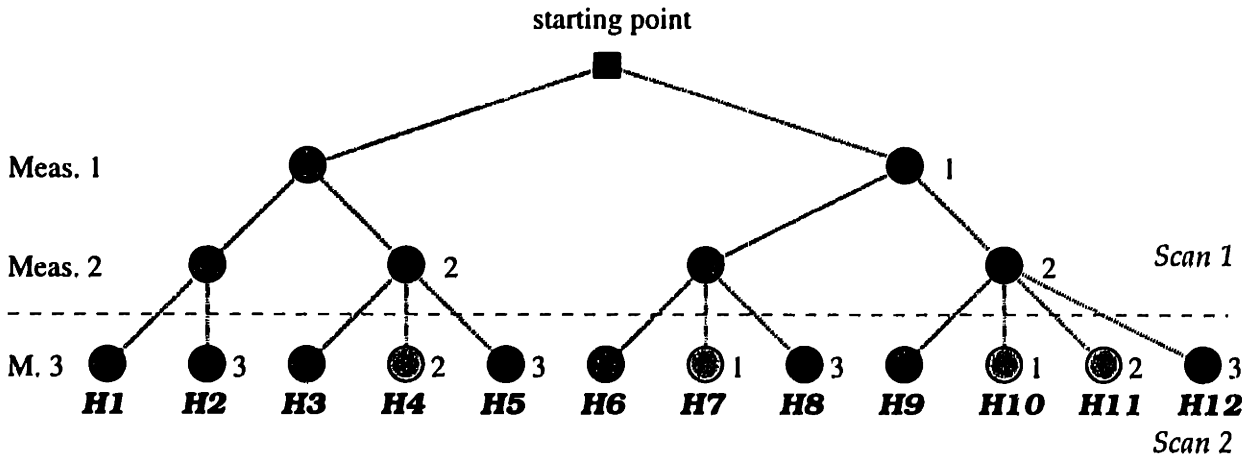
- false alarm, or
- start of a new track.

Generating hypotheses using a *measurement-oriented* approach seems more appropriate than using the other general approaches because of the track initiation requirement. The method presented by Reid in [8] basically creates a tree where each level of the tree pertains to a measurement (See Figure 2-6). The nodes at a particular level describe that measurement's origin, either as the continuation of some track that existed before, a false alarm, or the start of a new track. Measurements that are in the same scan cannot originate from the same target since a target generates up to one unique measurement. After creating all the branches of the tree, the hypotheses are each leaf of the tree where the origin of all of the measurements can be found by traversing up the tree from that leaf. Some problems with this method of hypothesis generation is that tracks are never hypothesized to terminate and tracks are not allowed to have any missing measurements in them. Thus, the possibility of track termination or missing reports is not hypothesized.

The method proposed by Blackman in [4] *does* handle track termination and missing reports. However, it is difficult to represent the hypotheses in a tree, so hypotheses are represented as a list of tracks and all possible tracks are stored in a track list. Figure 2-7 shows the flow diagram for this algorithm. This algorithm basically creates all the possible tracks that can be formed from each measurement in the scan, including tracks that have missing observations or have terminated. Then, hypotheses are formed by making as many combinations as possible with the tracks available so that the definition of a hypothesis is fulfilled.

2.3.2 Hypothesis Ranking

A score is assigned to every hypothesis generated so that the best hypothesis can be determined after each scan. Furthermore, since all the hypotheses are kept after



KEY

- Measurement is considered a false alarm.
- Measurement is considered the continuation of a track. The number refers to the track that this measurement continues.
- Measurement is considered the start of a new track. The number is the track's identifying number.

Hypotheses Description

H1 ***Meas. 1,2,3 false alarms***

H2 ***Meas. 1,2 false alarms; Meas. 3 start of track 3***

H3 ***Meas. 1,3 false alarms; Meas. 2 start of track 2***

H4 ***Meas. 1 false alarm; Track 2 consists of Meas. 2,3***

H5 ***Meas. 1 false alarm; Meas. 2 start of track 2; Meas. 3 start of track 3***

H6 ***Meas. 1 start of track 1; Meas. 2,3 false alarms***

H7 ***Meas. 2 false alarm; Track 1 consists of Meas. 1,3***

H8 ***Meas. 1 start of track 1; Meas. 2 false alarm; Meas. 3 start of track 3***

H9 ***Meas. 1 start of track 1; Meas. 2 start of track 2; Meas. 3 false alarm***

H10 ***Track 1 consists of Meas. 1,3; Meas. 2 start of track 2***

H11 ***Meas. 1 start of track 1; Track 2 consists of Meas. 2,3***

H12 ***Meas. 1,2,3 start tracks 1,2,3 respectively***

Note: Measurement 3 is assumed to gate with both tracks 1 and 2.

Figure 2-6: Hypothesis Tree

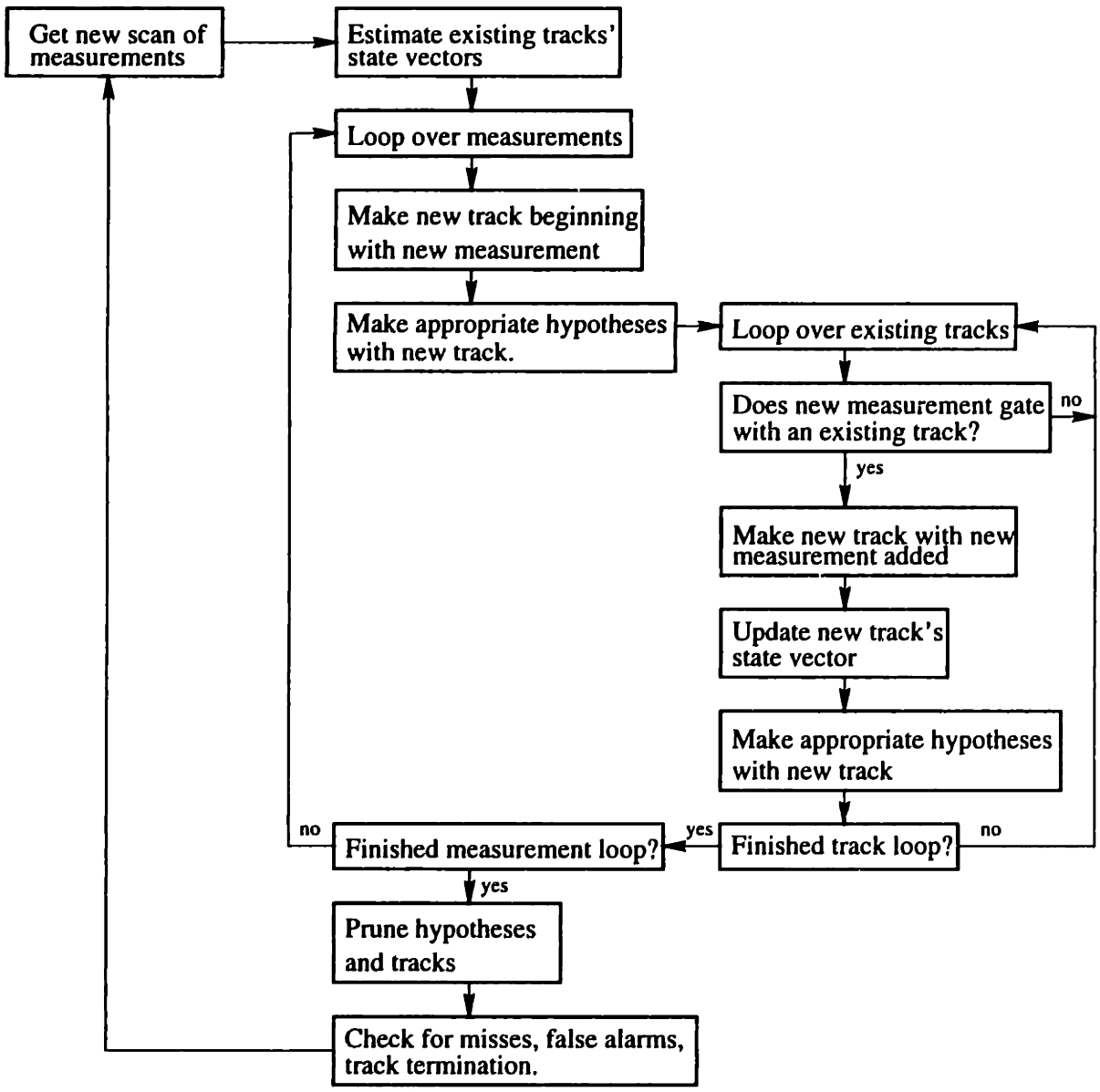


Figure 2-7: MHT Flow Diagram

every scan[3], the highest ranking hypothesis is indeed the best hypothesis up to that scan. Thus, the MHT approach is an optimal solution.

The score of a hypothesis is dependent on the quality of its tracks and the kinds of data associations made by the hypothesis. Determining the quality of a track is straightforward since a model of the target is known and the target's locations at different scans can be estimated. χ^2 statistics then determine the track's quality.

The quality of a track cannot be the only measure for the hypothesis score because some kinds of associations inherently yield a better fit. Consider two hypotheses when there are four scans with one measurement in each, placed such that the measurements seem to fall into a possible target trajectory:

$\mathbf{H_0}$: All measurements come from one track.

$\mathbf{H_1}$: All measurements are false alarms.

Clearly, H_0 is the better hypothesis, but since there is noise in the measurements and track states are estimated, there must be some error involved. H_1, on the other hand, has no error since no estimation is involved. Thus, the *kind* of association must also be considered for the hypothesis score. Assuming various probabilities for false alarms, noise in the measurements, and detectability by the sensor, the probability of the data association, which include false alarms, missing measurements, and terminated tracks, can be calculated. In [4, Section 14.3], the probabilities of the different kinds of tracks are given. Tracks are given a status based on the number of measurements in them and how well those measurements fit the tracks, and consequently, the status determines the data association probability.

2.3.3 Hypothesis Pruning

The number of hypotheses generated increases exponentially. Since memory is not infinite in a real implementation, the number of hypotheses stored after every scan

[3]Theoretically, hypotheses are never deleted, or *pruned*, although in any real implementation, the number of hypotheses kept after every scan is limited by memory requirements.

must be limited. Thus, hypothesis *pruning* becomes an important implementation issue. Limiting the number of hypotheses runs the risk of pruning the "correct" hypothesis; however, keeping too many hypotheses wastes too much computation time on unlikely hypotheses. Two methods of pruning immediately come to mind:

Method 1 Keep all hypotheses that have a score higher than some preset threshold value.

Method 2 Keep the best, say N_H, hypotheses after each scan.

Method 1 is a fairly good way of pruning except in the case where no hypothesis score is higher than the threshold value in which case tracking cannot continue. Furthermore, the number of hypotheses kept varies from scan to scan, and there may be times where very few hypotheses are kept. When very few hypotheses are kept, the basic premise of *multiple* hypothesis tracking is lost.

Method 2, since it always keeps the same number of hypotheses from scan to scan, always has the flavor of a *multiple* hypothesis tracking algorithm and the case of no hypothesis kept never arises. However, one problem with this method of pruning is that if there are more than N_H hypotheses with virtually the same score, then some "good" hypotheses will be pruned.

A more complex method of pruning will be presented in Chapter 4 to deal with accidentally pruning "good" hypotheses in Method 2.

Chapter 3

Target Breakup Detection

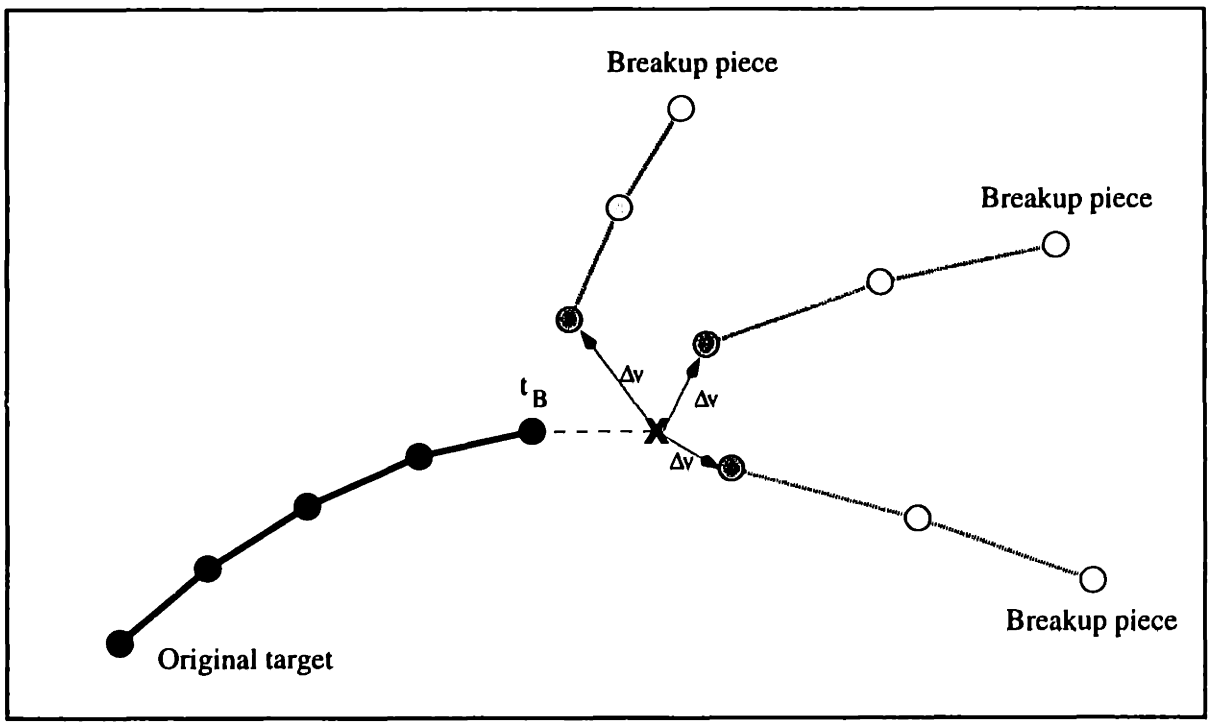


Figure 3-1: Target Breakup Parameters

The aim of target breakup detection is to determine whether a target has broken up and, if it has, track the breakup pieces. This entails finding (See Figure 3-1):

- t_B, the time of breakup for the target, and
- ΔV, the change of velocity that breakup causes to each of the breakup pieces.

In this chapter, "breakup track" refers to the original track and its breakup pieces collectively. So, all the measurements of Figure 3-1 are the measurements of a single "breakup track". Chapter 4 uses "breakup track" differently.

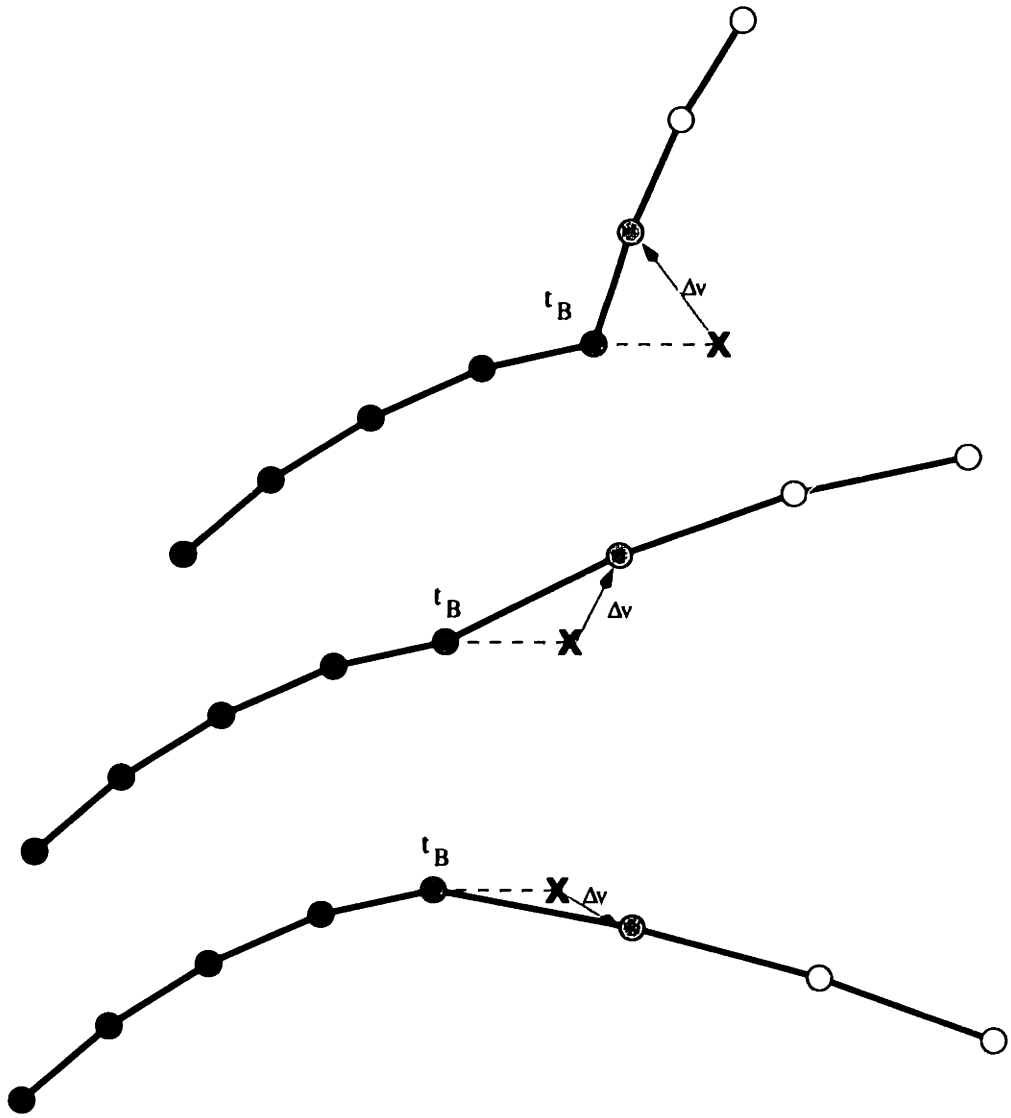


Figure 3-2: Breakup Tracks as Tracks with Perturbation

There are several ways to approach this problem. One could be to consider breakup tracks as those tracks whose trajectory had been perturbed by a ΔV. Figure 3-2 shows three tracks with a ΔV in each. Note that these tracks are the breakup piece tracks of Figure 3-1 extended to include the original target track. This approach would estimate the most likely time that ΔV occurred, t_B, and then estimate ΔV itself. In [10], Willsky proposes a generalized likelihood ratio approach which can

detect whether the track has a *significant* ΔV perturbation in its trajectory. However, a problem with this approach is that any choice of the threshold value, which represents the significance of the ΔV, is a dubious one since breakup causes a ΔV in any direction with any magnitude, large or small.

Rather than look for target breakup on a track-by-track basis, target breakup can instead be detected on a measurement-by-measurement basis. Since measurements are obtained one scan at a time, each new measurement could have been generated by a newly created breakup piece of some existing track. By approaching the problem this way, the breakup time, t_B, is gotten automatically, ΔV can be estimated since the breakup piece is hypothesized to have broken off from some existing track whose state vector is known, and the significance of the breakup is determined by how well the breakup track correlates with future measurements. Thus, incorporation into the MHT formulation seems to be the best choice, not only because it is measurement-oriented but also because it handles track initiation.

3.1 Problems with Direct Incorporation

Incorporating target breakup detection into the multiple hypothesis tracking formulation seems straightforward. In addition to the standard hypotheses of being a new track, the continuation of an existing track, or a false alarm, each measurement can now be hypothesized to be the start of a breakup piece of an existing track. A closer look at this straightforward solution reveals several inconsistencies.

Consider the case of N scans of measurements from scan times, $\{t_1, \ldots, t_N\}$, when there is one target traversing across the scanning volume (assume no false alarms). The following are possible hypotheses:

$\mathbf{H_0}$: All measurements come from one track. No breakup ΔV in this hypothesis.

$\mathbf{H_1}$: Breakup has occured at some time $t_B \in (t_1, \ldots, t_N)$. A breakup ΔV is incorporated.

$\mathbf{H_2}$: There are two independent tracks: one starting from time t_1 and ending at time

t_B; the other, starting from time t_{B+1} and ending at time t_N.

$\mathbf{H_3}$: All measurements are false alarms.

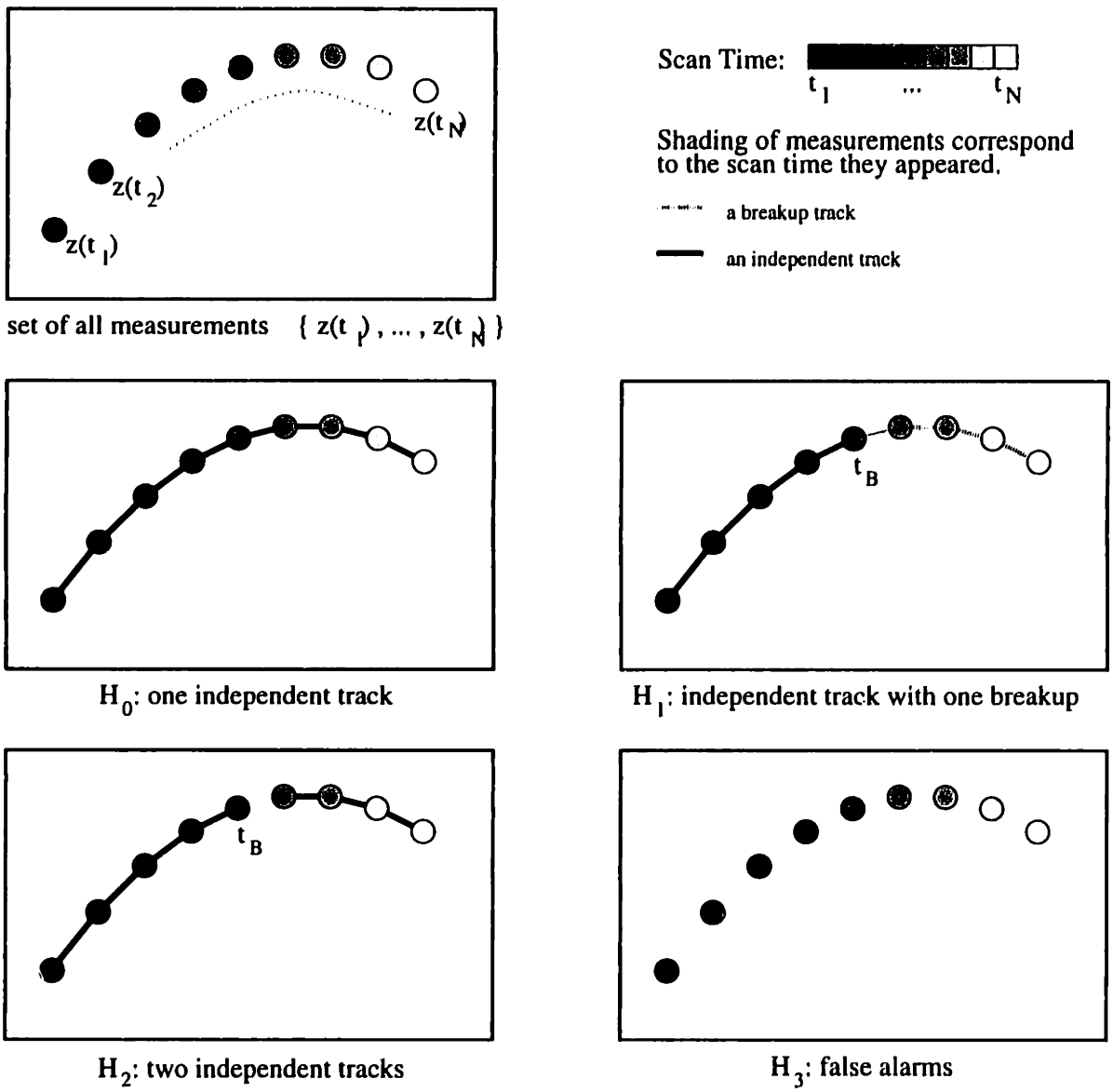


Figure 3-3: Possible Hypotheses when Target Breakup is Allowed

Figure 3-3 shows a pictorial representation of the above hypotheses. Looking carefully at these three hypotheses, it can be seen that

$$p(\mathbf{z} \mid H_0) \leq p(\mathbf{z} \mid H_1) \leq p(\mathbf{z} \mid H_2) \leq p(\mathbf{z} \mid H_3)$$

where

$$\mathbf{z} = \begin{bmatrix} \underline{z}(t_1) \\ \vdots \\ \underline{z}(t_N) \end{bmatrix}$$

and $\underline{z}(t_i)$ is the measurement at scan time t_i. H_1 and H_2 always fit the measurements better than H_0; however, since H_1 and H_2 use more complex models to fit the data, these two models may be *fitting* noise rather than *filtering out* noise. In this case, the interpretation of the data by H_1 and H_2 is more complex than necessary. H_2 will always fit the data better than H_1 because breakup allows only for a change of velocity whereas two independent tracks could be thought of as a breakup where the change of state is on both position and velocity. H_3 has perfect fit to the measurements, but this is a degenerate solution.

Thus, the question arises: what is a suitable loss function for the different models used to fit the data? In particular to this problem, what is a suitable penalty for the different hypotheses so that the fit to the data becomes statistically significant?

Note that the MHT formulation can already handle discriminating among H_0, H_2, and H_3 by calculating a score based on the status and quality of its constituent tracks. Allowing H_1 would entail calculating the probability of data association for breakup, which would require specifying some *a priori* densities, such as the likelihood a target will break up. Furthermore, this data association probability needs to be more likely than the data association probability for all independent tracks but less likely than for a single independent track. These probability calculations based on subjective densities, with a lot of tweaking, may result in a solution that can detect target breakups; however, a more fundamentally rooted solution is desirable so that subjective densities are not required.

The different data associations, H_0, H_1, H_2, and H_3, mentioned above differ in their complexity. The higher numbered hypotheses fit the measurements better because the model used to fit the data is more complex. This intuitive notion of "complexity" is perhaps the fundamental idea that will lead to a non-subjective solution.

3.2 Complexity vs. Likelihood

The intuitive notion of complexity is that a more complex model inherently has a better fit to the data so complexity should be a penalizing factor. To describe this mathematically, let Θ be the set of all possible data associations of the measurements, and let θ be an element of Θ. Define some cost quantity:

$$Cost(\theta) = -\alpha\ Likelihood(\mathbf{z} \mid \theta) + \beta\ Complexity(\theta)\ . \tag{3.1}$$

The first term determines how well the data fits the model, θ[1]. The second term, the complexity function, is the loss function which penalizes more complex models since the more complex model will invariably fit the data better than a simpler one. The constants, α and β, appropriately balance these two terms. Then, the θ that minimizes the cost will be the best model, which is the model with the least complexity that fits the data adequately.

The standard multiple hypothesis tracking formulation has compensated for cases such as H_3 by assigning scores to tracks and, ultimately, to hypotheses; the scores penalize false alarms. Thus, the cost function defined above can be adopted as the score for tracks and hypotheses. The balance between complexity and fit to data is crucial for target breakup detection, and for that matter, the entire multiple hypothesis tracking formulation to work properly.

This identification of the statistical model for a given data set has been studied in the previous literature. Standard estimation and detection techniques have been shown to be inadequate in identifying the underlying model generating the data. Although a loss function can be created using subjective arguments, Akaike [1] and Rissanen [9] have defined their loss function using basic principles. Akaike's criterion uses the minimization of the Kullback-Leibler mean information to determine the correct model. Essentially, the complexity of the model is the number of free parameters necessary to describe the model. Rissanen's minimum description length

[1]The different data associations are the different models, which in MHT are the different hypotheses for the data. Thus, model, data association, and hypothesis will be used interchangeably.

(MDL) is based on a more fundamental idea, the minimum number of bits necessary to describe a data set. Although the MDL is based on a more fundamental principle than Akaike's criterion, because of the simplicity of Akaike's criterion and the ease with which it fits into the MHT formulation, Akaike's criterion is utilized to come up with an appropriate cost function. A derivation of Akaike's criterion is presented in Appendix A. The next section describes and justifies its application to target breakup detection.

3.3 Akaike's Criterion

Akaike's criterion is defined as:

$$AIC(\theta) = -2\log(\textit{maximum likelihood of } \theta) + 2k(\theta) \tag{3.2}$$

where $k(\theta)$ is the number of independent free parameters in the model, θ, and the maximum likelihood is of the data conditioned on θ. Thus, the estimate of θ is:

$$\hat{\theta} = \arg_{\theta}\min\ AIC(\theta)\ . \tag{3.3}$$

Note that Eqn. 3.2 is of the same form as Eqn. 3.1, which matches the intuitive notion of complexity and hints at the adoption of Eqn. 3.2 as the score function. The first order of business is to see if the complexity function, $2k$, matches the intuitive notion of the complexity function in Eqn. 3.1. The second step is to derive the maximum likelihood functions. The third step is to determine how to go about detecting target breakup and estimating parameters for target breakup pieces. And finally, the effect of adding target breakup detection into the MHT formulation is discussed.

3.3.1 Complexity Function Discussion

Since complexity under Akaike's criterion involves counting the number of free parameters used in the model, the first thing to do is to count the number of free parameters for the different kinds of tracks. Then, the total complexity for a hypothesis is the

sum of the complexities of each track in the hypothesis. Table 3.1 shows the number of free parameters for each type of track.

Table 3.1: Free Parameters in Tracks

Type of Track	Free Parameter Description	Total
Independent Track	(3) initial position (3) initial velocity (1) start time	7
Lost Track	(3) initial position (3) initial velocity (1) start time (1) end time	8
False Alarm	(3) initial position (1) occurrence time	4
Track that Breaks Up Into N_B Pieces	(3) initial position (3) initial velocity (1) start time (1) breakup time ($3N_B$) velocity difference	$3N_B + 8$
Track with N_M Missing Measurements	(N_M) time for each missing measurement	add N_M

A discussion of the number of parameters and how it matches the intuitive notion of a complexity function follows.

1. An independent track is a track that is still a candidate for continuation. It should be the least complex of all the different types of tracks. At first glance, false alarms seem to be less less complex. However, suppose that an independent track consists of N_T measurements. The number of free parameters of the independent track is 7, while the number of free parameters to describe all the measurements as false alarms is $4N_T$. So for $N_T > 1$, describing the data as an independent track is less complex. Notice that a lost track has the same number of free parameters as two false alarms. There should be no difference in describing two measurements as a two-measurement lost track or as two false alarms, and indeed, there is not.

2. There should be a penalty if there are missing measurements in a track. Table 3.1 shows that there is a penalty of one free parameter for every missing measurement.

3. Describing the measurements as a breakup track with several breakup pieces should be less complex than describing the measurements as several independent tracks, where each breakup piece is considered an independent track, since breakup pieces are constrained to originate from the same position at the same time and indpendent tracks are not. A track with N_B breakup pieces requires $3N_B+8$ free parameters; describing the same measurements as N_B+1 independent tracks requires $7N_B$ free parameters for the N_B breakup particles plus 8 free parameters for the original target before breakup since that original target is assumed to be lost after breakup, leaving a total of $7N_B + 8$ free parameters. Notice that the independent track description is always greater than the breakup description except when $N_B = 0$, in which case there are no breakup pieces.

Adopting the number of free parameters as the complexity function seems to coincide very well with the intuitive notion of the complexity function. The next step in applying Akaike's criterion is to derive the maximum likelihood expression for a hypothesis.

3.3.2 Maximum Likelihood Derivations

The maximum likelihood in Akaike's criterion is conditioned on the model, θ. The different models in the tracking problem are the different data associations that can be made of the measurements, which are the hypotheses generated by the multiple hypothesis tracking formulation. Measurements in one track can be assumed to be generated independently of measurements in another track. Therefore, the likelihood of a hypothesis, θ, consisting of N_θ tracks is the product of the likelihood of each of its tracks.

$$l(\mathbf{x} \mid \theta) = \prod_{i=1}^{N_\theta} l_i(\mathbf{x}_i \mid \theta) = \prod_{i=1}^{N_\theta} p(\mathbf{z}_i \mid \mathbf{x}_i, \theta) \tag{3.4}$$

where

$$\mathbf{z} = \begin{bmatrix} \mathbf{z}_1 \\ \vdots \\ \mathbf{z}_{N_\theta} \end{bmatrix}$$

and

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_{N_\theta} \end{bmatrix} .$$

$\mathbf{z}$ is a $3M$-dimensional vector of the M measurements encountered thus far, each measurement having three components, range, azimuth, and elevation. Each $\mathbf{z}_i$ is the set of measurements for track i and has dimension, $3N_{T_i}$, where N_{T_i} is the total number of measurements in track i. So, for a non-breakup track i,[2]

$$\mathbf{z}_i = \begin{bmatrix} \underline{z}_i(t_0) \\ \vdots \\ \underline{z}_i(t_{N_{T_i}-1}) \end{bmatrix} . \tag{3.5}$$

For the case of a breakup track i that has broken up into N_{B_i} breakup pieces after time t_{B_i}, without loss of generality the measurements of the original track before

[2]Note that t_0 refers to the scantime when the first measurement of track i appears, not the initial scan time of the tracker. Thus, t_0 is the start time of the respective track and is different for different tracks.

breakup and all the breakup pieces of the breakup track are

$$\mathbf{z_i} = \begin{bmatrix} \underline{z}_{i_0}(t_0) \\ \vdots \\ \underline{z}_{i_0}(t_{B_i}) \\ \underline{z}_{i_1}(t_{B_i+1}) \\ \vdots \\ \underline{z}_{i_1}(t_{end_i}) \\ \vdots \\ \underline{z}_{i_{N_{B_i}}}(t_{B_i+1}) \\ \vdots \\ \underline{z}_{i_{N_{B_i}}}(t_{end_i}) \end{bmatrix} \tag{3.6}$$

where $\underline{z}_{i_0}(t)$ represents the measurements in the original track before breakup and $\underline{z}_{i_j}(t)$ for $j \neq 0$ represents the measurements in breakup piece j of track i. $end_i + 1$ is the number of time samples in breakup track i and since

$$N_{T_i} = (B_i + 1) + (end_i - B_i)N_{B_i}$$

where end_i and B_i are the subscripts of t_{end_i} and t_{B_i} respectively,

$$end_i = \frac{N_{T_i} - B_i - 1}{N_{B_i}} + B_i \ . \tag{3.7}$$

N_{T_i} is the sum of the number of measurements in the original track before breakup plus the sum of the number of measurements in each of the N_{B_i} breakup pieces. Since each track of a hypothesis, θ, contains N_{T_i} measurements, it follows that

$$M = \sum_{i=1}^{N_\theta} N_{T_i} \ .$$

$\mathbf{x}$ is the vector of all free parameters necessary to describe the hypothesis, θ, and has dimension k. $\mathbf{x_i}$ is the vector of free parameters necessary to describe track i.

To derive an expression for the likelihood (Eqn. 3.4) of a hypothesis, it remains to derive the likelihood for each track in the hypothesis. The likelihood for breakup tracks will be discussed in Section 3.3.3. The rest of the discussion in this section will focus on non-breakup tracks. Let $l_i(\mathbf{x}_i \mid \theta)$ be the likelihood of track i. The model for an independent track, based on Eqn. 2.1 and Eqn 2.2, is:

$$\dot{\underline{x}}_i(t) = \underline{f}(\underline{x}_i(t), t) \tag{3.8}$$

$$\underline{z}_i(t) = \underline{h}(\underline{x}_i(t), t) + \underline{v}_i(t) \tag{3.9}$$

where $\underline{v}_i(t)$ is a zero mean, white Guassian noise process with spectral density, R. The process noise, $\underline{w}(t)$ has been left out of Eqn. 3.8 because it is assumed that the dynamics are known perfectly. The initial condition, $\underline{x}_i(t_0)$ for track i is determined by $\mathbf{x}_i$, the free parameters of the track, which is what needs to be estimated to fit the measurements. Therefore, $\mathbf{x}_i$ contains the position and velocity components of $\underline{x}_i(t_0)$, the initial time t_0, and any other information necessary to describe the track, such as missing measurements, track termination, and breakup. The measurements $\underline{z}_i(t)$ are related to $\mathbf{z}_i$ as shown in Eqn. 3.5.

Before tackling the likelihood function, define the following function:

$$\underline{g}(\underline{x}_i(t_0), t) = \underline{h}(\int_{t_0}^{t} \underline{f}(\underline{x}_i(t), t)dt + \underline{x}_i(t_0), t) \ . \tag{3.10}$$

Then, Eqn. 3.9 can be reduced to

$$\underline{z}_i(t) = \underline{g}(\underline{x}_i(t_0), t) + \underline{v}_i(t) \ . \tag{3.11}$$

If $\underline{x}_i(t_0)$ is treated as a parameter, estimating it is just the problem of estimating a parameter in additive white Guassian noise. Thus, the likelihood function becomes

$$\begin{aligned} l_i(\mathbf{x}_i \mid \theta) &= l_i(\underline{x}_i(t_0) \mid \theta) \\ &= p(\mathbf{z}_i \mid \underline{x}_i(t_0), \theta) \end{aligned}$$

$$l_i(\mathbf{x}_i \mid \theta) = C_i \exp[-\frac{1}{2} \sum_{k=0}^{N_{T_i}-1} (\underline{z}_i(t_k) - \underline{g}(\underline{x}_i(t_0), t_k))^T R^{-1} (\underline{z}_i(t_k) - \underline{g}(\underline{x}_i(t_0), t_k))] \quad (3.12)$$

where

$$C_i = \frac{1}{(2\pi \mid R \mid)^{\frac{N_{T_i}}{2}}} .$$

Maximizing $l_i(\mathbf{x}_i \mid \theta)$ is the same as minimizing the quadratic function

$$J_i^{\theta} = \sum_{k=0}^{N_{T_i}-1} (\underline{z}_i(t_k) - \underline{g}(\underline{x}_i(t_0), t_k))^T R^{-1} (\underline{z}_i(t_k) - \underline{g}(\underline{x}_i(t_0), t_k)) . \quad (3.13)$$

Thus, for the case of additive white Guassian noise, the maximum likelihood estimate is the estimate which minimizes the "cost" function, J_i^{θ}. This is the weighted least squares problem. Furthermore since all the weights, R^{-1}, are equal and noise is Guassian, this problem becomes the minimum mean square estimate. The extended Kalman filter can recursively approximate the minimum mean square estimate of the state vector. The function $\underline{g}(\underline{x}_i(t_0), t)$ is the smoothed estimate

$$\hat{\underline{z}}_i(t) = \underline{h}(\hat{\underline{x}}_i(t \mid \mathbf{z}_i), t) .$$

False alarms can be treated as one point tracks. Obviously, the maximum likelihood value will be unity. The same goes for two-point tracks. Only when tracks become longer does estimation become necessary.

Returning back to the maximum likelihood expression for a hypothesis, taking the logarithm of both sides of Eqn. 3.4, since in Eqn. 3.2 the logarithm of the maximum likelihood is of interest,

$$\log\ l(\mathbf{x} \mid \theta) = \sum_{i=1}^{N_\theta} \log\ l_i(\mathbf{x}_i \mid \theta) \quad (3.14)$$

Combining Eqn. 3.12, Eqn. 3.13, and Eqn. 3.14,

$$\begin{aligned}\log\ l(\mathrm{x} \mid \theta) &= \sum_{i=1}^{N_\theta}(\log\ C_i - \frac{1}{2}J_i^\theta) \\ &= \log(\prod_{i=1}^{N_\theta} C_i) - \frac{1}{2}\sum_{i=1}^{N_\theta} J_i^\theta \\ &= \log(C) - \frac{1}{2}\sum_{i=1}^{N_\theta} J_i^\theta\end{aligned}$$

where

$$C = \prod_{i=1}^{N_\theta} C_i = \frac{1}{(2\pi \left| \ R \ \right|)^{\frac{M}{2}}}$$

and M, being the total number of measurements ever encountered, is, as was shown previously,

$$M = \sum_{i=1}^{N_\theta} N_{T_i} \ .$$

Since C is the same for every θ, it can be taken out of the criterion. So, the modified Akaike criterion, which is essentially, the original Akaike criterion minus $\log(C)$ is

$$Cost(\theta) = \sum_{i=1}^{N_\theta} J_i^\theta + 2k(\theta) \ . \tag{3.15}$$

So, the best estimate of θ is

$$\hat{\theta} = \arg_\theta \min\ (\sum_{i=1}^{N_\theta} J_i^\theta + 2k(\theta)) \ . \tag{3.16}$$

False alarms fit the data perfectly, and thus J_i^θ is zero. For a track with a missing measurement at time t_M, assume that

$$\underline{z}(t_M) = \underline{h}(\underline{\hat{x}}_i(t_M \mid \mathbf{z}_i), t_M) \ .$$

Thus, that component of J_i^θ will be zero, and the added free parameter necessary to describe the missing measurement will balance the cost. Breakup tracks require

more work since the time that breakup has occurred and the change of velocity need to be estimated before J_i^θ can be computed. The next section will describe the estimation of these parameters and computing J_i^θ for breakup targets in more detail.

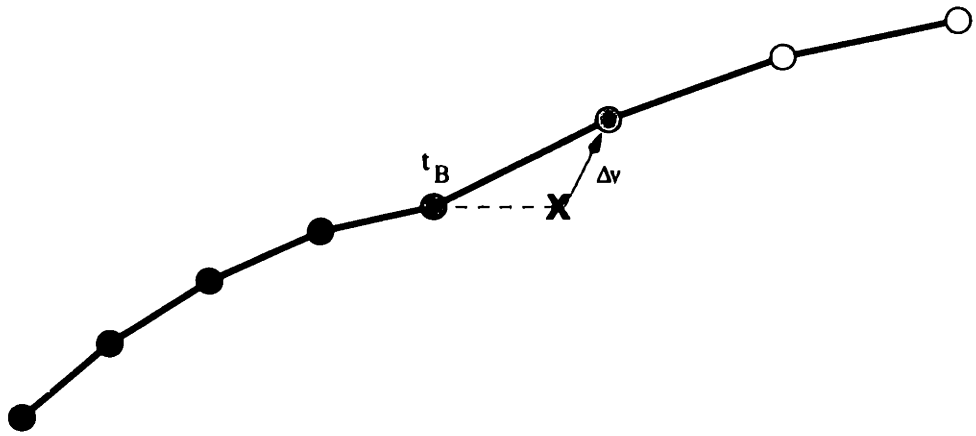
3.3.3 Target Breakup Detection and Parameter Estimation

To calculate the maximum likelihood of a track that breaks up into N_B breakup pieces, first consider the simpler problem where a track breaks up into one breakup piece. In this simpler scenario, there is a breakup time, t_B, and one ΔV which represents the change of state from the original track to the breakup piece's state as shown in Figure 3-4a. The x in the figure represents the next predicted position of the target if no ΔV were present. The casual definition of t_B being the breakup time needs to be more rigorously defined. The *exact* time of breakup, call it $T_{breakup}$, does not have to fall into the exact time of the scan, which is what t_B represents; $T_{breakup}$ could happen between scans. So, define t_B in the following way:

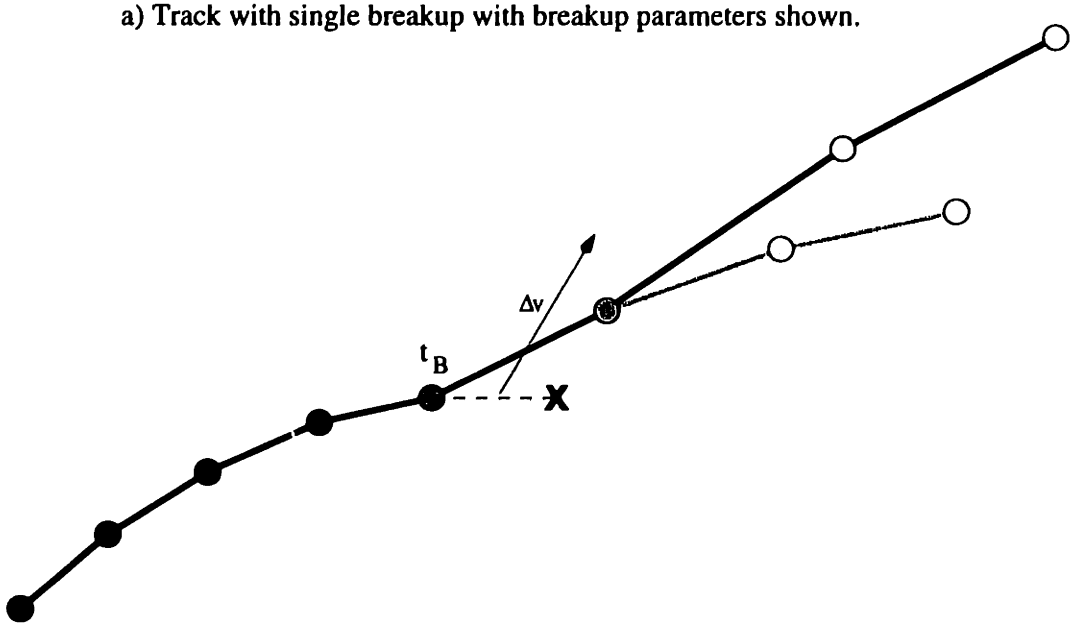
$$t_B \leq T_{breakup} < t_{B+1} \ .$$

Thus, breakup occurs either right at t_B or in that intermediary time between t_B and the next scan, t_{B+1}. Looking again at Figure 3-4a reveals that this track assumed that $T_{breakup} = t_B$ so that propagating the state gives the track file shown. Figure 3-4b shows what happens when $T_{breakup} = \frac{1}{2}(t_B + t_{B+1})$. The breakup piece's trajectory is different, and since the interval between the time when the target's trajectory gets changed by a ΔV, $T_{breakup}$, to the next scan time, t_{B+1}, is only half as long as that of Figure 3-4a, the magnitude of ΔV is approximately twice as large; the direction, however, remains the same. Thus, the trajectories of the breakup piece in Figure 3-4a and Figure 3-4b are different even though the measurements, $\{\underline{z}(t_0), \ldots, \underline{z}(t_{B+1})\}$, are the same.

Since $T_{breakup}$ lies in the range, $t_B \leq T_{breakup} < t_{B+1}$, estimating ΔV requires estimating $T_{breakup}$ also. Although estimating both parameters is possible, estimating the best $T_{breakup}$ along with ΔV is more precise than necessary for the purposes of



a) Track with single breakup with breakup parameters shown.



b) Track with breakup occuring in between scans with its projected breakup piece measurements. Breakup piece from a) shown for comparison.

Figure 3-4: Target with Single Breakup

target breakup detection. It suffices to determine at which scan breakup occurred rather than determining the exact time that breakup occurred. Thus, for the rest of this section, it will be assumed that

$$T_{breakup} = t_B \ .$$

If the time between t_B and t_{B+1} is small, the change in the maximum likelihood of the breakup track from approximating $T_{breakup}$ to be t_B will be small. The magnitude of ΔV is smallest when $T_{breakup}$ equals t_B, so the assumption on $T_{breakup}$ does not yield an excessively large magnitude of ΔV.

To derive an expression for the maximum likelihood of a breakup track with one breakup piece, start with its dynamics and measurements. The dynamics of a breakup track differ from the dynamics of an independent track only in that there is a sudden change of state, ΔV, at time t_B. Then, based on Eqn. 3.8 and Eqn. 3.9, the model for a breakup track is

$$\dot{\underline{x}}(t) = \underline{f}(\underline{x}(t), t) + \delta(t - t_B) \begin{bmatrix} 0 \\ \Delta V \end{bmatrix} \tag{3.17}$$

$$\underline{z}(t) = \underline{h}(\underline{x}(t), t) + \underline{v}(t) \tag{3.18}$$

where $\delta(\cdot)$ is the unit area impulse function, which is zero everywhere except when the argument is zero, and the subscript i has been left out for notational convenience. Again, $\underline{v}(t)$ is a zero mean, white Guassian noise process with spectral density R.

As before, define a function like $\underline{g}(\underline{x}(t_0), t)$ so that Eqn. 3.18 reduces to

$$\underline{z}(t) = \underline{b}(\underline{x}(t_0), \Delta V, t_B, t) + \underline{v}(t) \ . \tag{3.19}$$

The function $\underline{b}(\underline{x}(t_0), \Delta V, t_B, t)$ is the state transformed to the measurement space at time t for a track with initial condition $\underline{x}(t_0)$ and which undergoes breakup at time t_B with change of state ΔV. With these parameters set, it is obvious that prior to t_B, ΔV does not affect the track's state. One way of looking at the effect of ΔV is

that it imparts a new initial condition at time t_B after which the track follows the dynamics of Eqn. 3.8. Thus, the state, given an initial condition $\underline{x}(t_0)$, a breakup time t_B, and a change of velocity ΔV for the breakup piece, is

$$\underline{x}(t) = \begin{cases} \int_{t_0}^{t} \underline{f}(\underline{x}(t), t)dt + \underline{x}(t_0), & t_0 \leq t \leq t_{B_-}, \\ \int_{t_{B_+}}^{t} \underline{f}(\underline{x}(t), t)dt + \underline{x}(t_{B_+}), & t \geq t_{B_+} \end{cases} \tag{3.20}$$

where t_{B_-} refers to the time right before ΔV has perturbed the state and t_{B_+} refers to the time right after ΔV has perturbed the state. So,

$$\underline{x}(t_{B_+}) = \underline{x}(t_{B_-}) + \begin{bmatrix} 0 \\ \Delta V \end{bmatrix} .$$

Thus,

$$\underline{b}(\underline{x}(t_0), \Delta V, t_B, t) = \underline{h}(\underline{x}(t), t) \tag{3.21}$$

where $\underline{x}(t)$ is shown in Eqn. 3.20.

Referring back to Eqn. 3.19, if $\underline{x}(t_0)$ and ΔV are treated as parameters given the breakup time t_B, estimating the parameters is just the problem of estimating parameters in additive white Guassian noise. Thus, the likelihood function of this breakup track becomes, putting the subscript i back in,

$$l_i(\mathbf{x}_i \mid \theta) = C_i \exp[-\frac{1}{2} \sum_{k=0}^{N_{T_i}-1} \underline{\nu}_i(t_k)^T R^{-1} \underline{\nu}_i(t_k)] \tag{3.22}$$

where

$$\underline{\nu}_i(t_k) = \underline{z}_i(t_k) - \underline{b}_i(\underline{x}_i(t_0), \Delta V_i, t_B, t_k) ,$$

which is the residual between measured and predicted values of the measurement, and

$$C_i = \frac{1}{(2\pi \left| R \right|)^{\frac{N_{T_i}}{2}}}$$

and N_{T_i} is the number of measurements in this breakup track.

Maximizing $l_i(\mathbf{x}_i \mid \theta)$ is the same as minimizing the quadratic function

$$J_i^\theta = \sum_{k=0}^{N_{T_i}-1} \underline{\nu}_i(t_k)^T R^{-1} \underline{\nu}_i(t_k) \ . \tag{3.23}$$

So, the best estimate of the parameters is

$$\hat{\underline{x}}_i(t_0), \hat{\Delta V} = arg_{\underline{x}_i(t_0),\Delta V} min \ J_i^\theta(\underline{x}_i(t_0), \Delta V) \tag{3.24}$$

given a breakup time t_B and where the dependence of J_i^θ on the parameters are shown explicitly. Although estimating $\underline{x}_i(t_0)$ is not independent of estimating ΔV in general as shown in Eqn. 3.24, it is expected that the initial condition of the original track should not depend on whether a breakup occurs some time later in the track's trajectory. So, the following assumption is made:

- The estimate of the initial condition, $\underline{x}_i(t_0)$, depends only on those measurements of the track which have not been generated by a breakup piece since those pieces have been perturbed by a ΔV. These measurements are $\underline{z}(t_0), \ldots, \underline{z}(t_B)$.[3]

Then, it necessarily follows that:

- The estimate of ΔV is the same as the estimate of ΔV conditioned on the estimate of $\underline{x}_i(t_0)$. Thus, the effect of this assumption is that $\underline{x}_i(t_{B_-})$ is given when estimating $\underline{x}_i(t_{B_+})$.

Thus, making these assumptions, the estimate of $\underline{x}_i(t_0)$ is based on those measurements from the original track before breakup. So,

$$\hat{\underline{x}}_i(t_0) = \hat{\underline{x}}_i(t_0) \mid \underline{z}(t_0), \ldots, \underline{z}(t_B) = \arg_{\underline{x}_i(t_0)} \min \sum_{k=0}^{B} \underline{\nu}_i(t_k)^T R^{-1} \underline{\nu}_i(t_k)$$

[3] $\underline{z}(t_B) = \underline{z}(t_{B_-}) = \underline{z}(t_{B_+})$ since the effect of the ΔV is not apparent in the position part of the state at time t_B, which is the only part the measurement reveals.

where the dependence on the measurements are shown explicitly. Thus,

$$\hat{\underline{x}}(t) = \begin{cases} \int_{t_0}^{t} \underline{f}(\hat{\underline{x}}(t), t)dt + \hat{\underline{x}}(t_0), & t_0 \leq t \leq t_{B_-}, \\ \int_{t_{B_+}}^{t} \underline{f}(\hat{\underline{x}}(t), t)dt + \hat{\underline{x}}(t_{B_+}), & t \geq t_{B_+} \end{cases} . \tag{3.25}$$

Note that for $t_0 \leq t \leq t_{B_-}$, $\hat{\underline{x}}(t)$ is completely determined by $\hat{\underline{x}}(t_0)$. However, finding $\hat{\underline{x}}(t)$ for $t \geq t_{B_+}$ requires estimating ΔV since

$$\hat{\underline{x}}(t_{B_+}) = \hat{\underline{x}}(t_{B_-}) + \begin{bmatrix} 0 \\ \hat{\Delta V} \end{bmatrix} . \tag{3.26}$$

So, estimating ΔV,

$$\hat{\Delta V} = \hat{\Delta V} \mid \hat{\underline{x}}(t_0) = \arg_{\Delta V}\min(\sum_{k=0}^{N_{T_i}-1} \underline{\nu}_i(t_k)^T R^{-1} \underline{\nu}_i(t_k) \mid \hat{\underline{x}}(t_0)) .$$

Separating the sum at t_B,

$$\hat{\Delta V} \mid \hat{\underline{x}}(t_0) = \arg_{\Delta V}\min(\sum_{k=0}^{B} \underline{\nu}_i(t_k)^T R^{-1} \underline{\nu}_i(t_k) + \sum_{k=B+1}^{N_{T_i}-1} \underline{\nu}_i(t_k)^T R^{-1} \underline{\nu}_i(t_k) \mid \hat{\underline{x}}(t_0)) .$$

Since minimization over measurements $\underline{z}(t_0), \ldots, \underline{z}(t_B)$ is already captured in $\hat{\underline{x}}(t_0)$, the first sum on the right-hand side is constant and does not contribute to finding the minimum. Thus,

$$\hat{\Delta V} = \arg_{\Delta V}\min(\sum_{k=B+1}^{N_{T_i}-1} \underline{\nu}_i(t_k)^T R^{-1} \underline{\nu}_i(t_k) \mid \hat{\underline{x}}(t_0)) .$$

The estimate of ΔV depends only on the measurements particular to the breakup piece, $\underline{z}(t_{B+1}), \ldots, \underline{z}(t_{end_i})$, and the initial condition, $\hat{\underline{x}}(t_0)$. Because of this, extending the above calculation to several breakup pieces is natural. The initial condition estimate is still the same as before, but rather than having only one ΔV to estimate, there is a ΔV to estimate for each breakup piece.

Represent the measurements and states of the track in the following way. Let $\underline{z}_{i_0}(t)$ represent those measurements from the original track before breakup; therefore,

these measurements exist for times $\{t_0, \ldots, t_B\}$. The state associated with these measurements are $\underline{x}_{i_0}(t)$ which are similarly restricted to time $\{t_0, \ldots, t_B\}$.

Let $\underline{z}_{i_j}(t)$ represent those measurements from breakup piece j of track i after breakup; therefore, these measurements exist for times $\{t_{B+1}, \ldots, t_{end_i}\}$ where end_i has been defined in Eqn. 3.7. The associated state for these measurements are $\underline{x}_{i_j}(t)$, also restricted for times $\{t_{B+1}, \ldots, t_{end_i}\}$. Thus,

$$\underline{x}_{i_0}(t) = \int_{t_0}^{t} \underline{f}(\underline{x}_{i_0}(t), t)dt + \underline{x}_{i_0}(t_0), \quad t_0 \leq t \leq t_{B_-} \tag{3.27}$$

$$\underline{x}_{i_j}(t) = \int_{t_{B_+}}^{t} \underline{f}(\underline{x}_{i_j}(t), t)dt + \underline{x}_{i_j}(t_{B_+}), \quad t \geq t_{B_+} \tag{3.28}$$

where

$$\underline{x}_{i_j}(t_{B_+}) = \underline{x}_{i_0}(t_{B_-}) + \begin{bmatrix} 0 \\ \Delta V_j \end{bmatrix} \tag{3.29}$$

and where ΔV_j refers to the ΔV for the $j^{\underline{th}}$ breakup piece. Furthermore,

$$\underline{b}_{i_0}(\underline{x}_{i_0}(t_0), t) = \underline{h}(\underline{x}_{i_0}(t), t), \quad t_0 \leq t \leq t_{B_-} \tag{3.30}$$

$$\underline{b}_{i_j}(\underline{x}_{i_0}(t_0), \Delta V_j, t_B, t) = \underline{h}(\underline{x}_{i_j}(t), t), \quad t \geq t_{B_+} \tag{3.31}$$

where $\underline{b}_0(\cdot)$ is the track's state before breakup transformed to the measurement space and $\underline{b}_j(\cdot)$ is the track's $j^{\underline{th}}$ breakup piece's state transformed to the measurement space.

Thus, if track i breaks up into N_{B_i} pieces at time t_B, begins at time t_0, and ends at time t_{end_i},

$$J_i^{\theta} = \sum_{k=0}^{B} \underline{\nu}_{i_0}(t_k)^T R^{-1} \underline{\nu}_{i_0}(t_k) + \sum_{j=1}^{N_{B_i}} \sum_{k=B+1}^{end_i} \underline{\nu}_{i_j}(t_k)^T R^{-1} \underline{\nu}_{i_j}(t_k) \tag{3.32}$$

where $\underline{\nu}_{i_0}(t)$ is the residual of the original track before breakup and $\underline{\nu}_{i_j}(t)$ is the residual of the $j^{\underline{th}}$ breakup piece of track i and equal to

$$\underline{\nu}_{i_0}(t) = \underline{z}_{i_0}(t) - \underline{b}_{i_0}(\underline{x}_{i_0}(t_0), t)$$

$$\underline{\nu}_{i_j}(t) = \underline{z}_{i_j}(t) - \underline{b}_{i_j}(\underline{x}_{i_j}(t_0), \Delta V_j, t_B, t)$$

Thus,

$$\hat{\underline{x}}_{i_0}(t_0) = arg_{\underline{x}_{i_0}(t_0)} min \sum_{k=0}^{B} \underline{\nu}_{i_0}(t_k)^T R^{-1} \underline{\nu}_{i_0}(t_k)$$

$$\hat{\Delta V}_j = arg_{\Delta V_j} min \sum_{k=B+1}^{end_i} \underline{\nu}_{i_j}(t_k)^T R^{-1} \underline{\nu}_{i_j}(t_k) \ .$$

With these parameters estimated, the maximum likelihood expression is

$$J_i^\theta = \sum_{k=0}^{B} \hat{\underline{\nu}}_{i_0}(t_k)^T R^{-1} \hat{\underline{\nu}}_{i_0}(t_k) + \sum_{j=1}^{N_{B_i}} \sum_{k=B+1}^{end_i} \hat{\underline{\nu}}_{i_j}(t_k)^T R^{-1} \hat{\underline{\nu}}_{i_j}(t_k) \tag{3.33}$$

where

$$\hat{\underline{\nu}}_{i_0}(t) = \underline{z}_{i_0}(t) - \underline{b}_{i_0}(\hat{\underline{x}}_{i_0}(t_0), t)$$

$$\hat{\underline{\nu}}_{i_j}(t) = \underline{z}_{i_j}(t) - \underline{b}_{i_j}(\hat{\underline{x}}_{i_j}(t_0), \hat{\Delta V}_j, t_B, t)$$

This J_i^θ for a breakup track can now be plugged into the cost function, Eqn. 3.15, to get the cost of a hypothesis, θ, that consists of breakup tracks.

3.3.4 Effect on MHT Formulation

Akaike's criterion has replaced the score function for hypotheses and tracks in the MHT formulation so that subjective *a priori* probability densities are no longer necessary in determining the better hypotheses and tracks. Only with a criterion based on some theoretical foundation is target breakup detection able to be handled convincingly since a target breakup detection algorithm that requires lots of tweaking of various parameters would not be theoretically justified. Furthermore, since the target breakup detection algorithm is a sort of intermediary model in terms of complexity between the continuation of a single track model and the several independent tracks model, what parameters to tweak and how to tweak them is unclear. Akaike's criterion gives the theoretical justification for the scores given to various tracks and hypotheses, and simulations of its use in the MHT formulation are shown in Chapter 5.

Extending the MHT formulation to include target breakup will undoubtedly necessitate more computation. The question is how much more computation is required over the original MHT formulation's computation requirement. First, consider computation time. Hypothesizing each measurement to be the breakup piece of some target requires looping over the existing tracks. Since hypothesizing a measurement to be the continuation of an existing track already loops over the existing tracks, hypothesizing a measurement to be the breakup piece of an existing track can be included in this loop. However, since breakup requires tracking the breakup pieces, a process similar to the computationally expensive track initiation procedure, the cost in computation time is significant.

Now, considering computation memory, since each measurement can be hypothesized to be one of four possibilities rather than one of three, the hypothesis tree is even larger than before and pruning becomes a more crucial factor in the correct operation of this tracker. Taking a closer look at each possibility for a measurement, namely false alarm, start of a new track, continuation of an existing track, and start of a breakup piece, reveals that the last two possibilities, track continuation and breakup, are actually a composite of several possibilities since continuation and breakup stem from some previously existing track. Thus, the expansion of the hypothesis tree is a significant one.

Track initiation is the most computationally complex possibility in the original MHT formulation because a new track can be continued by just about every measurement in the next scan. Gating does not really help discriminate which measurements are more likely to have originated from the new track because the velocity part of the track's state vector is unknown, which makes the gate of this new independent track necessarily large. Fortunately, the start of a new track is not a composite of several possibilities like continuation and breakup, so the number of times track initiation is considered a possibility is equal to the total number of measurements ever encountered. In the target breakup detection incorporated MHT formulation, a measurement being hypothesized to be the start of a breakup piece is similar to initiating a track that is only slightly more constrained than the start of an independent track.

The measurement is essentially being hypothesized to be the second point of a new track, where the first point is the last point of some existing track. On the other hand, in track initiation, the measurement is hypothesized to be the first point of an independent track. Thus, the computational complexity for breakup and track initiation is similar. However, unlike track initiation, the number of breakup possibilities for a measurement is dependent on the number of tracks that existed in the previous scan. Thus, computation complexity has increased considerably by adding target breakup detection in terms of both time and memory.

Chapter 4

Implementation of a MHT with Target Breakup Detection

Incorporating target breakup detection into the MHT formulation requires modification to the basic flow of the algorithm and the scores for hypotheses. The next two sections describe these modifications in detail. The last section describes a modified pruning algorithm along with reasons why such an algorithm is both necessary and appropriate.

The use of the word *breakup track* refers to a different entity in this chapter than it did in the previous chapter. There, a breakup track is the entire collection of the original independent track that broke up at some time, t_B, and the tracks of its several constituent breakup pieces. Theoretically, it is easier to consider breakup tracks in this way so that calculating the maximum likelihood and free parameters is simpler. However, implementationally speaking, it is easier to consider the breakup pieces of the breakup track as the breakup tracks themselves. Using this definition of a breakup track, the original track which began as an independent track remains an independent track. Breakup tracks are similar to independent tracks in that once they have been initiated, they can be continued and possibly spawn breakup tracks themselves. The difference between breakup tracks and independent tracks lie primarily in the initiation procedure:

- **Independent tracks** are initiated by allowing the first measurement to correlate with any measurement in the next scan, so long as the predicted velocity of such a correlation is not beyond credible speeds.

- **Breakup tracks** are assumed to have originated from a previously existing track, where the initial state of the breakup track is determined by the state of the previously existing track plus some change of velocity, ΔV.

The initiation of a breakup track is more constraining than that of an independent track because starting from the very first measurement, the velocity part of the state can be estimated due to the knowledge that a breakup track spawned from a previously existing track. The initial state of an independent track has no constraints on its velocity components because it is not until the *second* measurement is obtained that the velocity can be estimated. Section 4.2.2 describes and justifies the initiation procedure in more detail.

4.1 Flow Modification

Since target breakup detection attributes an additional possibility, the start of the breakup of a track, to each measurement, the basic flow is augmented to include this possibility. Figure 4-1 shows the flow diagram of Figure 2-7 modified to include the possibility of target breakup. The modified part of the flow is denoted by the thick-lined boxes. Incorporating target breakup detection into the MHT formulation only requires adding two more modules into the original flow diagram. The basic flow of the rest of the algorithm can be kept as is with only slight modifications within certain modules to take into account the new scoring function and pruning algorithm.

The possibility of a measurement being the breakup of an existing track has been added within the condition that the measurement lies in the gate of the existing track. This may seem like an unnecessary constraint for the start of a breakup track. However, the purpose of gating is to separate those measurements that could have originated from the track with those that could not. A breakup piece can initially

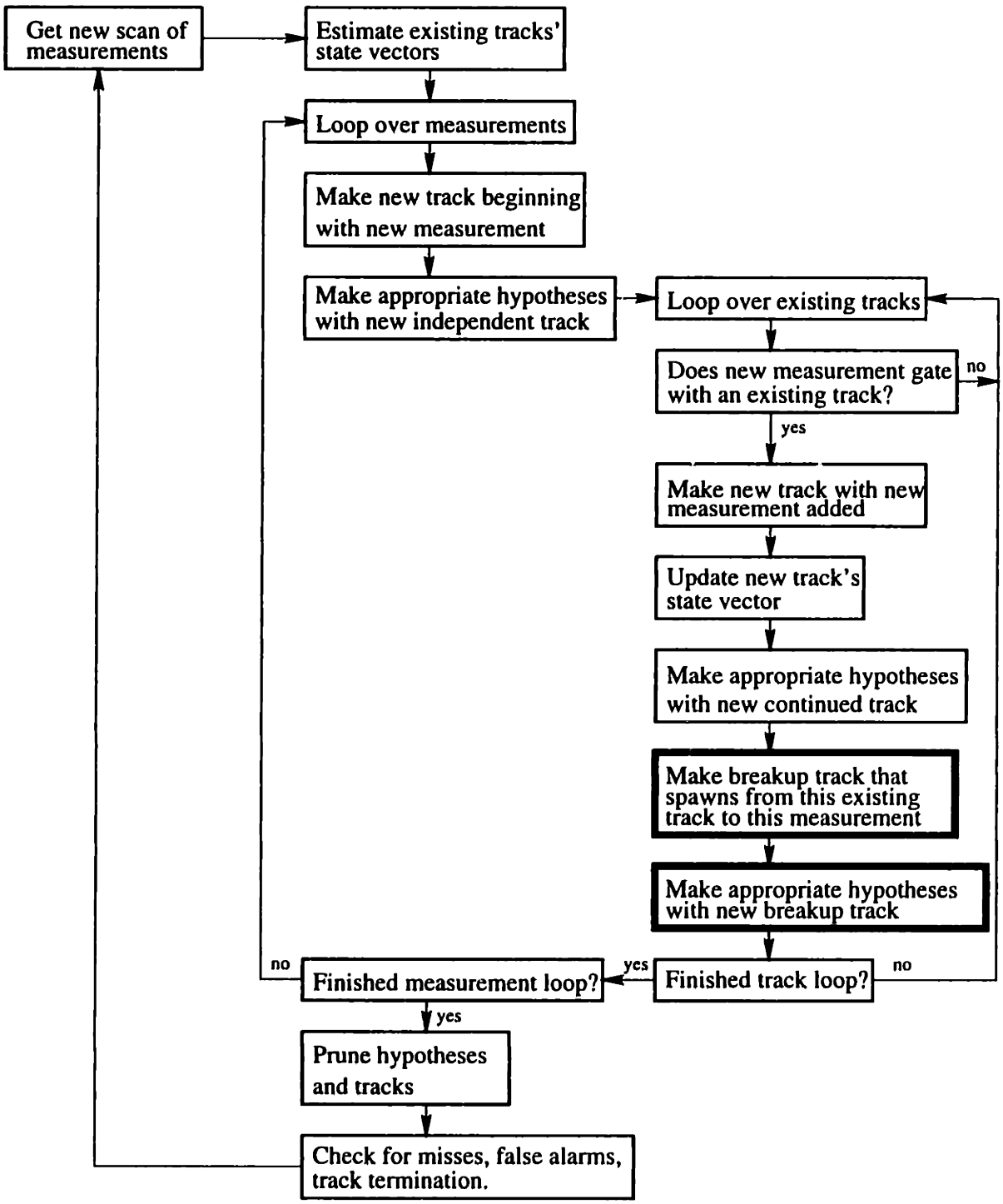


Figure 4-1: MHT Flow Diagram modified to include Target Breakup Detection

only go so far away from the track it originated, and thus, the first measurement from this breakup piece would be expected to appear close to the original track's trajectory and in the original track's gate. This constraint is consistent with how a breakup track behaves, and furthermore, by considering only those measurements within a track's gate for breakup, the algorithm is more efficient.

4.2 Score Function Modification

In determining the score for tracks and, ultimately, for hypotheses, two separate calculations are necessary: the number of free parameters for that particular kind of track and the maximum likelihood of the data to the prediction. Since the score for a hypothesis is the sum of the scores of its constituent tracks, how to go about calculating the score for a track is presented with the understanding that calculating the score for a hypothesis is then straightforward.

4.2.1 Free Parameter Calculation

The number of free parameters for each kind of track was given in Table 3.1. So, the ultimate goal of the free parameter calculation is to give tracks the correct number of free parameters while following the flow diagram of Figure 4-1. Thus, each module will be described in terms of its effect on the free parameters of a track.

The first module where free parameters are computed is the "Make new track beginning with new measurement" module. This module initiates new independent tracks. So, any tracks created in this module should be given seven free parameters for the independent track complexity.

The "Make new track with new measurement added" module, which is where tracks are continued, does not change any free parameters of any tracks because continuing a track does not change the complexity of that track. An independent track that is continued remains an independent track. Likewise, a breakup track that is continued also remains a breakup track.

The next module where free parameters of tracks are changed is the "Make

breakup track ..." module. A breakup track is given three free parameters. Furthermore, for each group of breakup tracks that originated from the same previously existing track, an extra free parameter is necessary to account for the breakup time. Although this requires some extra effort, like searching over the breakup tracks and looking at their origins, it is a simple operation. From Table 3.1, it may seem that the original track's free parameters needs to be changed; however, it is not always necessary that the original track terminates. Anyhow, the next module will take care of modifying the number of free parameters for tracks that terminate.

The "Check for misses, false alarms, track termination" module will modify the number of free parameters for tracks that have misses, are false alarms, or terminate. Any track that is given a miss for this scan gets an additional free parameter to denote the time of this miss. A one-point track that has enough misses to consider it a false alarm is given a total of three free parameters regardless of how many free parameters it may have had before. Tracks that terminate are given an additional free parameter to what it may have had before. This not only includes termination of independent tracks but also termination of breakup tracks which is correct since the termination of a breakup track also requires one additional free parameter.

By modifying the modules in the flow diagram in this way, the number of free parameters associated with the different types of tracks matches the number shown in Table 3.1.

4.2.2 Maximum Likelihood Calculation

The extended Kalman filter was used in the original MHT formulation to calculate the next predicted position of a track and to determine how well the data fit the track. The major difference between the original MHT formulation and the target breakup detection incorporated MHT formulation in terms of likelihood, is that the original MHT formulation based the fit to the data on *filtered* values of the track position whereas the target breakup detection incorporated MHT formulation bases its fit to the data on *smoothed* values of the track position.

The smoothed values of the track position is the result of finding the nonlinear least

squares estimate of the track position. Several algorithms have been presented in the literature, including those that calculate J_i^θ recursively for every new measurement to be included in the track, as in [7]. The problem with using smoothed estimates is that there is considerably more computation necessary to calculate smoothed estimates than filtered estimates. Since it is desirable to make these calculations as simple as possible, if filtered estimates do not deviate too far from smoothed estimates, approximating the maximum likelihood by using filtered estimates may be suitable.

The extended Kalman filter returns these filtered estimates at every scan, but it must be initialized with some initial value of the state; thus, this is the track initiation problem. For an independent track, the estimate of the initial condition, $\underline{\hat{x}}_i(t_0)$, is the estimate that minimized the cost function, J_i^θ, in Eqn. 3.13. For the aims of reducing computation but making the most accurate estimate of the initial state as possible, initialize the filter with the minimal number of measurements necessary to estimate an initial condition. For this case, this requires two measurements since the state includes a velocity part. So, looking at just the first two measurements, the cost function is

$$\begin{aligned} J_i^\theta &= (\underline{z}_i(t_0) - \underline{g}(\underline{x}_i(t_0), t_0))^T R^{-1} (\underline{z}_i(t_0) - \underline{g}(\underline{x}_i(t_0), t_0)) \\ &\quad + (\underline{z}_i(t_1) - \underline{g}(\underline{x}_i(t_0), t_1))^T R^{-1} (\underline{z}_i(t_1) - \underline{g}(\underline{x}_i(t_0), t_1)) \ . \end{aligned}$$

Upon close inspection, the cost function can be made to equal zero for any two measurements, $\underline{z}_i(t_0)$ and $\underline{z}_i(t_1)$. The initial state that makes J_i^θ go to zero satisfies the following two conditions:

$$\begin{aligned} \underline{h}(\underline{x}_i(t_0), t_0) &= \underline{z}_i(t_0) \\ \underline{h}(\int_{t_0}^{t_1} \underline{f}(\underline{x}_i(t), t) dt + \underline{x}_i(t_0), t_1) &= \underline{z}_i(t_1) \end{aligned}$$

where Eqn. 3.10 was substituted in for $\underline{g}(\cdot)$. Thus, the position part of $\underline{x}_i(t_0)$ when transformed to the measurement space is equal to $\underline{z}_i(t_0)$ and the velocity part of $\underline{x}_i(t_0)$ moves the track's position so that at time t_1, the track's position when transformed to

the measurement space is equal to $\underline{z}_i(t_1)$. The resulting $\underline{x}_i(t_0)$ can be used to initialize the extended Kalman filter for this track.

For initiation of a breakup track, or in Chapter 3 lingo, a breakup piece, the estimate that requires the least computation but is fairly accurate needs the state of the original track before breakup, $\underline{x}_{i_0}(t_{B-})$, and the first measurement of the breakup track, $\underline{z}_{i_j}(t_{B+1})$. Thus, the sought estimate is $\hat{\underline{x}}_{i_j}(t_{B+1})$ given $\hat{\underline{x}}_{i_0}(t_{B-})$ and $\underline{z}_{i_j}(t_{B+1})$. The notation above is taken from the latter half of Section 3.3.3. From Eqn. 3.29, choose ΔV_j so that integrating $\hat{\underline{x}}_{i_j}(t_{B+})$ to time t_{B+1} gives the $\hat{\underline{x}}_{i_j}(t_{B+1})$ such that, when converted to the measurement space, is $\underline{z}_{i_j}(t_{B+1})$. Then, the initial state for the breakup track is $\hat{\underline{x}}_{i_j}(t_{B+1})$. Notice that the initial state of a breakup track is more constrained than an independent track because it only takes one measurement to initiate a breakup track and two measurements to initiate an independent track. Of course, this constraint originates from the fact that breakup tracks originate from some previously existing track.

After initiating the independent or breakup track, the maximum likelihood part of the score function is then the sum of the χ^2 values of the filtered residual for every measurement in the track. This is what has been done in the implementation of this algorithm and results of the simulations are shown in Chapter 5.

4.3 Hypothesis Pruning Modification

Section 2.3.3 described two methods to hypothesis pruning. Method 2 was chosen as the better pruning method since it always kept *multiple* hypotheses after each scan. However, if more than the set number of hypotheses to be kept, N_H, have similar scores, then hypotheses with comparable scores are pruned inadvertantly. In the original MHT formulation, this situation arises only when several tracks are initiated in the same scan. New tracks' scores are not different until they contain three or more points since the maximum likelihood of a one-point track (false alarm) and a two-point track is equal to unity. Likewise, when targets break up, the scan suddenly requires initiating many breakup tracks. Thus, in the scenario where targets may

break up, this situation is *expected* to arise and must be dealt with by a suitable pruning method.

A possible solution to this problem is to keep more than N_H hypotheses if the scores of the hypotheses are "close". It is difficult to determine quantitatively what "close" is, so a more careful examination is required to determine which parts of the scores are more important.

The situation when several competing hypotheses arise comes from the fact that not enough measurements have been obtained to give differences in the maximum likelihood part of the score function. Furthermore, when a breakup has occurred, suddenly there are several hypotheses with the same complexity, and likewise when several independent tracks have appeared in the scan volume. Thus, the complexity part of the score seems to be the most relevant part of the score to key off of when deciding how many hypotheses to keep.

Thus, the proposed solution to pruning, which will be called **Method 3**, is as follows:

1. Rank all the hypotheses, lowest score first since the lowest score is the best hypothesis.

2. Keep the first N_H hypotheses, unless there isn't N_H hypotheses to keep, in which case pruning is finished.

3. Keep the next hypothesis if its complexity is less than or equal to the best hypothesis's complexity.

4. Repeat step 3 if an extra hypothesis was kept. Otherwise, end.

Comparing hypothesis complexity with the best hypothesis makes sense because in the case when the $N_H{}^{\underline{th}}$ hypothesis is much worse than the best hypothesis, comparing with the $N_H{}^{\underline{th}}$ hypothesis makes no sense since extra hypotheses are not necessary. Results of simulations comparing the pruning algorithms, Method 2 and Method 3, are given in Chapter 5.

Chapter 5

Simulation Results and Discussion

To demonstrate the performance of the target breakup incorporated multiple hypothesis tracker, several simulations have been run with various scenarios of targets and breakups. The data for all simulations presented in this section have been generated with the following attributes:

- target dynamics do not include drag effects;
- measurements have not been corrupted by noise;
- false alarms have a uniform density throughout the scan volume; and
- target breakup pieces' magnitude of ΔV is a uniform density from 0 up to some maximum assumed speed; the direction is uniformly distributed around a sphere.

Six sets of simulations have been performed to exhibit various scenarios that could occur. Five of the simulations focus on testing if target breakup is detected correctly; the last one focuses on comparing pruning algorithms. Figures 5-1 - 5-5 show an approximate picture of what the output graph of these simulations should look like. The simulations testing correctness of target breakup are:

- *A. Base Case* One track breaks up into ten pieces (See Figure 5-1). This is to test target breakup isolated from other "distractions" in the scan volume. The

rest of the simulations will add certain "distractions" to see how they affect target breakup detection.

- *B. Crossing Tracks* Two tracks initially in the scan volume cross in the second scan (See Figure 5-2). One of the tracks breaks up into ten pieces.

- *C. Continuing vs. Breakup Tracks* Nine tracks are initially in the scan volume moving generally parallel with each other. One of the tracks breaks up into five pieces (See Figure 5-3). The tracker is tested for handling several continuing tracks, in the midst of breakup pieces.

- *D. Independent vs. Breakup Tracks* Two tracks initially in the scan volume move in parallel, and one breaks up into five pieces. Around the time of breakup, specifically t_B and t_{B+1}, independent tracks also form (See Figure 5-4). Thus, the tracker is tested to distinguish independent tracks from breakup tracks.

- *E. False Alarms* There are four simulations in this group. The data is the same as in the base case, A, up above, but starting with the third scan, five, ten, fifteen, or twenty false alarms are present in the every scan (See Figure 5-5). Here, the tracker is tested to see how many false alarms it can handle before making errors.

All the above simulations were run with the pruning algorithm, Method 3, presented in Section 4.3. For comparison, an implementation with pruning algorithm, Method 2, from Section 2.3.3 was also run to get the last set of simulations:

- *F. Pruning Comparison* Two simulations with pruning algorithm, Method 2, running on the data from case B and case C. The number of hypotheses kept after each scan, N_H, is 500.

Pruning occurs, ideally, only at the end of a completed scan. However, since the maximum number of hypotheses ever kept is limited by the available memory, where in this implementation that maximum number is 1000, hypotheses must be pruned during the analysis of a scan because each scan generates over thousands of hypotheses,

many of which never survive to the end of the scan. Thus, the pruning algorithms, Method 2 and Method 3 referred to above, prunes those 1000 hypotheses that survive to the end of a scan.

5.1 Breakup Track Detection Simulations

Tracker Output Description

Output of the tracker comes in two forms:

- a tabular summary which is a list of hypotheses ranked by score where each hypothesis is a list of tracks, along with a track list showing the measurements that compose the track (See Figures 5-12 - 5-16).

- a graph of Elevation vs. Azimuth and/or Elevation vs. Time showing measurements as squares and predicted position as x's (See Figures 5-6 - 5-11). Tracks are denoted by lines from x's to x's.[1] All measurements ever encountered are shown in these graphs. The time of a measurement is not shown explicitly in the Elevation vs. Azimuth graph; however, the time can usually be inferred from the the tracks.

In the track summary part of the tabular summary, -1's refer to missing measurements in the track. The "Status" column refers to the kind of track which aids in determining the number of free parameters associated with that track. The codes are:

- C - confirmed, an independent track with two or more measurements;

- B - breakup, a breakup piece track;

- U - unassigned, a one-point track whose initial velocity has not been estimated;

- F - false alarm, a false alarm;

[1] Note that since measurements are unperturbed by noise the x's match exactly the boxes of the next measurement if the next measurement exists.

- T - tentative, a one-point or two-point track with initial velocity estimated;
- L - lost track, an independent track that has terminated; and
- D - lost breakup track, a breakup piece track that has terminated.

The number following "orig:" for breakup tracks refers to the measurement from which the breakup track has spawned.

The hypothesis summary lists hypotheses by rank and lists its constituent tracks. The cost and complexity of each hypothesis is also shown.

One important note about the graph is that the best hypothesis at each scan is drawn based on the best hypothesis from the previous scan, and thus, the graph shows a sort of "filtered" picture of the tracker's analysis as opposed to a "smoothed" picture. So, for example, if at scan S_1, there are two measurements, M_1 and M_2, and at scan S_2, there are two measurements, M_3 and M_4. Four possible tracks at scan S_2 are:

- T_1 with measurements M_1 and M_3,
- T_2 with measurements M_2 and M_4,
- T_3 with measurements M_1 and M_4, and
- T_4 with measurements M_2 and M_3,

and two possible hypotheses are:

- H_1 with tracks T_3 and T_4, and
- H_2 with tracks T_1 and T_2.

H_1 and H_2 will have the same score because the tracks only have two measurements in them, so assuming that H_1 was ranked first, the graph will draw a line from M_1 to M_4 and a line from M_2 to M_3. Now, suppose at scan S_3, there are two measurements, M_5 and M_6. Four possible tracks at scan S_3 are[2]:

[2]The tracks listed here are, by no means, the only tracks possible. They are just four of many.

- T_1 with measurements M_1, M_3, M_5
- T_2 with measurements M_2, M_4, M_6
- T_3 with measurements M_1, M_4, M_5, and
- T_4 with measurements M_2, M_3, M_6,

and two possible hypotheses are:

- H_1 with tracks T_3 and T_4, and
- H_2 with tracks T_1 and T_2.

At this scan, the tracks contain three measurements, and the hypotheses will have different scores. If H_2 is now the best hypothesis, a line will be drawn from M_3 to M_5 and another line from M_4 to M_6. Thus, the resulting graph will have a line from M_1 to M_4 to M_6 and a line from M_2 to M_3 to M_5 even though the best hypothesis after scan S_3 says there should be a line from M_1 to M_3 to M_5 and a line from M_2 to M_4 to M_6.

Therefore, the graph does not show the tracks of the best hypothesis at the end of the simulation; it shows a steady progression of the tracks from the best hypothesis at each scan.

A. Base Case

Figure 5-6 shows ten breakup pieces fanning out after the fifth scan. With no noise corrupting the measurements, the track initiation and breakup track initiation procedure as described in Section 4.2.2 suffices to give the exact ΔV for each breakup piece. Thus, the predicted positions of the breakup pieces fall on top of the measurements.

Looking at the best hypothesis at the end of this tracking period in Figure 5-12, the breakup pieces have indeed been identified as breakup tracks rather than independent tracks. The number labelled as "orig:" next to each breakup track refers to the last measurement of the original track from which the breakup piece spawned. This case demonstrates that the target breakup scenario in its simplest form can be handled by this tracker.

B. Crossing Tracks

The two tracks which cross in the second scan have been identified as two independent tracks that cross. Adding target breakup detection has not messed up the tracking of independent tracks. Figure 5-7 shows these two tracks crossing near the bottom of the graph with one track breaking up at a later time.

In the best hypothesis of Figure 5-13, track 1 contains all the odd numbered measurements up until the breakup time, and track 4 all the even numbered measurements up until the breakup time. This is how the data was generated, so this scenario has been tracked correctly.

C. Continuing vs. Breakup Tracks

The breakup pieces in this simulation are not far away from the independent tracks as in case B as can be seen in Figure 5-8. At no scan during the simulation does the breakup affect the tracking of the independent tracks. The latter half of the independent tracks were not mistaken for breakup tracks because of the strong correlation those measurements had with the established independent tracks.

The final best hypothesis of this simulation, as shown in Figure 5-14, is indeed a collection of nine continuing tracks and five breakup tracks.

D. Independent vs. Breakup Tracks

An independent track has more degrees of freedom than a breakup track because an independent track is not constrained to originate from some already existing track. The independent tracks that begin at t_B and t_{B+1} are not identified as breakup tracks by the tracker since no ΔV from the original track would allow these tracks to propogate upwards as they do in Figure 5-9 (the two trajectories on the left). It is difficult to discriminate the tracks from one another and to tell the time of when the tracks have formed in Figure 5-9, so Figure 5-10 plots the same tracks on different axes, elevation vs. time.

The independent tracks that start at time t_B and t_{B+1} are not considered breakup

tracks as the hypothesis listing, Figure 5-15, shows.

E. False Alarms

There are four simulations run in this set.

- *E1. 5 FA's* Base case with five false alarms.
- *E2. 10 FA's* Base case with ten false alarms.
- *E3. 15 FA's* Base case with fifteen false alarms.
- *E4. 20 FA's* Base case with twenty false alarms.

Figure 5-11 shows the result of case E1, and as can be seen, it is too messy to tell much about what is happening; however, the breakup tracks along the upper right of the graph seem to be identified correctly.

The list of hypotheses and tracks of Figure 5-16 do indeed show that breakup tracks have been identified. Track 31, the original track in the scan, contains the measurements that were generated for it, as do the breakup tracks that spawn from it. Notice that some of the false alarms have been hypothesized to be two-point lost tracks. As mentioned previously in Section 3.3.1, describing two measurements as a two-point lost track is indistinguishable from describing it as two false alarms.

As more false alarms are added at every scan, as in cases E2, E3, and E4, the tracker will eventually begin to err because the intermediary pruning that occurs, since the maximum number of hypotheses ever kept is 1000, will begin to prune some of the correct hypotheses. Simulation E2 tracks correctly until scan 7, after which the correct hypothesis gets inadvertantly pruned by the intermediary pruning process. Simulation E3 tracks less scans correctly than E2, and by the time there are twenty false alarms per scan, as in E4, only four scans are tracked correctly.

5.2 Pruning Comparison Simulations

Pruning Method 2 keeps $N_H = 500$ hypotheses per scan. Pruning Method 3 keeps 10 hypotheses per scan unless it is necessary to keep more.

F1. Pruning Comparison with Data of Case B

Table 5.1 shows the number of hypotheses kept after each scan dependent on the pruning method chosen.

Scan	Method 2	Method 3
1	7	7
2	87	10
3	500	10
4	500	10
5	500	10
6	500	10
7	500	10
8	500	10
9	500	20

Table 5.1: Pruning Comparison for Simulation F1

Although N_H of Method 2 could have been chosen to be 20 since that is the largest number of hypotheses kept by Method 3, there is no way to know beforehand how many hypotheses to keep after each scan. Since breakup spawns several breakup tracks that have the same score initially, a large value for N_H was chosen, although in this case, that large number was unnecessary.

F2. Pruning Comparison with Data of Case C

Table 5.2 shows the number of hypotheses kept after each scan dependent on the pruning method chosen.

In this case the choice of $N_H = 500$ for Method 2 is not large enough to detect target breakups. Furthermore, notice that similar performance of tracking is obtained from scan 1 to scan 12 by keeping less than 500 hypotheses.

Scan	Method 2	Method 3
1	500	10
⋮	⋮	⋮
9	500	10
10	500	18
11	500	28
12	500	289
13	500	852
14	500	1000
15	500	1000
16	500	1000
17	500	1000

Table 5.2: Pruning Comparison for Simulation F2

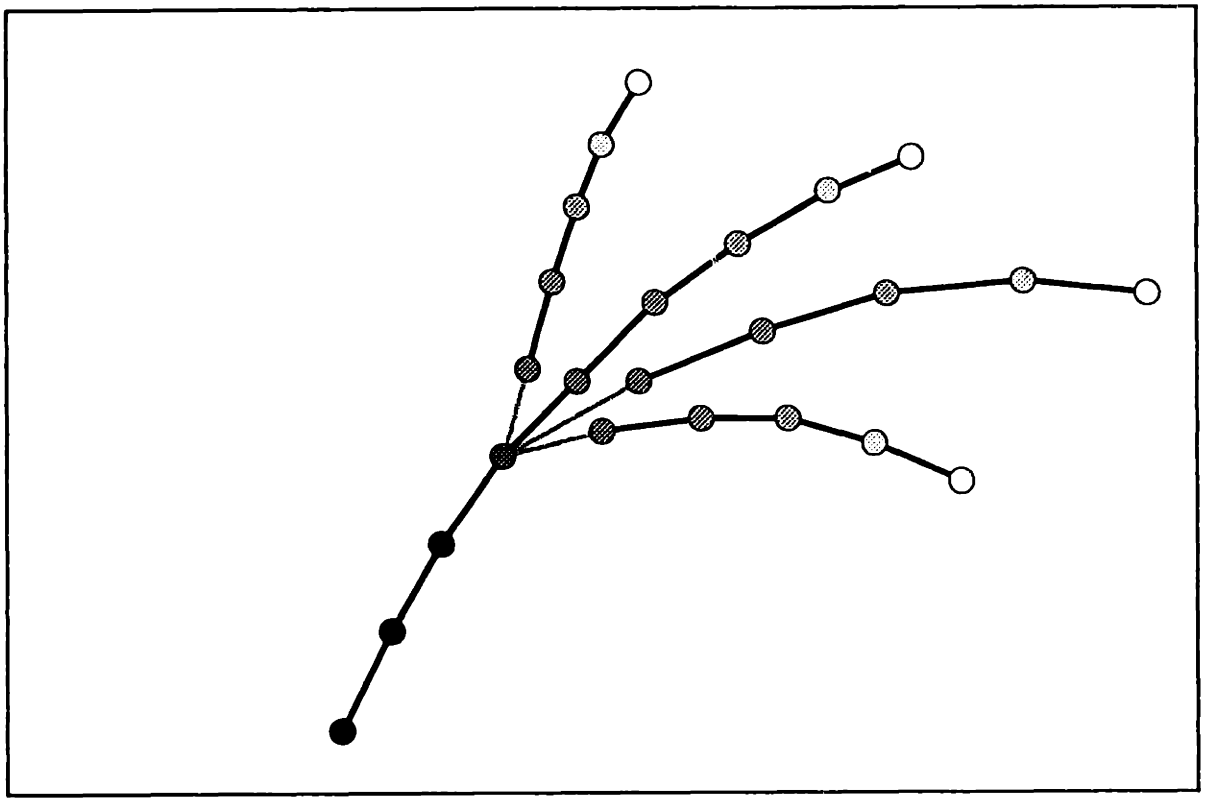


Figure 5-1: Conceptual Visualization of Simulation A

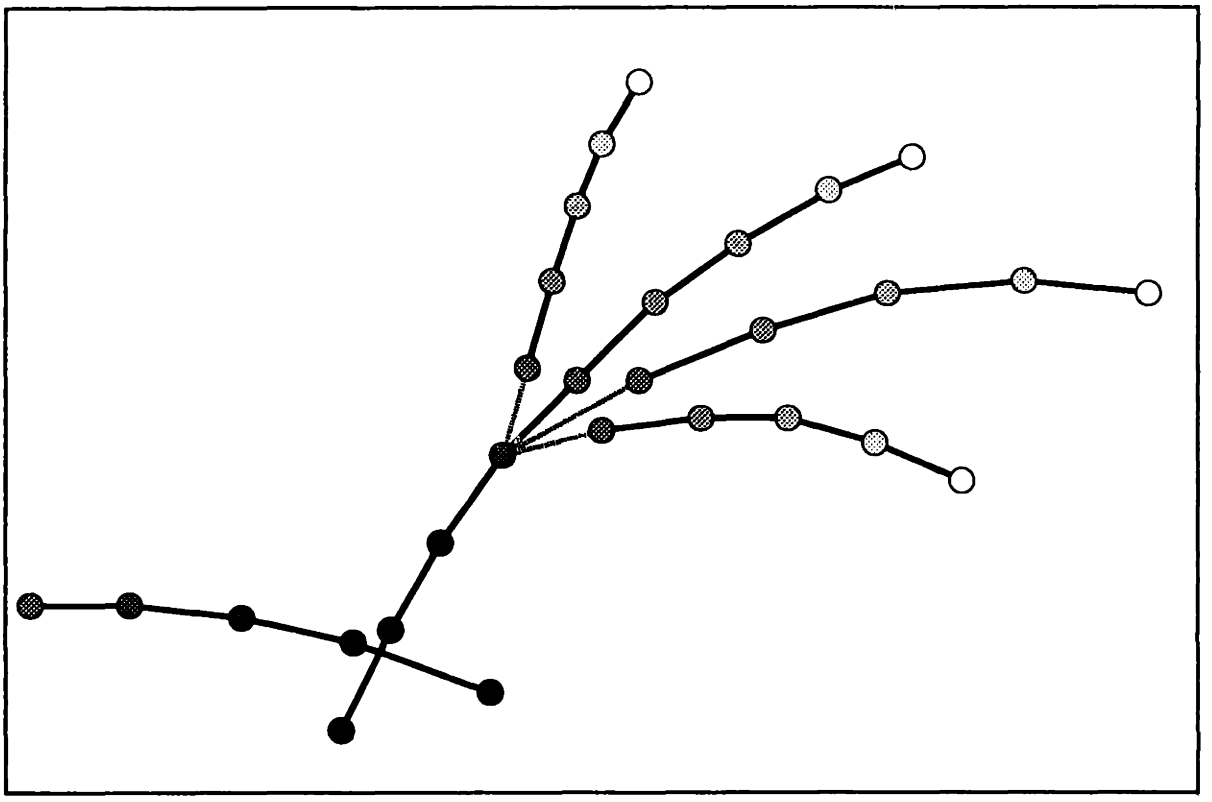


Figure 5-2: Conceptual Visualization of Simulation B

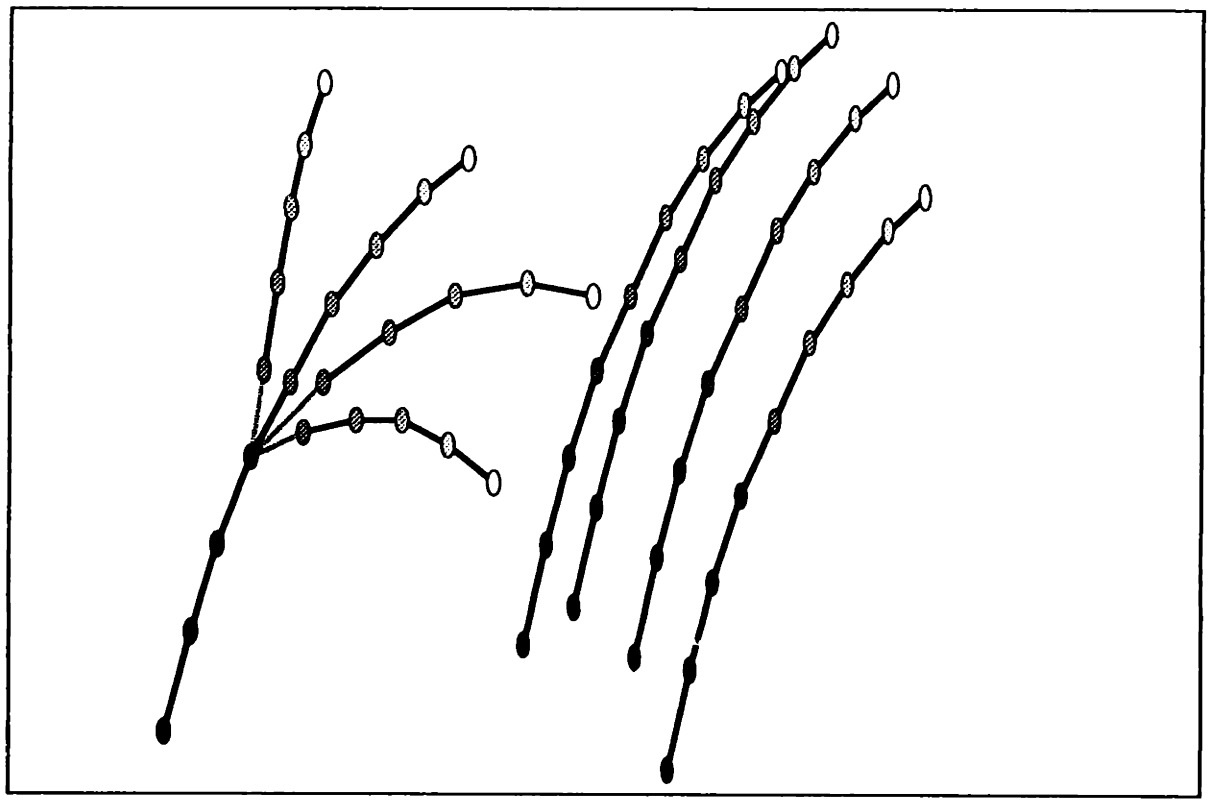


Figure 5-3: Conceptual Visualization of Simulation C

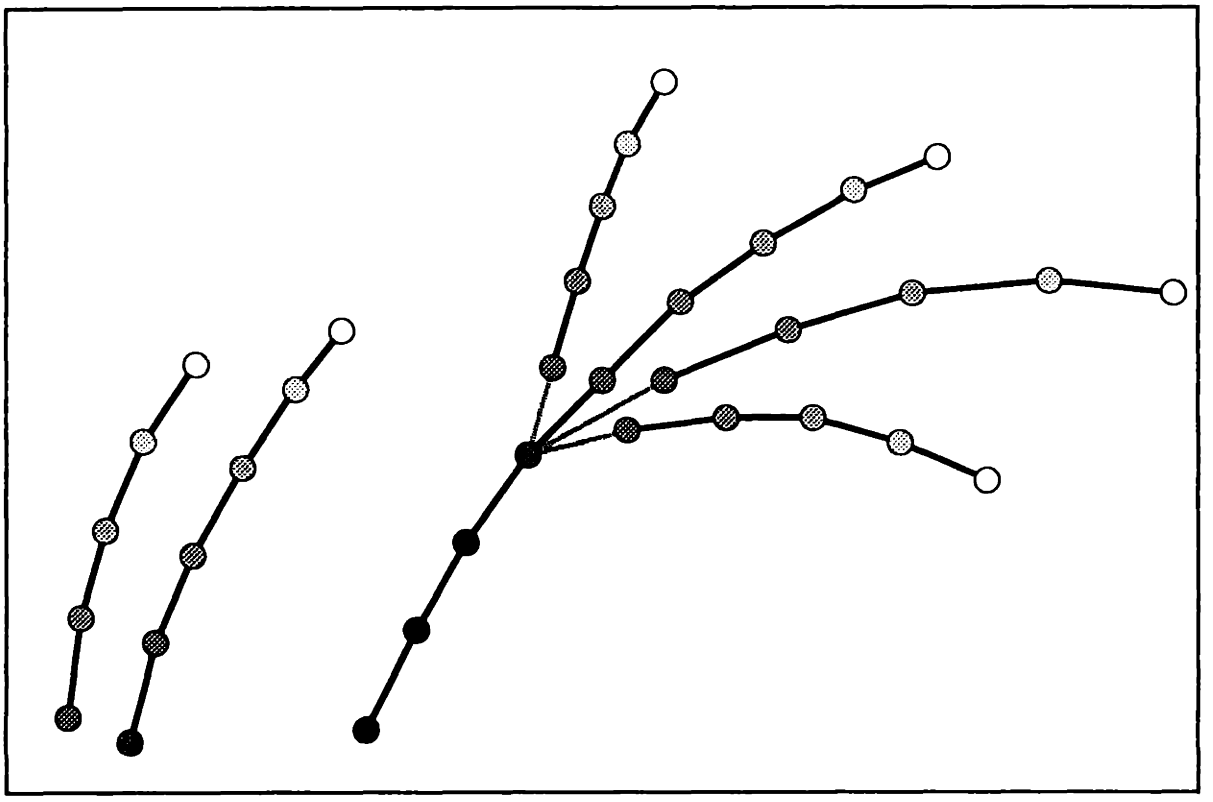


Figure 5-4: Conceptual Visualization of Simulation D

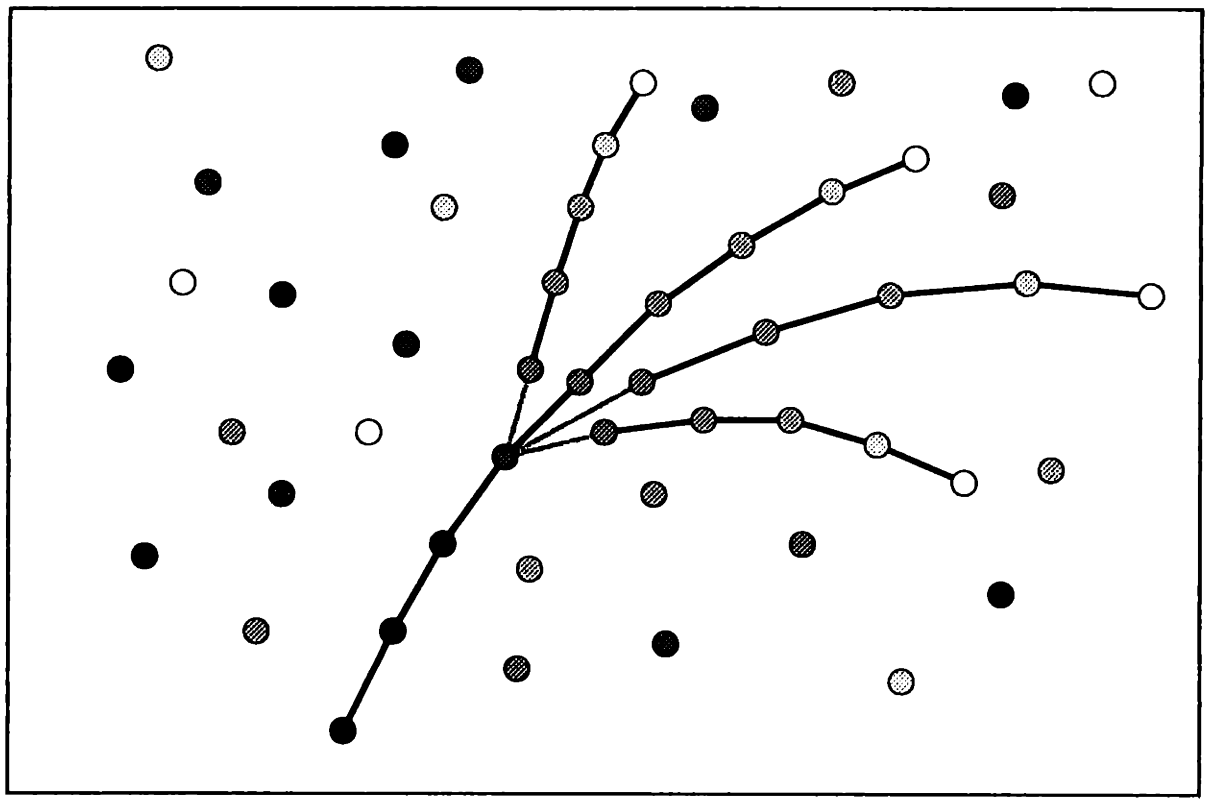


Figure 5-5: Conceptual Visualization of Simulation E

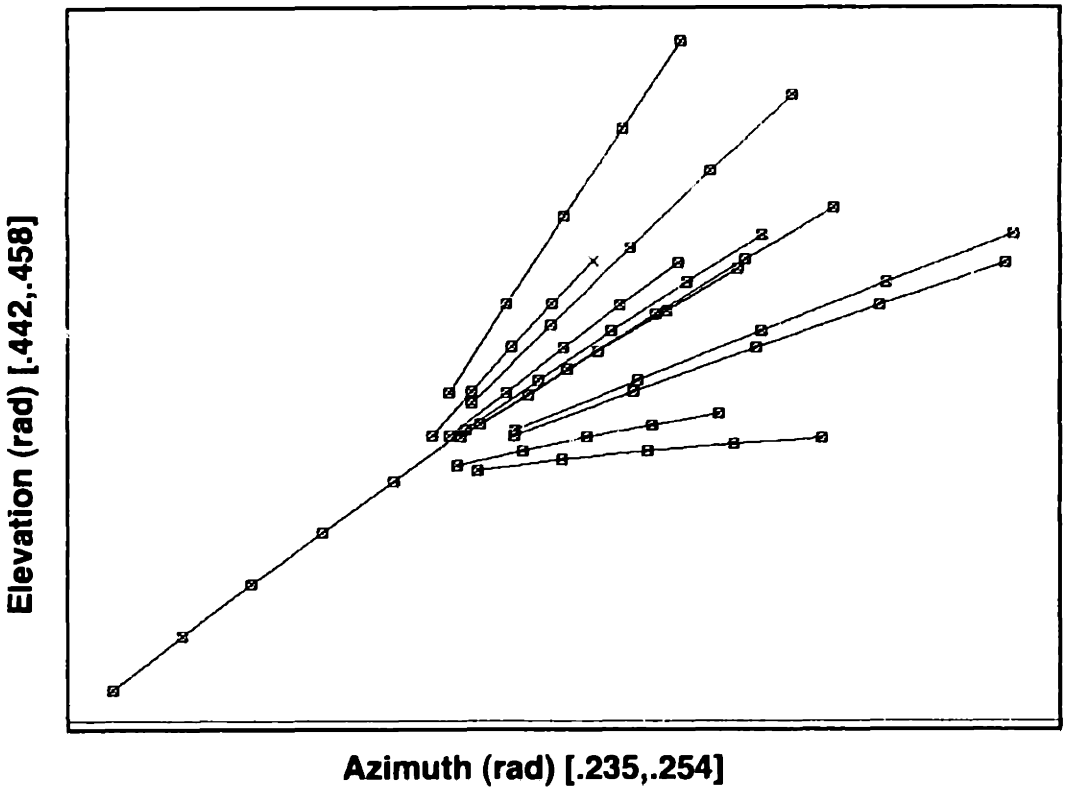


Figure 5-6: Simulation A - Elevation vs. Azimuth

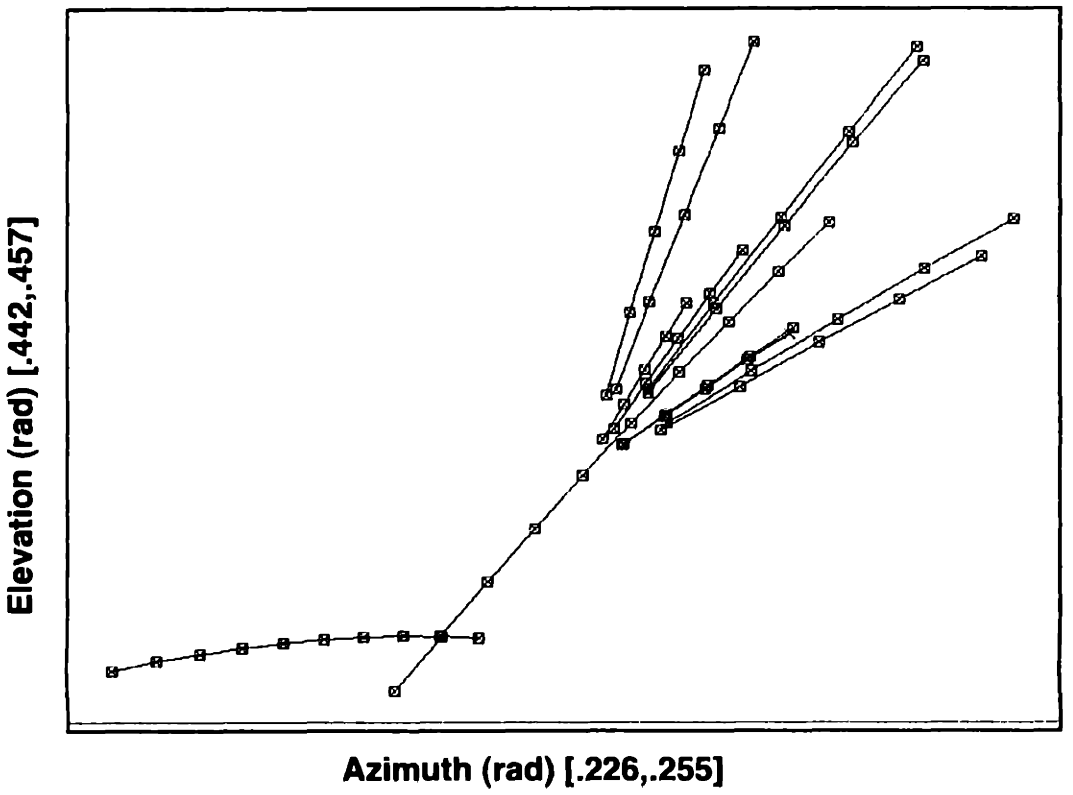


Figure 5-7: Simulation B – Elevation vs. Azimuth

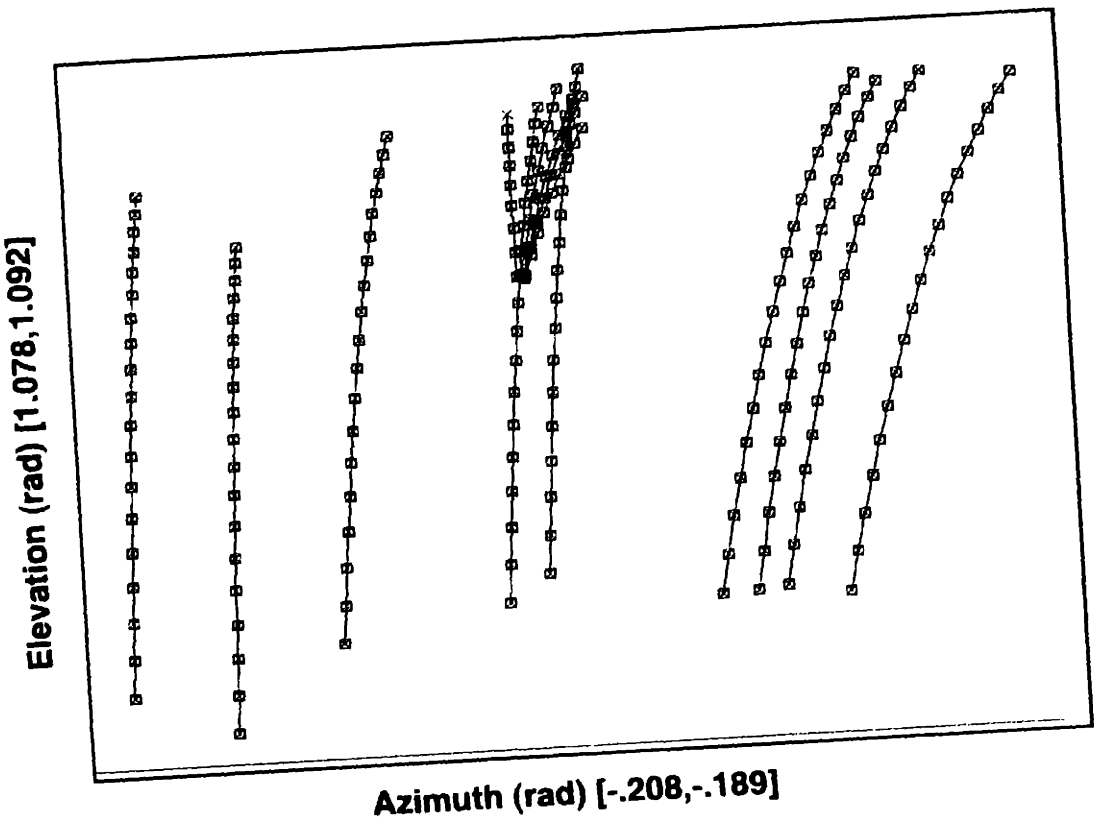


Figure 5-8: Simulation C – Elevation vs. Azimuth

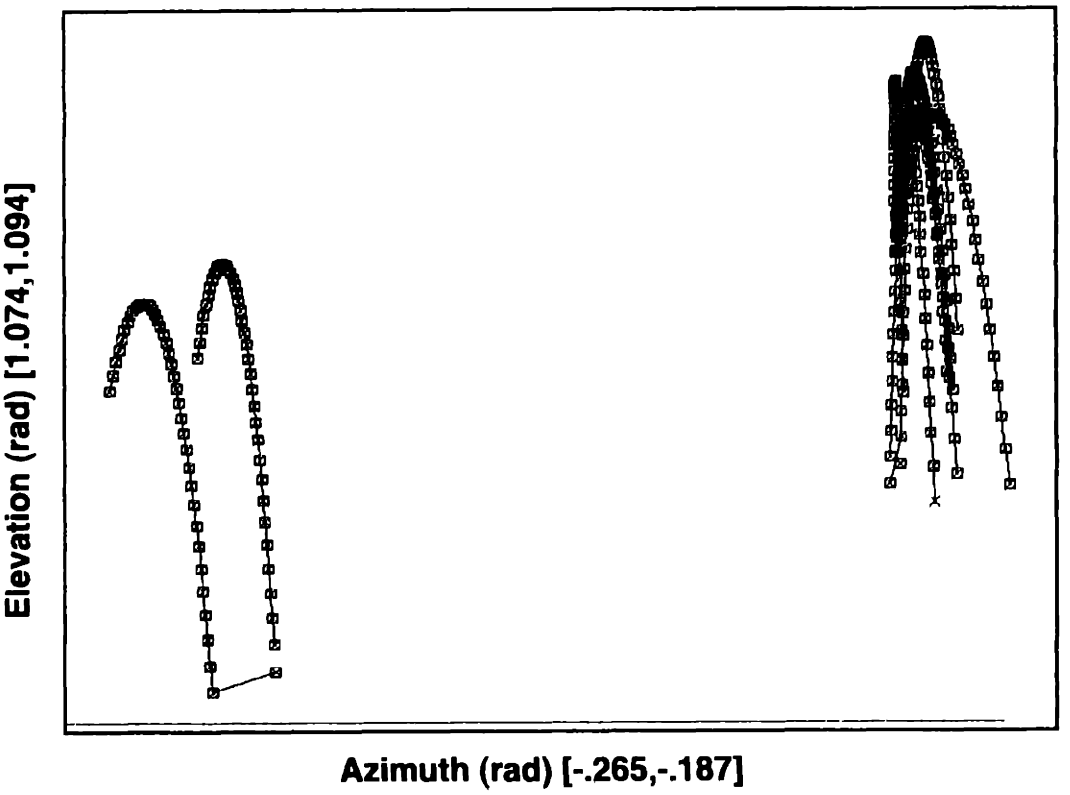


Figure 5-9: Simulation D – Elevation vs. Azimuth

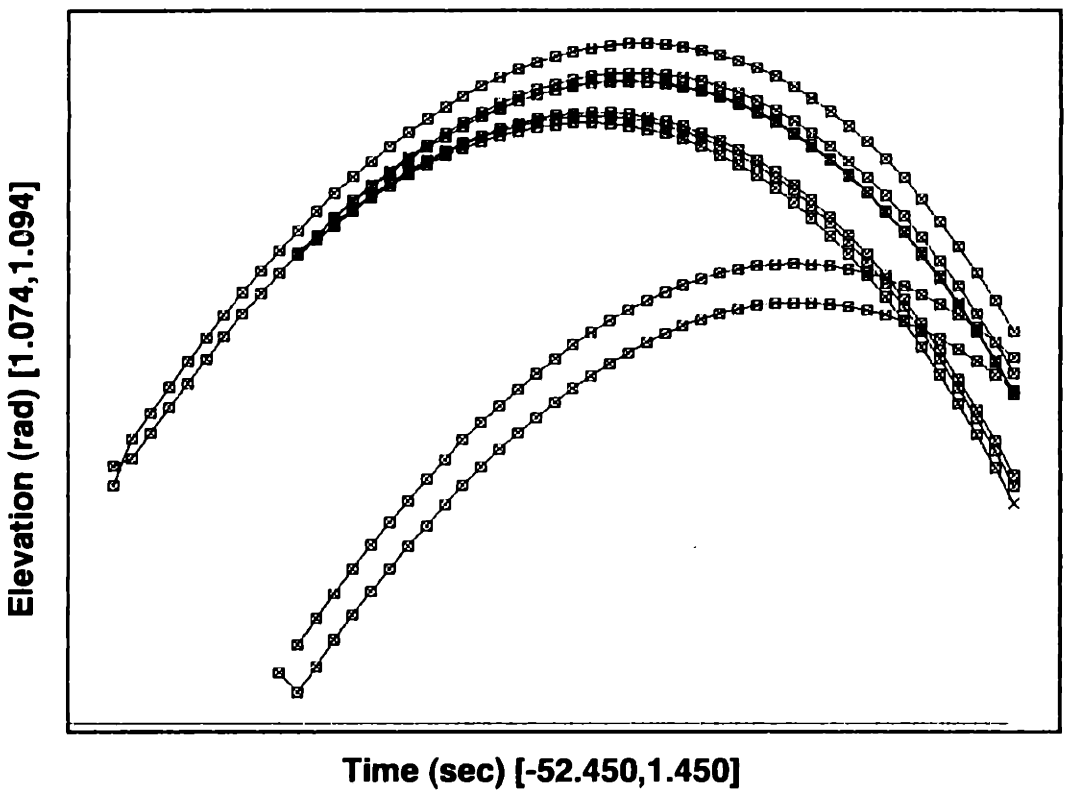


Figure 5-10: Simulation D – Elevation vs. Time

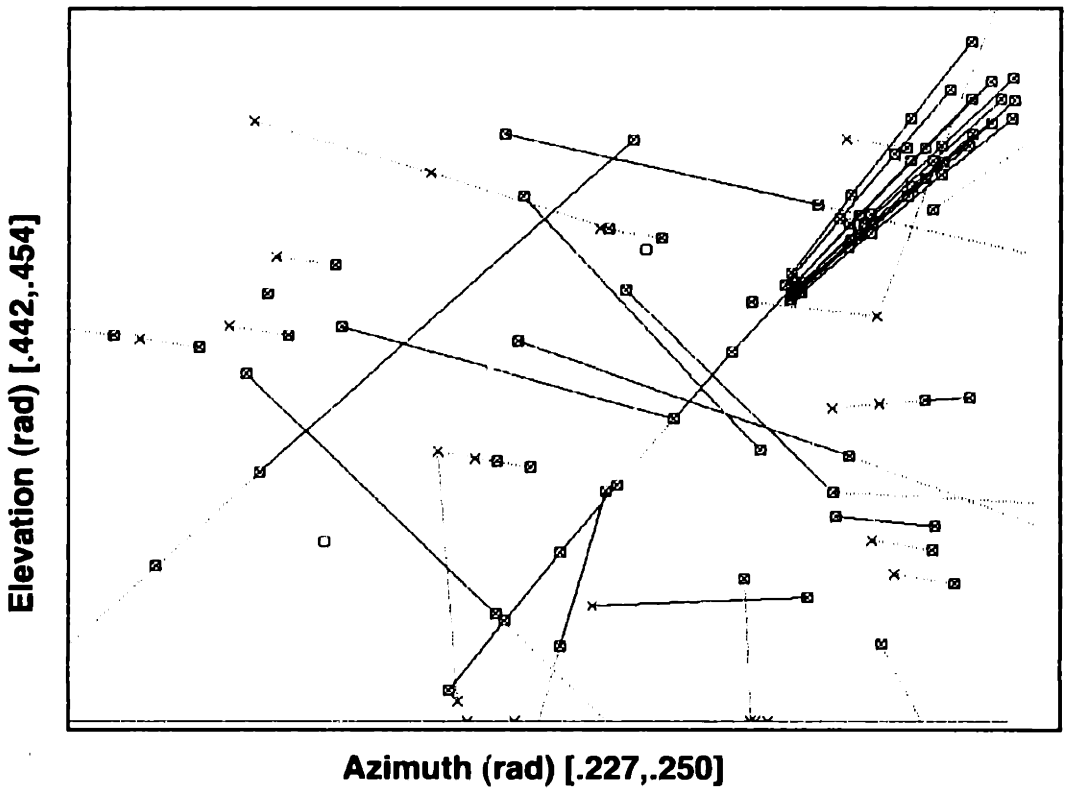


Figure 5-11: Simulation E1 – Elevation vs. Azimuth

Results of scan 9
Number of Tracks: 22
Number of Hypotheses: 10

Track Summary

Track	Observations										Status	Quality	
1	1	2	3	4	5	-1	17	-1	39	-1	C	0.00000	
2	6	-1	28	-1	50	0	0	0	0	0	B	0.00000	orig: 5
3	1	2	3	4	-1	6	17	28	39	50	C	0.00000	
4	1	2	3	4	5	6	17	28	39	50	C	0.00000	
5	5	7	18	29	40	51	0	0	0	0	C	0.00000	
6	7	18	29	40	51	0	0	0	0	0	B	0.00000	orig: 5
7	6	8	19	30	41	52	0	0	0	0	C	0.00000	
8	8	19	30	41	52	0	0	0	0	0	B	0.00000	orig: 5
9	5	9	20	31	42	53	0	0	0	0	C	0.00000	
10	9	20	31	42	53	0	0	0	0	0	B	0.00000	orig: 5
11	10	21	32	43	54	0	0	0	0	0	B	0.00000	orig: 5
12	5	11	22	33	44	55	0	0	0	0	C	0.00000	
13	11	22	33	44	55	0	0	0	0	0	B	0.00000	orig: 5
14	12	23	34	45	56	0	0	0	0	0	B	0.00000	orig: 5
15	5	13	24	35	46	57	0	0	0	0	C	0.00000	
16	13	24	35	46	57	0	0	0	0	0	B	0.00000	orig: 5
17	5	14	25	36	47	58	0	0	0	0	C	0.00000	
18	14	25	36	47	58	0	0	0	0	0	B	0.00000	orig: 5
19	5	15	26	37	48	59	0	0	0	0	C	0.00000	
20	15	26	37	48	59	0	0	0	0	0	B	0.00000	orig: 5
21	5	16	27	38	49	60	0	0	0	0	C	0.00000	
22	16	27	38	49	60	0	0	0	0	0	B	0.00000	orig: 5

Hypothesis Summary

Hypothesis	Tracks										Cost	Complexity
1	4	6	8	10	11	13	14	16	18	20	76.00000	38.
	22	0	0	0	0	0	0	0	0	0		
2	1	2	6	8	10	11	13	14	16	18	84.00000	42.
	20	22	0	0	0	0	0	0	0	0		
3	3	21	6	8	10	11	13	14	16	18	84.00000	42.
	20	0	0	0	0	0	0	0	0	0		
4	3	9	6	8	11	13	14	16	18	20	84.00000	42.
	22	0	0	0	0	0	0	0	0	0		
5	[illegible]	19	6	8	10	11	13	14	16	18	84.00000	42.
	22	0	0	0	0	0	0	0	0	0		
6	3	12	6	8	10	11	14	16	18	20	84.00000	42.
	22	0	0	0	0	0	0	0	0	0		
7	3	7	6	10	11	13	14	16	18	20	84.00000	42.
	22	0	0	0	0	0	0	0	0	0		
8	3	15	6	8	10	11	13	14	18	20	84.00000	42.
	22	0	0	0	0	0	0	0	0	0		
9	3	17	6	8	10	11	13	14	16	20	84.00000	42.
	22	0	0	0	0	0	0	0	0	0		
10	3	5	8	10	11	13	14	16	18	20	84.00000	42.
	22	0	0	0	0	0	0	0	0	0		

Figure 5-12: Final scan summary of simulation A

Results of scan 9
Number of Tracks: 46
Number of Hypotheses: 20

Track Summary

Track	Observations										Status	Quality		
1	1	3	5	7	9	11	23	35	47	59	C	0.00000		
2	1	4	5	7	9	11	23	35	47	59	C	3.42175		
3	2	3	6	8	10	12	24	36	48	60	C	3.50653		
4	2	4	6	8	10	12	24	36	48	60	C	0.00000		
5	13	25	37	49	61	0	0	0	0	0	B	0.00000	orig:	9
6	18	25	37	49	61	0	0	0	0	0	B	1.65867	orig:	9
7	13	30	37	49	61	0	0	0	0	0	B	0.37682	orig:	9
8	18	30	37	49	61	0	0	0	0	0	B	1.80062	orig:	9
9	13	25	42	49	61	0	0	0	0	0	B	0.83632	orig:	9
10	18	25	42	49	61	0	0	0	0	0	B	1.91755	orig:	9
11	13	30	42	49	61	0	0	0	0	0	B	1.06921	orig:	9
12	18	30	42	49	61	0	0	0	0	0	B	1.91556	orig:	9
13	13	25	37	54	61	0	0	0	0	0	B	1.36764	orig:	9
14	18	25	37	54	61	0	0	0	0	0	B	2.02308	orig:	9
15	13	30	37	54	61	0	0	0	0	0	B	1.56757	orig:	9
16	18	30	37	54	61	0	0	0	0	0	B	1.98814	orig:	9
17	13	25	42	54	61	0	0	0	0	0	B	1.79980	orig:	9
18	18	25	42	54	61	0	0	0	0	0	B	1.87780	orig:	9
19	13	30	42	54	61	0	0	0	0	0	B	1.85580	orig:	9
20	18	30	42	54	61	0	0	0	0	0	B	1.69892	orig:	9
21	14	26	38	50	62	0	0	0	0	0	B	0.00000	orig:	9
22	15	27	39	51	63	0	0	0	0	0	B	0.00000	orig:	9
23	16	28	40	52	64	0	0	0	0	0	B	0.00000	orig:	9
24	17	29	41	53	64	0	0	0	0	0	B	4.26414	orig:	9
25	16	28	40	52	65	0	0	0	0	0	B	4.26412	orig:	9
26	17	29	41	53	65	0	0	0	0	0	B	0.00000	orig:	9
27	13	25	37	49	66	0	0	0	0	0	B	1.69892	orig:	9
28	18	25	37	49	66	0	0	0	0	0	B	1.85581	orig:	9
29	13	30	37	49	66	0	0	0	0	0	B	1.87779	orig:	9
30	18	30	37	49	66	0	0	0	0	0	B	1.79981	orig:	9
31	13	25	42	49	66	0	0	0	0	0	B	1.98814	orig:	9
32	18	25	42	49	66	0	0	0	0	0	B	1.56758	orig:	9
33	13	30	42	49	66	0	0	0	0	0	B	2.02308	orig:	9
34	18	30	42	49	66	0	0	0	0	0	B	1.36764	orig:	9
35	13	25	37	54	66	0	0	0	0	0	B	1.91556	orig:	9
36	18	25	37	54	66	0	0	0	0	0	B	1.06921	orig:	9
37	13	30	37	54	66	0	0	0	0	0	B	1.91754	orig:	9
38	18	30	37	54	66	0	0	0	0	0	B	0.83633	orig:	9
39	13	25	42	54	66	0	0	0	0	0	B	1.80061	orig:	9
40	18	25	42	54	66	0	0	0	0	0	B	0.37682	orig:	9
41	13	30	42	54	66	0	0	0	0	0	B	1.65866	orig:	9
42	18	30	42	54	66	0	0	0	0	0	B	0.00000	orig:	9
43	19	31	43	55	67	0	0	0	0	0	B	0.00000	orig:	9
44	20	32	44	56	68	0	0	0	0	0	B	0.00000	orig:	9
45	21	33	45	57	69	0	0	0	0	0	B	0.00000	orig:	9
46	22	34	46	58	70	0	0	0	0	0	B	0.00000	orig:	9

Hypothesis Summary

Hypothesis	Tracks										Cost	Complexity
1	1	4	5	21	22	23	26	42	43	44	90.00000	45.
	45	46	0	0	0	0	0	0	0	0		
2	1	4	7	21	22	23	26	40	43	44	90.75364	45.
	45	46	0	0	0	0	0	0	0	0		
3	1	4	9	21	22	23	26	38	43	44	91.67265	45.
	45	46	0	0	0	0	0	0	0	0		
4	1	4	11	21	22	23	26	36	43	44	92.13842	45.
	45	46	0	0	0	0	0	0	0	0		
5	1	4	13	21	22	23	26	34	43	44	92.73529	45.
	45	46	0	0	0	0	0	0	0	0		
6	1	4	15	21	22	23	26	32	43	44	93.13515	45.
	45	46	0	0	0	0	0	0	0	0		
7	1	4	41	21	22	23	26	6	43	44	93.31734	45.
	45	46	0	0	0	0	0	0	0	0		
8	1	4	27	21	22	23	26	20	43	44	93.39784	45.
	45	46	0	0	0	0	0	0	0	0		
9	1	4	17	21	22	23	26	30	43	44	93.59961	45.
	45	46	0	0	0	0	0	0	0	0		
10	1	4	39	21	22	23	26	8	43	44	93.60124	45.
	45	46	0	0	0	0	0	0	0	0		

Figure 5-13: Abridged final scan summary of simulation B

```
Results of scan   18
Number of Tracks:        845
Number of Hypotheses:    1000
 Track Summary
 Track        Observations                                    Status      Quality
   1    |   1  10  19  28  37  46  55  64  73  82 |  C  |      0.00000
        |  91 105 119 133 147 161 175 189 203   0 |     |
   3    |   2  11  20  29  38  47  56  65  74  83 |  C  |      0.00000
        |  92 106 120 134 148 162 176 190 204   0 |     |
   4    |   3  12  21  30  39  48  57  66  75  84 |  C  |      0.00000
        |  93 107 121 135 149 163 177 191 205   0 |     |
   5    |   4  13  22  31  40  49  58  67  76  85 |  C  |      0.00000
        |  94 108 122 136 150 164 178 192 206   0 |     |
   6    |   5  14  23  32  41  50  59  68  77  86 |  C  |      0.00000
        |  95 109 123 137 151 165 179 193 207   0 |     |
   7    |   6  15  24  33  42  51  60  69  78  87 |  C  |      0.00000
        |  96 110 124 138 152 166 180 194 208   0 |     |
   8    |   7  16  25  34  43  52  61  70  79  88 |  C  |      0.00000
        |  97 111 125 139 153 167 181 195 209   0 |     |
   9    |   8  17  26  35  44  53  62  71  80  89 |  C  |      0.00000
        |  98 112 126 140 154 168 182 196 210   0 |     |
  10    |   9  18  27  36  45  54  63  72  81  90 |  C  |      0.00000
        |  99 113 127 141 155 169 183 197 211   0 |     |
  11    | 100 114 128 142 156 170 184 198 212   0 |  B  |      0.00000 orig:  82
 429    | 101 115 129 143 157 171 185 199 213   0 |  B  |      0.00000 orig:  82
 436    | 102 116 130 144 158 172 186 200 214   0 |  B  |      0.00000 orig:  82
 437    | 103 117 131 145 159 173 187 201 215   0 |  B  |      0.00000 orig:  82
 845    | 104 118 132 146 160 174 188 202 216   0 |  B  |      0.00000 orig:  82

 Hypothesis Summary
 Hypothesis       Tracks                                          Cost   Complexity
     1        |   1   3   4   5   6   7   8   9  10  11 |      158.00000  79.
              | 429 436 437 845   0   0   0   0   0   0 |
```

Figure 5-14: Abridged final scan summary of simulation C

```
Results of scan   49
Number of Tracks:       38
Number of Hypotheses:   38
 Track Summary
 Track       Observations                                          Status      Quality
   2    |   1   3   5   7   9  11  13  15  17  19 |   C   |       0.00000
        |  22  31  40  49  58  67  76  85  94 103 |       |
        | 112 121 130 139 148 157 166 175 184 193 |       |
        | 202 211 220 229 238 247 256 265 274 283 |       |
        | 292 301 310 319 328 337 346 355 364 373 |       |
   7    |   2   4   6   8  10  12  14  16  18  20 |   C   |       0.00000
        |  23  32  41  50  59  68  77  86  95 104 |       |
        | 113 122 131 140 149 158 167 176 185 194 |       |
        | 203 212 221 230 239 248 257 266 275 284 |       |
        | 293 302 311 320 329 338 347 356 365 374 |       |
  11    |  21  24  33  42  51  60  69  78  87  96 |   C   |       0.00000
        | 105 114 123 132 141 150 159 168 177 186 |       |
        | 195 204 213 222 231 240 249 258 267 276 |       |
        | 285 294 303 312 321 330 339 348 357 366 |       |
        | 375   0   0   0   0   0   0   0   0   0 |       |
  12    |  25  34  43  52  61  70  79  88  97 106 |   C   |       0.00000
        | 115 124 133 142 151 160 169 178 187 196 |       |
        | 205 214 223 232 241 250 259 268 277 286 |       |
        | 295 304 313 322 331 340 349 358 367 376 |       |
  13    |  26  35  44  53  62  71  80  89  98 107 |   B   |       0.00000 orig:  19
        | 116 125 134 143 152 161 170 179 188 197 |       |
        | 206 215 224 233 242 251 260 269 278 287 |       |
        | 296 305 314 323 332 341 350 359 368 377 |       |
  21    |  27  36  45  54  63  72  81  90  99 108 |   B   |       0.00000 orig:  19
        | 117 126 135 144 153 162 171 180 189 198 |       |
        | 207 216 225 234 243 252 261 270 279 288 |       |
        | 297 306 315 324 333 342 351 360 369 378 |       |
  32    |  28  37  46  55  64  73  82  91 100 109 |   B   |       0.00000 orig:  19
        | 118 127 136 145 154 163 172 181 190 199 |       |
        | 208 217 226 235 244 253 262 271 280 289 |       |
        | 298 307 316 325 334 343 352 361 370 379 |       |
  33    |  29  38  47  56  65  74  83  92 101 110 |   B   |       0.00000 orig:  19
        | 119 128 137 146 155 164 173 182 191 200 |       |
        | 209 218 227 236 245 254 263 272 281 290 |       |
        | 299 308 317 326 335 344 353 362 371 380 |       |
  38    |  30  39  48  57  66  75  84  93 102 111 |   B   |       0.00000 orig:  19
        | 120 129 138 147 156 165 174 183 192 201 |       |
        | 210 219 228 237 246 255 264 273 282 291 |       |
        | 300 309 318 327 336 345 354 363 372 381 |       |

 Hypothesis Summary
 Hypothesis       Tracks                                                  Cost   Complexity
     1       |   2   7  11  12  13  21  32  33  38   0 |       88.00000  44.
```

Figure 5-15: Abridged final scan summary of simulation D

```
Results of scan   9
Number of Tracks:        58
Number of Hypotheses:    42
 Track Summary
 Track          Observations                                    Status      Quality
   1    |   6   -1   -1    0    0    0    0    0    0    0  |   F   |      0.00000
   2    |   7   -1   -1    0    0    0    0    0    0    0  |   F   |      0.00000
   3    |   8   -1   -1    0    0    0    0    0    0    0  |   F   |      0.00000
   4    |   5   11   -1   -1   -1    0    0    0    0    0  |   L   |      0.00000
   5    |   4   12   -1   -1   -1    0    0    0    0    0  |   L   |      0.00000
   6    |  14   -1   -1    0    0    0    0    0    0    0  |   F   |      0.00000
   7    |  10   16   -1   -1   -1    0    0    0    0    0  |   L   |      0.00000
   9    |  13   17   -1   -1   -1    0    0    0    0    0  |   L   |      0.00000
  10    |  19   -1   -1   -1    0    0    0    0    0    0  |   D   |      0.00000
  14    |  20   22   -1   -1   -1    0    0    0    0    0  |   L   |      0.00000
  15    |  23   -1   -1    0    0    0    0    0    0    0  |   F   |      0.00000
  16    |  24   -1   -1    0    0    0    0    0    0    0  |   F   |      0.00000
  18    |  18   25   -1   -1   -1    0    0    0    0    0  |   L   |      0.00000
  19    |  26   -1   -1    0    0    0    0    0    0    0  |   F   |      0.00000
  21    |  38   -1   -1    0    0    0    0    0    0    0  |   F   |      0.00000
  22    |  40   -1   -1   -1    0    0    0    0    0    0  |   D   |      0.00000
  23    |  41   -1   -1.  -1    0    0    0    0    0    0  |   D   |      0.00000
  24    |  42   -1   -1   -1    0    0    0    0    0    0  |   D   |      0.00000
  25    |  54   -1   -1    0    0    0    0    0    0    0  |   F   |      0.00000
  26    |  56   -1   -1    0    0    0    0    0    0    0  |   F   |      0.00000
  27    |  39   57   -1   -1    0    0    0    0    0    0  |   T   |      0.00000
  28    |  58   -1   -1    0    0    0    0    0    0    0  |   B   |      0.00000 orig:  27
  29    |  70   -1    0    0    0    0    0    0    0    0  |   P   |      0.00000
  30    |  55   74   -1    0    0    0    0    0    0    0  |   T   |      0.00000
  31    |   1    2    3    9   15   21   27   43   59   75  |   C   |      0.00000
  32    |  28   44   60   76    0    0    0    0    0    0  |   B   |      0.00000 orig:  21
  33    |  29   45   61   77    0    0    0    0    0    0  |   B   |      0.00000 orig:  21
  34    |  30   46   62   78    0    0    0    0    0    0  |   B   |      0.00000 orig:  21
  41    |  31   47   63   79    0    0    0    0    0    0  |   B   |      0.00000 orig:  21
  42    |  32   48   64   80    0    0    0    0    0    0  |   B   |      0.00000 orig:  21
  43    |  33   49   65   81    0    0    0    0    0    0  |   B   |      0.00000 orig:  21
  44    |  34   50   66   82    0    0    0    0    0    0  |   B   |      0.00000 orig:  21
  45    |  35   51   67   83    0    0    0    0    0    0  |   B   |      0.00000 orig:  21
  46    |  36   52   68   84    0    0    0    0    0    0  |   B   |      0.00000 orig:  21
  53    |  37   53   69   85    0    0    0    0    0    0  |   B   |      0.00000 orig:  21
  54    |  86    0    0    0    0    0    0    0    0    0  |   U   |      0.00000
  55    |  72   87    0    0    0    0    0    0    0    0  |   T   |      0.00000
  56    |  88    0    0    0    0    0    0    0    0    0  |   U   |      0.00000
  57    |  73   89    0    0    0    0    0    0    0    0  |   T   |      0.00000
  58    |  71   90    0    0    0    0    0    0    0    0  |   T   |      0.00000

Hypothesis Summary
Hypothesis       Tracks                                              Cost    Complexity
    1        |  31   5   4   1   2   3   7   9   6  18  |      518.00000 259.
             |  10  14  15  16  19  32  33  34  41  42  |
             |  43  44  45  46  53  21  27  22  23  24  |
             |  25  30  26  28  29  58  55  57  54  56  |
```

Figure 5-16: Abridged final scan summary of simulation E1

Chapter 6

Conclusions and Future Work

In this thesis, the problem of target breakup detection has been defined, given a solution, and successfully incorporated into the multiple hypothesis tracking formulation. Simulations have been performed to test out possible scenarios which the tracker should be able to handle, including correctness of identifying breakup and non-breakup tracks and implementation issues such as pruning. However, these simulations, by no means have demonstrated all aspects of the tracker. Some other things worth looking at are:

- *Ability to handle noise in the scan volume.* This includes the density of false alarms present in the scan volume and noisy measurements. A Monte Carlo test would be most appropriate for determining the "noise level" that this tracker can handle.

- *Different combinations of difficult tracks.* The simulations included crossing tracks and independent vs. breakup track scenarios. Some more possibilities include breakup tracks that also break up into more breakup tracks, breakup tracks from two separate origins that cross, and tracks with missing measurements.

- *Real Data.* The data that the tracker was tested on was generated by the assumed model of the dynamics of a target. This assumed model does not match exactly the real model of a target, and thus, this tracker needs to be

tested on real data to prove its worth. After all, that is the final goal of this thesis, to track real targets and detect breakups of real targets.

There are many other scenarios to simulate to observe the performance of the tracker, but the three mentioned above come immediately to mind.

One of the major assumptions imposed on targets was that targets cannot exhibit any maneuvers. In this way, it was possible to distinguish an independent track from a breakup track because only breakup tracks exhibited a ΔV in the middle of tracking. So, a natural extension for the target breakup detection algorithm presented in this thesis is to handle target breakup detection when targets have the ability to maneuver. However, thinking about this more closely, it seems that target breakup detection automatically detects target maneuvers. Maneuvers can be thought of as an independent track that breaks up into exactly one breakup piece and where the original independent track ends at the time of breakup. This is exactly the hypothesis, H_1, presented in the beginning of Section 3.1. Rather than identifying a track with a single breakup as a target breakup, if the tracker can discriminate this special case of breakup as a target maneuver, then target maneuverability has been handled. Whether the target breakup model is a good model for target maneuvers needs to be explored further in order to justify the above presumption about maneuver detection.

One problem with the implementation of the tracker, whose code is printed in Appendix B, is that the code runs much too slowly to be used in a real tracking system which requires real-time tracking. The simulated data returns a scan every second. Fifty seconds of data where nine tracks are present in the scan volume takes over five hours to analyze. Thus, optimization for speed is crucial for the usefulness of this algorithm. All is not lost because the code of Appendix B is not the most optimal implementation of this tracker and can be improved; furthermore, better hardware, such as supercomputers, would definitely help execution time. A good pruning algorithm will definitely lessen execution time; thus, coming up with a better pruning algorithm would be a good start to improving the implementation of the algorithm.

Appendix A

Derivation of Akaike's Criterion

The derivation of Akaike's criterion that follows has been taken from [1]. The basic idea is to minimize the Kullback-Leibler mean information since it is a measure of the deviation between two probability distributions. The Kullback-Leibler mean information takes only positive values, and a value of zero implies that the two probability distributions are equal. Thus, considering N independent observations of a random variable, $x_1, x_2, \ldots, x_N$, which could have been generated by one of the several projected probability densities, $f(x \mid \theta)$ parameterized by the vector θ but which actually has a "true" probability density of $g(x)$, the best choice of the parameter θ is

$$\hat{\theta} = \arg_\theta \min\, I(g; f(\cdot \mid \theta)) \tag{A.1}$$

where

$$I(a; b) = \int a(x) \log \frac{a(x)}{b(x)}\, dx$$

is the Kullback-Leibler mean information between two densities, $a(x)$ and $b(x)$. Thus, $f(x \mid \hat{\theta})$ is the density that most closely matches the true density $g(x)$. However, $g(x)$ is unknown so that the minimization in Eqn. A.1 must be performed without the knowledge of $g(x)$. This can be done, as will be demonstrated shortly.

Define the function

$$S(c; d) = \int c(x) \log d(x)\, dx$$

for some densities, $c(x)$ and $d(x)$. Then,

$$I(g; f(\cdot \mid \theta)) = S(g; g) - S(g; f(\cdot \mid \theta)) \ .$$

Thus, Eqn. A.1 can be reduced to

$$\hat{\theta} = \arg_{\theta}\max \ S(g; f(\cdot \mid \theta)) \tag{A.2}$$

since $S(g; g)$ does not change with different θ.

Returning back to the N-sequence, $x_1, x_2, \ldots, x_N$, the average log-likelihood, given that the density of each x_i is $f(x_i \mid \theta)$, is

$$\frac{1}{N} \sum_{i=1}^{N} \log f(x_i \mid \theta) \ .$$

If N approaches infinity, then

$$S(g; f(\cdot \mid \theta)) = \lim_{N \to \infty} \frac{1}{N} \sum_{i=1}^{N} \log f(x_i \mid \theta) \ . \tag{A.3}$$

Thus, the maximum likelihood estimate of θ, gotten by plugging Eqn. A.3 for finite N into Eqn. A.2, is

$$MLE \ \hat{\theta} = \arg_{\theta}\max \frac{1}{N} \sum_{i=1}^{N} \log f(x_i \mid \theta) \tag{A.4}$$

for some N. Note that the most natural estimate of $S(g; f(\cdot \mid \theta))$, the average log-likelihood, does not depend on $g(x)$. Thus, $\hat{\theta}$ can be found without the knowledge of $g(x)$.

Consider the situation where $g(x) = f(x \mid \theta_0)$ and adopt the new notation, $I(\theta_0; \theta)$ and $S(\theta_0; \theta)$, for $I(g; f(\cdot \mid \theta))$ and $S(g; f(\cdot \mid \theta))$ repectively. When θ is sufficiently close to θ_0, $I(\theta_0; \theta)$ can be approximated by

$$I(\theta_0; \theta) \approx (\frac{1}{2}) \left\| \ \theta - \theta_0 \ \right\|_J^2$$

where

$$\left\| \theta - \theta_0 \right\|_J^2 = (\theta - \theta_0)^T J(\theta - \theta_0)$$

and J is the Fisher information matrix defined by

$$J_{ij} = E\{\frac{\partial \log f(X \mid \theta)}{\partial \theta_i} \frac{\partial \log f(X \mid \theta)}{\partial \theta_j}\}$$

where J_{ij} denotes the $(i,j)^{\underline{th}}$ element of J and θ_i the $i^{\underline{th}}$ component of θ. Thus, the quantity, $(\frac{1}{2})\left\| \theta - \theta_0 \right\|_J^2$, which is the variation of $S(g; f(\cdot \mid \theta))$, measures the deviation of the distribution $f(x \mid \theta)$ from the "true" distribution $f(x \mid \theta_0)$.

Now, consider the situation where the parameters $\theta \in \Theta$ are restricted to a subspace of Θ, say $\boldsymbol{\Theta}$, of lower dimension than Θ. Furthermore $\boldsymbol{\Theta}$ does not include θ_0, and the dimension of $\boldsymbol{\Theta}$ is k. Let ξ be the estimate from Eqn. A.2,

$$\xi = \arg_\theta \max\, S(g; f(\cdot \mid \theta)) \ ,$$

restricted to the subspace $\boldsymbol{\Theta}$, and let $\hat{\theta}$ be the maximum likelihood estimate from Eqn. A.4,

$$\hat{\theta} = \arg_\theta \max \frac{1}{N} \sum_{i=1}^{N} \log f(x_i \mid \theta) \ ,$$

also restricted to the subspace $\boldsymbol{\Theta}$. Then, it can be shown that the distribution of $N\left\| \hat{\theta} - \xi \right\|_J^2$ for sufficiently large N is approximated under certain regularity conditions by a chi-square distribution with degree of freedom, k. Thus,

$$\begin{aligned}
E_\infty[2NI(\theta_0; \hat{\theta})] &= E_\infty[N\left\| \hat{\theta} - \theta_0 \right\|_J^2] \\
&= E_\infty[N\left\| \hat{\theta} - \xi + \xi - \theta_0 \right\|_J^2] \\
&= E_\infty[N\left\| \hat{\theta} - \xi \right\|_J^2 + N\left\| \xi - \theta_0 \right\|_J^2] \\
&= E_\infty[N\left\| \hat{\theta} - \xi \right\|_J^2] + E_\infty[N\left\| \xi - \theta_0 \right\|_J^2]
\end{aligned}$$

where E_∞ denotes the mean of the approximate distribution. The first term on the right hand side has a chi-square distribution, and the second term is constant with

regards to E_∞. Thus,

$$E_\infty[2NI(\theta_0;\hat{\theta})] = N\left\| \xi - \theta_0 \right\|_J^2 + k \ . \tag{A.5}$$

k, as mentioned before, is the dimension of Θ and can be interpreted as the number of independently adjusted parameters for the maximization of the likelihood.

When there are several models, the one to adopt as the "correct" model is the one that minimizes $E[I(\theta_0;\theta)]$. Thus, an estimate for $N\left\| \xi - \theta_0 \right\|_J^2$ is necessary. Since it was assumed that $N\left\| \hat{\theta} - \xi \right\|_J^2$ was a chi-square distribution, $\sqrt{N}(\hat{\theta}-\xi)$ is a Guassian distribution with mean zero and variance J^{-1}.

$$\begin{aligned} N\left\| \xi - \theta_0 \right\|_J^2 &= 2NI(\theta_0;\xi) \\ &= 2N(S(\theta_0;\theta_0) - S(\theta_0,\xi)) \\ &= 2(\sum_{i=1}^{N} \log f(x_i \mid \theta_0) - \sum_{i=1}^{N} \log f(x_i \mid \xi)) \\ &= 2(\sum_{i=1}^{N} \log f(x_i \mid \theta_0) - \sum_{i=1}^{N} \log f(x_i \mid \hat{\theta})) + k \end{aligned}$$

where the last line obtained an added factor of k since replacing ξ by $\hat{\theta}$ introduced a downward bias. Since it is the minimum of $E[I(\theta_0;\hat{\theta})]$ that is of interest, the term, $\sum_{i=1}^{N} \log f(x_i \mid \theta_0)$, can be discarded. Then, plugging in the estimate of $N\left\| \xi - \theta_0 \right\|_J^2$ into Eqn. A.5, Akaike's criterion is obtained:

$$AIC(\hat{\theta}) = (-2)\log(\textit{maximum likelihood}) + 2k \tag{A.6}$$

where the maximum likelihood is $\sum_{i=1}^{N} \log f(x_i \mid \hat{\theta})$. Thus, the $\hat{\theta}$ that minimizes $AIC(\hat{\theta})$ is the best estimate of the model.

Appendix B

FORTRAN Code Implementing MHT with Target Breakup Detection

The following code must be compiled in FORTRAN 77 with a file called "metric.p" and the IMSL math library. The file and software package are available at MIT Lincoln Laboratory Group 32.

```
*-----------------------------------------------------------------------+
*                        Top Level Main Routine                         |
*                                for                                    |
*                     Mulitple Hypothesis Tracking                      |
*-----------------------------------------------------------------------+

**** Global Variables ****

      IMPLICIT REAL*8 (A-H,O-Z)

*-----------------------------------------------------------------------+
* Parameter description:                                                |
*     N               dimension of state vector                         |
*     M               dimension of measurement vector                   |
*     D2R             conversion constant from degrees to radians       |
*     R2D             conversion constant from radians to degrees       |
*     pi              constant for pi                                   |
*     maxLength       maximum number of observations for a track        |
*     maxObs          maximum number of observations stored             |
*     maxSize         maximum number of tracks in a hypothesis          |
*     maxTrack        maximum number of tracks in the tracklist         |
*     IbigPrune       maximum number of hypothesis ever stored          |
*     IlittlePrune    maximum number of hypothesis stored after scan    |
*     nScan           tracks combined after nScan observations same     |
*-----------------------------------------------------------------------+

      PARAMETER (N=6,M=3,D2R=.017453293D0,R2D=1D0/D2R,
     .   pi=3.141592653589793D0, maxLength=100,maxObs=1000,
     .   maxSize=50,maxTrack=10000,IbigPrune=1000,IlittlePrune=10,
     .   nScan=5)

*-----------------------------------------------------------------------+
* Common block description:                                             |
```

```
*     Idebug          different levels of printing debug information    |
*     Iunit           Observations data (1-radians 0-degrees)           |
*     IDATA           data file  (input)                                |
*     IDBUGO          debug file (output)                               |
*     IPICT           pict file (output)                                |
*     IMODEL          model file (output)                               |
*     trackList       array of tracks                                   |
*     gatesize        size of gate                                      |
*     distance        distance between predicted and actual position    |
*     Q,R             intensities for model error and measurement error |
*     V,Vin           innovations covariance and inverse                |
*     GBR,Xs          Ground based radar location in LLA coordinates    |
*                     and ECI coordinates at initial time               |
*     sensorTime      time of sensor                                    |
*-----------------------------------------------------------------------+

      COMMON /CDEBUG/Idebug
      COMMON /CUNITS/Iunit
      COMMON /CFILES/IDATA, IDBUGO, IPICT, IMODEL
      COMMON /CTRACKLIST/trackList,numTracks
      COMMON /COBSERVATIONS/Observations,numObs
      COMMON /CGATE/gatesize, distance
      COMMON /CMODEL/Q(N,N), R(M,M), P(N,N), V(M,M), Vinv(M,M)
      COMMON /CSENSOR/GBR(M), Xs(N), sensorTime

      DIMENSION Observations(maxObs,M+1)

      STRUCTURE /TRACK/
        CHARACTER status
        INTEGER length,using,points(maxLength),new,misses
        REAL*8 score,predState(N),updState(N),predP(N,N),updP(N,N)
        END STRUCTURE
      RECORD /TRACK/trackList(maxTrack)
      INTEGER numTracks

      STRUCTURE /HYPOTHESIS/
        INTEGER numTracks,hypoTracks(maxSize)
        REAL*8 score,complexity      ! score = likelihood + complexity
        END STRUCTURE
      RECORD /HYPOTHESIS/oldHypos(IbigPrune)

      INTEGER firstObsInScan,numObsInScan,sloop,numOldHypos
      INTEGER I
      REAL*8 aveCurTime,avePrevTime

**** Begin Main Routine ****

      CALL FileDef
      CALL EARTH84

**** Program Initialization ****
      CALL InitialData(numOldHypos,oldHypos,numScan)
      CALL MakePictFile(1,numObs,oldHypos)

      DO 800 sloop = 1,numScan
         IF (Idebug .GE. 1) WRITE (IDBUGO,*) 'Beginning Scan #',sloop
         WRITE (*,*) 'Processing Scan #',sloop
**** Read Observations ****
         firstObsInScan = numObs + 1
         READ (IDATA,*) numObsInScan
         numObs = numObs + numObsInScan
         DO 100 I = firstObsInScan,numObs
 100        CALL ReadObservation(I)

**** Move the sensor to appropriate ECI coordinate ****
         avePrevTime = sensorTime
         aveCurTime = 0.D0
         DO 200 I = firstObsInScan,numObs
 200        aveCurTime = aveCurTime + Observations(I,M+1)
         aveCurTime = aveCurTime/numObsInScan
         CALL MoveSensor(aveCurTime)

**** Predict next target states ****
         DO 300 I = 1,numTracks
            IF (trackList(I).status .NE. 'U') THEN
               CALL TimeUpdate(I,avePrevTime,aveCurTime)
            END IF
 300     CONTINUE

**** Data Correlation and Make hypotheses ****
         CALL DoScan(firstObsInScan,numObs,oldHypos,
     .                numOldHypos,avePrevTime,aveCurTime)
         CALL MakePictFile(firstObsInScan,numObs,oldHypos)

**** Print results from this scan ****
```

```
         WRITE (IDBUGO,*) ' '
         WRITE (IDBUGO,*) 'Results of scan ',sloop
         CALL PrintSummary(oldHypos,numOldHypos)

 800  CONTINUE

      CLOSE(IDATA)
      CLOSE(IDBUGO)
      CLOSE(IPICT)
      CLOSE(IMODEL)

      STOP
      END !**** Main Routine ****

*-----------------------------------------------------------------------+
*     SUBROUTINE FileDef - define the files used                        |
*-----------------------------------------------------------------------+

      SUBROUTINE FileDef

      COMMON /CDEBUG/Idebug
      COMMON /CFILES/IDATA, IDBUGO, IPICT, IMODEL

      IDATA  = 10 !**** Contains all of the observation data   ****
      IDBUGO = 11 !**** Contains debug info                    ****
      IPICT  = 12 !**** Contains info for picture generator    ****
      IMODEL = 13 !**** Contains sensor and Kalman filter info ****

      OPEN (UNIT=IDATA,FILE='MHT.DATA',STATUS='OLD')
      OPEN (UNIT=IDBUGO,FILE='IDBUGO.OUTPUT',STATUS='NEW')
      OPEN (UNIT=IPICT,FILE='PICT.DATA',STATUS='NEW')
      OPEN (UNIT=IMODEL,FILE='MODEL.DATA',STATUS='OLD')

      RETURN
      END   !**** FileDef ****

*-----------------------------------------------------------------------+
*     SUBROUTINE InitialData - get data for initial tracks, sensor, and |
*                              Kalman filter noise matrices             |
*-----------------------------------------------------------------------+

      SUBROUTINE InitialData(numOldHypos,oldHypos,numScan)

      IMPLICIT REAL*8 (A-H,O-Z)

      PARAMETER (N=6,M=3,D2R=.017453293D0,R2D=1D0/D2R,
     .   pi=3.141592653589793 2D0, maxLength=100,maxObs=1000,
     .   maxSize=50,maxTrack=10000,IbigPrune=1000,IlittlePrune=10,
     .   nScan=5)

      PARAMETER (DnewTarget = 5.D-2, DfalseTarget = 5.D-2)

      COMMON /CDEBUG/Idebug
      COMMON /CUNITS/Iunit
      COMMON /CFILES/IDATA, IDBUGO, IPICT, IMODEL
      COMMON /CTRACKLIST/trackList,numTracks
      COMMON /COBSERVATIONS/Observations,numObs
      COMMON /CGATE/gatesize, distance
      COMMON /CMODEL/Q(N,N), R(M,M), P(N,N), V(M,M), Vinv(M,M)
      COMMON /CSENSOR/GBR(M), Xs(M), sensorTime

      DIMENSION Observations(maxObs,M+1)

      STRUCTURE /TRACK/
        CHARACTER status
        INTEGER length,using,points(maxLength),new,misses
        REAL*8 score,predState(N),updState(N),predP(N,N),updP(N,N)
        END STRUCTURE
      RECORD /TRACK/trackList(maxTrack)
      INTEGER numTracks

      STRUCTURE /HYPOTHESIS/
        INTEGER numTracks,hypoTracks(maxSize)
        REAL*8 score,complexity
        END STRUCTURE
      RECORD /HYPOTHESIS/oldHypos(IbigPrune)

      INTEGER I,J, numOldHypos

**** Read in some debug parameters ****
      READ (IDATA,*) Idebug
      READ (IDATA,*) Iunit
      READ (IDATA,*) sensorTime

**** Read in Kalman filter noise variances ****
```

```
      READ (IMODEL,*) ((Q(I,J),J=1,N),I=1,N)
      READ (IMODEL,*) ((R(I,J),J=1,M),I=1,M)
      READ (IMODEL,*) ((P(I,J),J=1,N),I=1,N)

**** Read in sensor information ****
      CALL GetInitialSensorState

**** Read in gatesize ****
      READ (IMODEL,*) gatesize
      IF (Idebug .GE. 2) THEN
         WRITE (IDBUGO,*) 'Gatesize: ',gatesize
      END IF

**** Initialize Hypothesis List and Track List *****
      DO 100 I=1,IbigPrune
         oldHypos(I).numTracks = 0
         oldHypos(I).complexity = 0D0
 100     oldHypos(I).score = 0D0
      DO 200 I=1,maxTrack
         trackList(I).status = 'U'
         trackList(I).length = 0
         trackList(I).using = 0
         trackList(I).new = 0
         trackList(I).misses = 0
 200     trackList(I).score = 0D0

**** Read in first set of Observations and make each of them tracks ****
      numTracks = 0
      READ (IDATA,*) numObs
      DO 300 I=1,numObs
         CALL ReadObservation(I)
         CALL MakeTrack(0,I)
         trackList(I).using = 1 !**** this track is used by the initial ****
                                !**** hypothesis                          ****
         trackList(I).status = 'U' !**** since we assume a priori knowledge **
         oldHypos(1).score = oldHypos(1).score + trackList(I).score
         oldHypos(1).hypoTracks(I)=I
 300     CONTINUE
      oldHypos(1).complexity = 7D0 * numObs
      oldHypos(1).score = oldHypos(1).score + oldHypos(1).complexity
      oldHypos(1).numTracks = numTracks
      numOldHypos = 1

**** Read in number of scans of data ****
      READ (IDATA,*) numScan

      RETURN
      END !**** InitialData ****

*-----------------------------------------------------------------------+
*     SUBROUTINE GetInitialSensorState - get the initial sensor state   |
*-----------------------------------------------------------------------+

      SUBROUTINE GetInitialSensorState

      IMPLICIT REAL*8 (A-H,O-Z)

      PARAMETER (N=6,M=3,D2R=.017453293D0,R2D=1D0/D2R,
     .   pi=3.141592653589793 2D0, maxLength=100,maxObs=1000,
     .   maxSize=50,maxTrack=10000,IbigPrune=1000,IlittlePrune=10,
     .   nScan=5)

      COMMON /CDEBUG/Idebug
      COMMON /CUNITS/Iunit
      COMMON /CFILES/IDATA, IDBUGO, IPICT, IMODEL
      COMMON /CSENSOR/GBR(M), Xs(N), sensorTime

      DIMENSION zeroVector(N)
      REAL*8 OM
      INTEGER IsenseUnit

**** Initialize some variables ****
      DO 100 I=1,N
 100     zeroVector(I) = 0D0
      OM = .72921151467D-4  !**** Earth rotation rate ****

**** Read in the sensor information ****
      READ (IMODEL,*) IsenseUnit        ! 1 - radians  0 - degrees
      READ (IMODEL,*) (GBR(I),I=1,M)

**** Make appropriate conversions ****
      IF (IsenseUnit .EQ. 0) THEN
         GBR(1) = GBR(1)*D2R
         GBR(2) = GBR(2)*D2R + OM*sensorTime
      ELSE
```

```
         GBR(2) = GBR(2) + OM*sensorTime
      ENDIF

**** Calculate sensor state (LLA to ECI) ****
      CALL XYZECI(zeroVector,Xs,GBR(1),GBR(3),GBR(2),1)

**** Debug output ****
      IF (Idebug .GE. 2) THEN
         WRITE (IDBUGO,*) '-----------------------------------'
         WRITE (IDBUGO,*) 'Sensor   : GDLat (rad)    :',GBR(1)
         WRITE (IDBUGO,*) 'Location : ECEF Long (rad):',GBR(2)
         WRITE (IDBUGO,*) '         : Alt (km)       :',GBR(3)/1D3
         WRITE (IDBUGO,*) 'Initial Time (s)          :',sensorTime
         WRITE (IDBUGO,*) '-----------------------------------'
         WRITE (IDBUGO,*) 'Sensor ECI state:',(Xs(I),I=1,6)
         WRITE (IDBUGO,*) '-----------------------------------'
      ENDIF

      RETURN
      END !**** GetInitialSensorState ****

*-----------------------------------------------------------------------+
*     SUBROUTINE ReadObservation - read in an observation and put into  |
*                                  Observations(num)                    |
*-----------------------------------------------------------------------+

      SUBROUTINE ReadObservation(num)

      IMPLICIT REAL*8 (A-H,O-Z)

      PARAMETER (N=6,M=3,D2R=.017453293D0,R2D=1D0/D2R,
     .   pi=3.1415926535897932D0, maxLength=100,maxObs=1000,
     .   maxSize=50,maxTrack=10000,IbigPrune=1000,IlittlePrune=10,
     .   nScan=5)

      COMMON /CDEBUG/Idebug
      COMMON /CUNITS/Iunit
      COMMON /CFILES/IDATA, IDBUGO, IPICT, IMODEL
      COMMON /COBSERVATIONS/Observations,numObs

      DIMENSION Observations(maxObs,M+1)

      INTEGER I,num

**** Read from file ****
      READ (IDATA,*) (Observations(num,I),I=1,M+1)

**** Convert to radians if in degrees ****
      IF (Iunit .EQ. 0) THEN  !**** read in as degrees
         Observations(num,2) = Observations(num,2)*D2R
         Observations(num,3) = Observations(num,3)*D2R
      END IF
      IF (Observations(num,2) .GT. pi) then
         Observations(num,2) = Observations(num,2) - 2D0*pi
      END IF

**** Debug Info Output ****
      IF (Idebug .GE. 2) THEN
         WRITE (IDBUGO,900) 'Read obs(',num,') = ',
     .          (Observations(num,I),I=1,M+1)
      END IF

  900 FORMAT (A,I3,A,' ',F16.7,' ',F16.7,' ',F16.7,' ',F16.7)

      RETURN
      END !**** ReadObservation ****

*-----------------------------------------------------------------------+
*     SUBROUTINE FindObsState - find the ECI coordinates of the observ. |
*-----------------------------------------------------------------------+

      SUBROUTINE FindObsState(num,X)

      IMPLICIT REAL*8 (A-H,O-Z)

      PARAMETER (N=6,M=3,D2R=.017453293D0,R2D=1D0/D2R,
     .   pi=3.141592653589793D0, maxLength=100,maxObs=1000,
     .   maxSize=50,maxTrack=10000,IbigPrune=1000,IlittlePrune=10,
     .   nScan=5)

      COMMON /CDEBUG/Idebug
      COMMON /CFILES/IDATA, IDBUGO, IPICT, IMODEL
      COMMON /COBSERVATIONS/Observations,numObs
      COMMON /CSENSOR/GBR(M), Xs(N), sensorTime
```

```
      DIMENSION Observations(maxObs,M+1)

      DIMENSION rae(M),enu(N),X(N),sensor(M),Xstate(N)
      REAL*8 delta,OM

**** Find sensor position at observation time ****
      delta = Observations(num,M+1) - sensorTime
      OM = .72921151467D-4
      sensor(1) = GBR(1)
      sensor(2) = GBR(2) + OM*delta
      sensor(3) = GBR(3)
      CALL XGBR(delta,1,sensor,3,Xstate)
      CALL ECILLA(Xstate,sensor(1),sensor(2),sensor(3))

**** Convert observation to ECI ****
      DO 100 I = 1,M
 100     rae(I) = Observations(num,I)
      IJOB  = 1
      IEFIX = 1
      CALL RAEXYZ(rae,enu,M,IJOB,IER)
      DO 200 I = M+1,N
 200     enu(I) = 0D0

*****************************************
** Fudging velocities of initial tracks **
*****************************************
*      IF (num .EQ. 1) THEN
*         enu(4) = 2.81D0
*         enu(5) = -1713.92D0
*         enu(6) = -348.07D0
*      ELSE IF (num .EQ. 2) THEN
*         enu(4) = -996.61D0
*         enu(5) = -2713.43D0
*         enu(6) = -1349.11
*      END IF

*      IF (num .EQ. 1) THEN
*         enu(4) = 80.75357D0
*         enu(5) = -381.20269D0
*         enu(6) = -490.59328D0
*      ELSE IF (num .EQ. 2) THEN
*         enu(4) = 80.41604
*         enu(5) = -380.73522
*         enu(6) = -487.54462
*      ELSE IF (num .EQ. 3) THEN
*         enu(4) = 82.27586
*         enu(5) = -383.43454
*         enu(6) = -490.42602
*      ELSE IF (num .EQ. 4) THEN
*        enu(4) = 80.75446
*         enu(5) = -381.72344
*         enu(6) = -490.93692
*      ELSE IF (num .EQ. 5) THEN
*         enu(4) = 83.50017
*         enu(5) = -382.79434
*         enu(6) = -489.94941
*      ELSE IF (num .EQ. 6) THEN
*         enu(4) = 87.13216
*         enu(5) = -384.17541
*         enu(6) = -492.00993
*      ELSE IF (num .EQ. 7) THEN
*         enu(4) = 86.28474
*        enu(5) = -382.80877
*         enu(6) = -491.35981
*      ELSE IF (num .EQ. 8) THEN
*         enu(4) = 85.30807
*         enu(5) = -381.84414
*        enu(6) = -490.59327
*      ELSE IF (num .EQ. 9) THEN
*         enu(4) = 88.01744
*         enu(5) = -383.72916
*         enu(6) = -492.61121
*      ENDIF

      CALL XYZECI(enu,X,sensor(1),sensor(3),sensor(2),IEFIX)

      RETURN
      END !**** FindObsState ****

*-----------------------------------------------------------------------+
*     SUBROUTINE MoveSensor - move the sensor to current time           |
*-----------------------------------------------------------------------+

      SUBROUTINE MoveSensor(curTime)
```

```
      IMPLICIT REAL*8 (A-H,O-Z)

      PARAMETER (N=6,M=3,D2R=.017453293D0,R2D=1D0/D2R,
     .   pi=3.1415926535897932D0, maxLength=100,maxObs=1000,
     .   maxSize=50,maxTrack=10000,IbigPrune=1000,IlittlePrune=10,
     .   nScan=5)

      COMMON /CDEBUG/Idebug
      COMMON /CFILES/IDATA, IDBUGO, IPICT, IMODEL
      COMMON /CSENSOR/GBR(M), Xs(N), sensorTime

      REAL*8 delta,OM

      delta = curTime - sensorTime
      OM = .72921151467D-4
      GBR(2) = GBR(2) + OM*delta
      CALL XGBR(delta,1,GBR,M,Xs,N)
      CALL ECILLA(Xs,GBR(1),GBR(2),GBR(3))
      sensorTime = curTime

**** Debug output ****
      IF (Idebug .GE. 2) THEN
         WRITE (IDBUGO,*) '---New Sensor Position--------------'
         WRITE (IDBUGO,*) 'Sensor   : GDLat (rad)     :',GBR(1)
         WRITE (IDBUGO,*) 'Location : ECEF Long (rad):',GBR(2)
         WRITE (IDBUGO,*) '         : Alt (km)        :',GBR(3)/1D3
         WRITE (IDBUGO,*) 'Initial Time (s)           :',sensorTime
         WRITE (IDBUGO,*) '-----------------------------------'
         WRITE (IDBUGO,*) 'Sensor ECI state:',(Xs(I),I=1,6)
         WRITE (IDBUGO,*) '-----------------------------------'
      ENDIF

      RETURN
      END  !**** MoveSensor ****

*-----------------------------------------------------------------------+
*     SUBROUTINE FindTrackRAE - find RAE equivalent of state            |
*-----------------------------------------------------------------------+

      SUBROUTINE FindTrackRAE(num,predRae,updRae)

      IMPLICIT REAL*8 (A-H,O-Z)

      PARAMETER (N=6,M=3,D2R=.017453293D0,R2D=1D0/D2R,
     .   pi=3.1415926535897932D0, maxLength=100,maxObs=1000,
     .   maxSize=50,maxTrack=10000,IbigPrune=1000,IlittlePrune=10,
     .   nScan=5)

      COMMON /CDEBUG/Idebug
      COMMON /CFILES/IDATA, IDBUGO, IPICT, IMODEL
      COMMON /CTRACKLIST/trackList,numTracks
      COMMON /CSENSOR/GBR(M), Xs(N), sensorTime

      STRUCTURE /TRACK/
        CHARACTER status
        INTEGER length,using,points(maxLength),new,misses
        REAL*8 score,predState(N),updState(N),predP(N,N),updP(N,N)
        END STRUCTURE
      RECORD /TRACK/trackList(maxTrack)
      INTEGER numTracks

      INTEGER num
      DIMENSION predRae(M),updRae(M),enu(M)
      REAL*8 geola,ecilo,height

      CALL ECILLA(Xs,geola,ecilo,height)
      CALL ECIXYZ(trackList(num).predState,enu,geola,ecilo,height,
     .            M,1,1,IER)
      CALL XYZRAE(enu,predRae,M,1,IER)

      CALL ECIXYZ(trackList(num).updState,enu,geola,ecilo,height,
     .            M,1,1,IER)
      CALL XYZRAE(enu,updRae,M,1,IER)

      RETURN
      END  !**** FindTrackRAE ****

*-----------------------------------------------------------------------+
*     SUBROUTINE MakePictFile - make the input file for the picture     |
*                               driver                                  |
*-----------------------------------------------------------------------+

      SUBROUTINE MakePictFile(first,last,oldHypos)

      IMPLICIT REAL*8 (A-H,O-Z)
```

```
      PARAMETER (N=6,M=3,D2R=.017453293D0,R2D=1D0/D2R,
     .   pi=3.1415926535897932D0, maxLength=100,maxObs=1000,
     .   maxSize=50,maxTrack=10000,IbigPrune=1000,IlittlePrune=10,
     .   nScan=5)

      PARAMETER (DnewTarget = 5.D-2, DfalseTarget = 5.D-2)

      COMMON /CDEBUG/Idebug
      COMMON /CUNITS/Iunit
      COMMON /CFILES/IDATA, IDBUGO, IPICT, IMODEL
      COMMON /CTRACKLIST/trackList,numTracks
      COMMON /COBSERVATIONS/Observations,numObs
      COMMON /CMODEL/Q(N,N), R(M,M), P(N,N), V(M,M), Vinv(M,M)
      COMMON /CSENSOR/GBR(M), Xs(N), sensorTime

      DIMENSION Observations(maxObs,M+1)

      STRUCTURE /TRACK/
        CHARACTER status
        INTEGER length,using,points(maxLength),new,misses
        REAL*8 score,predState(N),updState(N),predP(N,N),updP(N,N)
        END STRUCTURE
      RECORD /TRACK/trackList(maxTrack),curTrack
      INTEGER numTracks

      STRUCTURE /HYPOTHESIS/
        INTEGER numTracks,hypoTracks(maxSize)
        REAL*8 score,complexity
        END STRUCTURE
      RECORD /HYPOTHESIS/oldHypos(IbigPrune),htemp

      INTEGER first,last,num,I,J
      DIMENSION predRae(M),updRae(M)

      WRITE (IPICT,*) last-first+1
      DO 100 I=first,last
  100    WRITE (IPICT,900) I,(Observations(I,J),J=1,M+1)

      htemp = oldHypos(1)
      WRITE (IPICT,*) htemp.numTracks
      DO 110 I = 1,htemp.numTracks
         curTrack = trackList(htemp.hypoTracks(I))
         WRITE (IPICT,*) curTrack.length
         IF (curTrack.length .EQ. 1) THEN
            WRITE (IPICT,910) 0,curTrack.points(1)
            CALL FindTrackRAE(htemp.hypoTracks(I),predRae,updRae)
            WRITE (IPICT,905) (predRae(J),J=1,M)
            WRITE (IPICT,905) (updRae(J),J=1,M)
         ELSE
            num = 0
            DO 105 J=1,curTrack.length - 1
               IF ((curTrack.points(curTrack.length-J) .NE. -1)
     .              .AND. (num .EQ. 0)) THEN
                  num = curTrack.length - J
               END IF
  105       CONTINUE
            IF ((curTrack.status .EQ. 'L') .OR.
     .          (curTrack.status .EQ. 'D')) THEN
               WRITE (IPICT,910) 0,curTrack.points(num)
            ELSE
               WRITE (IPICT,910) curTrack.points(num),
     .                     curTrack.points(curTrack.length)
            END IF
            CALL FindTrackRAE(htemp.hypoTracks(I),predRae,updRae)
            WRITE (IPICT,905) (predRae(J),J=1,M)
            WRITE (IPICT,905) (updRae(J),J=1,M)
         END IF
  110 CONTINUE

  900 FORMAT (I4,' ',F16.7,' ',F16.7,' ',F16.7,' ',F16.7)
  905 FORMAT (F16.7,' ',F16.7,' ',F16.7)
  910 FORMAT (2I4)

      RETURN
      END  !**** MakePictFile ****

*-----------------------------------------------------------------------+
*     SUBROUTINE TimeUpdate - make predictions for next track's state   |
*                             from updState to predState                |
*-----------------------------------------------------------------------+

      SUBROUTINE TimeUpdate(num,T1,T2)

      IMPLICIT REAL*8 (A-H,O-Z)
```

```
      PARAMETER (N=6,M=3,D2R=.017453293D0,R2D=1D0/D2R,
     .   pi=3.1415926535897932D0, maxLength=100,maxObs=1000,
     .   maxSize=50,maxTrack=10000,IbigPrune=1000,IlittlePrune=10,
     .   nScan=5)

      COMMON /CDEBUG/Idebug
      COMMON /CFILES/IDATA, IDBUGO, IPICT, IMODEL
      COMMON /CTRACKLIST/trackList,numTracks
      COMMON /CMODEL/Q(N,N), R(M,M), P(N,N), V(M,M), Vinv(M,M)

      STRUCTURE /TRACK/
        CHARACTER status
        INTEGER length,using,points(maxLength),new,misses
        REAL*8 score,predState(N),updState(N),predP(N,N),updP(N,N)
        END STRUCTURE
      RECORD /TRACK/trackList(maxTrack),curTrack
      INTEGER numTracks

      INTEGER num, I, J
      REAL*8 delta,T2,T1
      DIMENSION A(N,N), phi(N,N), temp(N,N), wxt(N), tempP(N,N)

      curTrack = trackList(num)
      delta = T2-T1
      CALL JACECI(curTrack.updState,A,N)

**** Calculate Transition Matrix (phi) ****
      DO I = 1,N
         DO J = 1,N
            A(I,J) = A(I,J) * delta
         END DO
      END DO
      CALL DMRRRR(N,N,A,N,N,N,A,N,N,N,phi,N)
      DO I = 1,N
         DO J = 1,N
            phi(I,J) = A(I,J) + 0.5D0*phi(I,J)
            IF (I .EQ. J) phi(I,J) = 1.D0 + phi(I,J)
         END DO
      END DO

**** Predict next covariance matrix (P) ****
      DO I = 1,N
         DO J = 1,N
            tempP(I,J) = curTrack.updP(I,J)
         END DO
      END DO
      CALL DMRRRR(N,N,phi,N,N,N,tempP,N,N,N,temp,N)
      CALL DMXYTF(N,N,temp,N,N,N,phi,N,N,N,tempP,N)
      DO I = 1,N
         DO J = 1,N
            curTrack.predP(I,J) = tempP(I,J) + Q(I,J)
         END DO
      END DO

**** Integrate to predict next state vector Xt ****
      DO I = 1,N
         wxt(I) = curTrack.updState(I)
      END DO
      CALL MOVESV(T1,wxt,T2,curTrack.predState)

      trackList(num) = curTrack

      RETURN
      END  !**** TimeUpdate ****

*-----------------------------------------------------------------------+
*     SUBROUTINE DoScan - iterates over                                 |
*             all of the observations making all the different          |
*             hypotheses possible.  (i.e. this routine is the MHT part) |
*             The logic of this routine goes something like this:       |
*                 1) for each observation, make a potential track and   |
*                    merge this track with the existing hypothesis      |
*                 2) for each track, check to see if the observation    |
*                    gates with the track.  If it does make new hypos.  |
*                 3) delete tracks that no hypothesis is using          |
*-----------------------------------------------------------------------+

      SUBROUTINE DoScan(first,last,oldHypos,numOldHypos,T1,T2)

      IMPLICIT REAL*8 (A-H,O-Z)

      PARAMETER (N=6,M=3,D2R=.017453293D0,R2D=1D0/D2R,
     .   pi=3.1415926535897932D0, maxLength=100,maxObs=1000,
     .   maxSize=50,maxTrack=10000,IbigPrune=1000,IlittlePrune=10,
```

```
     .   nScan=5)

      PARAMETER (DnewTarget = 5.D-2, DfalseTarget = 5.D-2)

      COMMON /CDEBUG/Idebug
      COMMON /CFILES/IDATA, IDBUGO, IPICT, IMODEL
      COMMON /CTRACKLIST/trackList,numTracks
      COMMON /COBSERVATIONS/Observations,numObs
      COMMON /CGATE/gatesize, distance

      DIMENSION Observations(maxObs,M+1)

      STRUCTURE /TRACK/
        CHARACTER status
        INTEGER length,using,points(maxLength),new,misses
        REAL*8 score,predState(N),updState(N),predP(N,N),updP(N,N)
        END STRUCTURE
      RECORD /TRACK/trackList(maxTrack),tempTrack,tempTrack1
      INTEGER numTracks

      STRUCTURE /HYPOTHESIS/
        INTEGER numTracks,hypoTracks(maxSize)
        REAL*8 score,compiexity
        END STRUCTURE
      RECORD /HYPOTHESIS/oldHypos(IbigPrune),newHypos(IbigPrune)

      INTEGER first,last,loop,I,numOldHypos,numNewHypos,J
      REAL*8 T1,T2
      INTEGER numOldTracks,originNum
      DIMENSION curObs(M+1)
      LOGICAL Gate,NewTrackPossibility,MakeBreakupTrack
      INTEGER BreakupOrigins(maxObs)

      numOldTracks = numTracks  !**** number of tracks existing prior ****
                                !**** to this scan                    ****
      DO 300 loop = first,last
         DO 25 I = 1,maxObs
 25         BreakupOrigins(I) = 0

         WRITE (*,*) 'Processing observation ',loop
         numNewHypos = 0
         DO 50 I = 1,numTracks
 50         trackList(I).using = 0

         DO 100 I = 1,M+1
 100        curObs(I) = Observations(loop,I)
**** Make a potential track with one observation ****
         CALL MakeTrack(0,loop)
         CALL IncorporateTrack(0,numTracks,oldHypos,numOldHypos,
     .                         newHypos,numNewHypos)

**** And tracks that gate with this observation make new hypotheses ****
         DO 200 I = 1,numOldTracks
            IF ((trackList(I).status .NE. 'F') .AND.
     .          (trackList(I).status .NE. 'L') .AND.
     .          (trackList(I).status .NE. 'D') .AND.
     .          (trackList(I).status .NE. 'U')) THEN
**** Here's the continuing track part ****
               IF (Gate(loop,I)) THEN
                  IF (trackList(I).status .EQ. 'B') THEN
                     WRITE (*,*) 'Breakup track ending with ',
     .                    trackList(I).points(trackList(I).length),
     .                    ' gates with obs ',loop
                     WRITE (*,*) 'distance: ',distance
                  END IF
                  CALL MakeTrack(I,loop)
                  CALL IncorporateTrack(I,numTracks,oldHypos,
     .                       numOldHypos,newHypos,numNewHypos)
               END IF
            END IF
 200     CONTINUE

**** For unassigned and newly formed breakup tracks predict velocity ****
         DO 275 I = 1,numOldTracks
            IF (trackList(I).status .EQ. 'U') THEN
**** Save the unassigned track because PredictAPriori demolishes it ****
               tempTrack = trackList(I)
               CALL PredictAPriori(I,loop,T1,T2)
               WRITE (*,*) 'Gatesize: ',gatesize
               WRITE (*,*) trackList(I).predState(4)
               WRITE (*,*) trackList(I).predState(5)
               WRITE (*,*) trackList(I).predState(6)
               IF (NewTrackPossibility(trackList(I))) THEN
                  IF (Gate(loop,I)) THEN
                     CALL MakeTrack(I,loop)
```

```
                  CALL IncorporateTrack(I,numTracks,oldHypos,
     .                 numOldHypos,newHypos,numNewHypos)
               END IF
            END IF
**** Restore the unassigned track ****
            tempTrack1 = trackList(I)
            trackList(I) = tempTrack
         END IF
**** Make a breakup track with confirmed tracks ****
         IF ((trackList(I).status .EQ. 'C') .OR.
     .       ((trackList(I).status .EQ. 'B') .AND.
     .        (trackList(I).length .GT. 2))) THEN
            originNum = trackList(I).points(trackList(I).length)
            IF ((MakeBreakupTrack(tempTrack1,trackList(I),loop,
     .              T1,T2)) .AND.
     .           (BreakupOrigins(originNum) .EQ. 0)) THEN
               BreakupOrigins(originNum) = 1
               numTracks = numTracks + 1
               trackList(numTracks) = tempTrack1
               CALL TimeUpdate(numTracks,T1,T2)
               CALL UpdateState(trackList(numTracks).predState,
     .              trackList(numTracks).predP,loop,
     .              trackList(numTracks).updState,
     .              trackList(numTracks).updP)
               IF (loop .EQ. 13) THEN
                  WRITE (*,*) 'final pred state for breakup 13: ',
     .                 (trackList(numTracks).predState(J),J=1,N)
                  WRITE (*,*) 'final upd state for breakup 13: ',
     .                 (trackList(numTracks).updState(J),J=1,N)
               END IF
c               DO 270 J = 1,N
c 270             trackList(numTracks).updState(J) =
c     .                trackList(numTracks).predState(J)
               CALL IncorporateTrack(0,numTracks,oldHypos,
     .              numOldHypos,newHypos,numNewHypos)
               WRITE (*,*) 'Breakup track (orig:',
     .              trackList(I).points(trackList(I).length),') ',
     .                          loop,' made.'
            END IF
         END IF
 275     CONTINUE


         numOldHypos = numNewHypos
         DO 250 I = 1,numNewHypos
 250        oldHypos(I) = newHypos(I)

         IF (Idebug .GE. 1) THEN
            WRITE (IDBUGO,*) ' '
            WRITE (IDBUGO,*) 'Summary after incorporating observation:',
     .                       loop
            CALL PrintSummary(oldHypos,numOldHypos)
         END IF
 300  CONTINUE

      CALL CheckMisses(numOldTracks,oldHypos,numOldHypos)
      CALL PruneHypotheses(oldHypos,numOldHypos,IlittlePrune)
      CALL PackTracks(oldHypos,numOldHypos)

      RETURN
      END !**** DoScan ****

*-----------------------------------------------------------------------+
*     SUBROUTINE PredictAPriori - predict the a priori velocity vector  |
*          of the state                                                 |
*-----------------------------------------------------------------------+

      SUBROUTINE PredictAPriori(prev,updObs,T1,T2)

      IMPLICIT REAL*8 (A-H,O-Z)

      PARAMETER (N=6,M=3,D2R=.017453293D0,R2D=1D0/D2R,
     .   pi=3.1415926535897932D0, maxLength=100,maxObs=1000,
     .   maxSize=50,maxTrack=10000,IbigPrune=1000,IlittlePrune=10,
     .   nScan=5)

      COMMON /CDEBUG/Idebug
      COMMON /CUNITS/lunit
      COMMON /CFILES/IDATA, IDBUGO, IPICT, IMODEL
      COMMON /CTRACKLIST/trackList,numTracks
      COMMON /COBSERVATIONS/Observations,numObs

      DIMENSION Observations(maxObs,M+1)

      STRUCTURE /TRACK/
```

```
        CHARACTER status
        INTEGER length,using,points(maxLength),new,misses
        REAL*8 score,predState(N),updState(N),predP(N,N),updP(N,N)
        END STRUCTURE
      RECORD /TRACK/trackList(maxTrack)
      INTEGER numTracks

      REAL*8 T1,T2
      INTEGER prev,updObs,I,J
      DIMENSION state(N),first(N),diff(M),pred(N)

      trackList(prev).status = 'P'
      CALL FindObsState(updObs,state)

      DO 100 J=1,N
 100     first(J) = trackList(prev).predState(J)
      DO 200 J=1,M
 200     diff(J) = (state(J) - first(J))/(T2-T1)

 250  CONTINUE
      DO 300 J=1,M
 300     first(J+M) = diff(J) + first(J+M)
      DO 350 J=1,M
 350     first(J) = trackList(prev).predState(J)
      CALL MOVESV(T1,first,T2,pred)
      DO 400 J=1,M
 400     diff(J) = (state(J)-pred(J))/(T2-T1)
      IF ((diff(1) .GE. 1D-5) .OR.
     .    (diff(2) .GE. 1D-5) .OR.
     .    (diff(3) .GE. 1D-5)) THEN
         GOTO 250
      END IF
      DO 450 J=1,N
         trackList(prev).predState(J) = first(J)
 450     trackList(prev).updState(J) = first(J)

c      DO 500 J=1,M
c         trackList(prev).predState(J+M) = (state(J) -
c     .        trackList(prev).predState(J))/(T2-T1)
c 500     trackList(prev).updState(J+M) = trackList(prev).predState(J+M)

      IF (Idebug .GE. 2) THEN
         WRITE (IDBUGO,*) '--PredictAPriori-- Obs ',updObs,' state:'
         WRITE (IDBUGO,*) (state(I),I=1,N)
         WRITE (IDBUGO,*) 'State before predicting:'
         WRITE (IDBUGO,*) (trackList(prev).predState(I),I=1,N)
      END IF

      CALL TimeUpdate(prev,T1,T2)
      IF (Idebug .GE. 2) THEN
         WRITE (IDBUGO,*) 'State after predicting:'
         WRITE (IDBUGO,*) (trackList(prev).predState(I),I=1,N)
      END IF

      RETURN
      END  !**** PredictAPriori ****


*-----------------------------------------------------------------------+
*     SUBROUTINE MakeTrack - make a new track by updating the track PREV|
*          with the observation(updObs) and put the track in the last   |
*          position of the trackList. increase numTracks by one         |
*        Note: as of yet, if the trackList overflows then there is a    |
*              problem ....                                             |
*-----------------------------------------------------------------------+

      SUBROUTINE MakeTrack(prev,updObs)

      IMPLICIT REAL*8 (A-H,O-Z)

      PARAMETER (N=6,M=3,D2R=.017453293D0,R2D=1D0/D2R,
     .   pi=3.1415926535897932D0, maxLength=100,maxObs=1000,
     .   maxSize=50,maxTrack=10000,IbigPrune=1000,IlittlePrune=10,
     .   nScan=5)

      PARAMETER (DnewTarget=5.D-2, DfalseTarget=5.D-2, Pdetect=.95D0,
     .   dim=3,tlength=6)

      COMMON /CDEBUG/Idebug
      COMMON /CFILES/IDATA, IDBUGO, IPICT, IMODEL
      COMMON /CTRACKLIST/trackList,numTracks
      COMMON /COBSERVATIONS/Observations,numObs
      COMMON /CGATE/gatesize, distance
      COMMON /CMODEL/Q(N,N), R(M,M), P(N,N), V(M,M), Vinv(M,M)
```

```
      DIMENSION Observations(maxObs,M+1)

      STRUCTURE /TRACK/
        CHARACTER status
        INTEGER length,using,points(maxLength),new,misses
        REAL*8 score,predState(N),updState(N),predP(N,N),updP(N,N)
        END STRUCTURE
      RECORD /TRACK/trackList(maxTrack),newTrack,oldTrack
      INTEGER numTracks

      INTEGER prev,updObs,J,K
      REAL*8 dummy

      IF (prev .EQ. 0) THEN  !**** a new track ****
         numTracks = numTracks + 1
         trackList(numTracks).status = 'U'
         trackList(numTracks).length = 1
         trackList(numTracks).using = 0
         trackList(numTracks).points(1) = updObs
         DO 200 J = 2,maxLength
 200        trackList(numTracks).points(J) = 0
         trackList(numTracks).new = 0
         trackList(numTracks).misses = 0
         trackList(numTracks).score = 0D0
         CALL FindObsState(updObs,trackList(numTracks).predState)
         DO 225 J = 1,N
 225        trackList(numTracks).updState(J) =
     .                        trackList(numTracks).predState(J)
         DO 275 J = 1,N
            DO 250 K = 1,N
               trackList(numTracks).predP(J,K) = P(J,K)
 250           trackList(numTracks).updP(J,K) = P(J,K)
 275        CONTINUE
         IF (updObs .EQ. 9) THEN
            WRITE (*,*) 'pred state for meas. 9 :',
     .           (trackList(numTracks).predState(J),J=1,N)
            WRITE (*,*) 'upd state for meas. 9 :',
     .           (trackList(numTracks).updState(J),J=1,N)
         END IF
      ELSE  !**** a new observation to update an existing track ****
         numTracks = numTracks + 1
         oldTrack = trackList(prev)
         newTrack = oldTrack
         newTrack.length = oldTrack.length + 1
         newTrack.using = 0
         newTrack.points(newTrack.length) = updObs
         newTrack.new = oldTrack.new
         newTrack.misses = 0
         DO 375 J = 1,N
            newTrack.predState(J) = oldTrack.predState(J)
         DO 350 K = 1,N
 350        newTrack.predP(J,K) = oldTrack.predP(J,K)
 375     CONTINUE
         CALL UpdateState(newTrack.predState,newTrack.predP,updObs,
     .                    newTrack.updState,newTrack.updP)

**** Scoring the track ****
         IF (oldTrack.status .EQ. 'P') THEN
            newTrack.status = 'T'
         END IF
         newTrack.score = newTrack.score + distance
         IF (((newTrack.length) .GT. 2) .AND.
     .        (newTrack.status .NE. 'B'))
     .        newTrack.status = 'C'
         trackList(numTracks) = newTrack

         IF (Idebug .GE. 2) THEN
            WRITE (IDBUGO,*) 'Dummy is ',dummy
            WRITE (IDBUGO,500) 'V: ',(V(1,I),I=1,M)
            WRITE (IDBUGO,500) '   ',(V(2,I),I=1,M)
            WRITE (IDBUGO,500) '   ',(V(3,I),I=1,M)
         END IF
      END IF

 500  FORMAT (A,3F15.5)

      RETURN
      END  !**** MakeTrack ****

*-----------------------------------------------------------------------+
*     SUBROUTINE UpdateState - update the state with the new measurement|
*-----------------------------------------------------------------------+

      SUBROUTINE UpdateState(oldState,oldP,updObs,newState,newP)
```

```
      IMPLICIT REAL*8 (A-H,O-Z)

      PARAMETER (N=6,M=3,D2R=.017453293D0,R2D=1D0/D2R,
     .   pi=3.141592653589793D0, maxLength=100,maxObs=1000,
     .   maxSize=50,maxTrack=10000,IbigPrune=1000,IlittlePrune=10,
     .   nScan=5)

      COMMON /CDEBUG/Idebug
      COMMON /CFILES/IDATA, IDBUGO, IPICT, IMODEL
      COMMON /COBSERVATIONS/Observations,numObs
      COMMON /CGATE/gatesize, distance
      COMMON /CMODEL/Q(N,N), R(M,M), P(N,N), V(M,M), Vinv(M,M)
      COMMON /CSENSOR/GBR(M), Xs(N), sensorTime

      DIMENSION Observations(maxObs,M+1)

      REAL*8 newState,newP
      DIMENSION oldState(N),oldP(N,N),newState(N),newP(N,N)
      DIMENSION H(M,N),gain(N,M),prod1(N,M),prod2(N,N),tempP(N,N),
     .          rae(M),enu(M),corr(N),resid(M)
      INTEGER updObs,I

**** Calculate observation matrix H ****
      CALL HJRAE(oldState,Xs,H,N,M)

**** Calculate Kalman Filter Gain ****
      CALL DMXYTF(N,N,oldP,N,M,N,H,M,N,M,prod1,N)
      CALL DMRRRR(N,M,prod1,N,M,M,Vinv,M,N,M,gain,N)

**** Update Covariance matrix P ****
      CALL DMRRRR(N,M,gain,N,M,N,H,M,N,N,prod2,N)
      CALL DMRRRR(N,N,prod2,N,N,N,oldP,N,N,N,tempP,N)
      DO 100 I = 1,N
         DO 100 J = 1,N
 100        newP(I,J) = oldP(I,J)-tempP(I,J)

**** Update the target state estimate using new observation ****
**** First calculate the residual ****
      CALL PENUST(oldState,Xs,rae,enu)
      IF (rae(2) .GT. pi)
     .   rae(2) = rae(2) - 2.D0*pi
      DO 200 I = 1,M
 200     resid(I) = Observations(updObs,I) - rae(I)

**** Then add in the weighted residual to estimate ****
      CALL DMURRV(N,M,gain,N,M,resid,1,N,corr)
      DO 300 I = 1,N
 300     newState(I) = oldState(I) + corr(I)

**** Some debug Info ****
      IF (Idebug .GE. 2) THEN
         WRITE (IDBUGO,500) '-----Measurement Update-----'
         WRITE (IDBUGO,500) 'oldState: ',(oldState(I),I=1,N)
         WRITE (IDBUGO,500) 'newState: ',(newState(I),I=1,N)
         WRITE (IDBUGO,500) 'oldP: ',(oldP(1,I),I=1,N)
         WRITE (IDBUGO,500) '      ',(oldP(2,I),I=1,N)
         WRITE (IDBUGO,500) '      ',(oldP(3,I),I=1,N)
         WRITE (IDBUGO,500) '      ',(oldP(4,I),I=1,N)
         WRITE (IDBUGO,500) '      ',(oldP(5,I),I=1,N)
         WRITE (IDBUGO,500) '      ',(oldP(6,I),I=1,N)
         WRITE (IDBUGO,500) 'newP: ',(newP(1,I),I=1,N)
         WRITE (IDBUGO,500) '      ',(newP(2,I),I=1,N)
         WRITE (IDBUGO,500) '      ',(newP(3,I),I=1,N)
         WRITE (IDBUGO,500) '      ',(newP(4,I),I=1,N)
         WRITE (IDBUGO,500) '      ',(newP(5,I),I=1,N)
         WRITE (IDBUGO,500) '      ',(newP(6,I),I=1,N)
         WRITE (IDBUGO,501) 'gain: ',(gain(1,I),I=1,M)
         WRITE (IDBUGO,501) '      ',(gain(2,I),I=1,M)
         WRITE (IDBUGO,501) '      ',(gain(3,I),I=1,M)
         WRITE (IDBUGO,501) '      ',(gain(4,I),I=1,M)
         WRITE (IDBUGO,501) '      ',(gain(5,I),I=1,M)
         WRITE (IDBUGO,501) '      ',(gain(6,I),I=1,M)
      END IF

 500  FORMAT (A,6F15.5)
 501  FORMAT (A,3F15.5)

      RETURN
      END  !**** Update State ****

*-----------------------------------------------------------------------+
*     SUBROUTINE IncorporateTrack - incorporates this track into all    |
*        hypotheses that contain the previous track.                    |
*-----------------------------------------------------------------------+
```

```
      SUBROUTINE IncorporateTrack(prev,cur,oldHypos,numOldHypos,
     .                            newHypos,numNewHypos)

      IMPLICIT REAL*8 (A-H,O-Z)

      PARAMETER (N=6,M=3,D2R=.017453293D0,R2D=1D0/D2R,
     .   pi=3.1415926535897932D0, maxLength=100,maxObs=1000,
     .   maxSize=50,maxTrack=10000,IbigPrune=1000,IlittlePrune=10,
     .   nScan=5)

      COMMON /CDEBUG/Idebug
      COMMON /CFILES/IDATA, IDBUGO, IPICT, IMODEL
      COMMON /CTRACKLIST/trackList,numTracks

      STRUCTURE /TRACK/
        CHARACTER status
        INTEGER length,using,points(maxLength),new,misses
        REAL*8 score,predState(N),updState(N),predP(N,N),updP(N,N)
        END STRUCTURE
      RECORD /TRACK/trackList(maxTrack)
      INTEGER numTracks

      STRUCTURE /HYPOTHESIS/
        INTEGER numTracks,hypoTracks(maxSize)
        REAL*8 score,complexity
        END STRUCTURE
      RECORD /HYPOTHESIS/oldHypos(IbigPrune),newHypos(IbigPrune),htemp

      INTEGER prev,cur,numOldHypos,numNewHypos,I,J,K,temp

      IF (prev .EQ. 0) THEN  !**** a new potential track is being added ****
         DO 100 I = 1,numOldHypos
            htemp = oldHypos(I)
            htemp.numTracks = htemp.numTracks + 1
            htemp.hypoTracks(htemp.numTracks) = cur
            CALL CalculateHypothesisScore(htemp)
**** Adjust the using field of the track ****
            DO 50 J = 1,htemp.numTracks
               temp = htemp.hypoTracks(J)
 50            trackList(temp).using = trackList(temp).using + 1
            CALL AddToHypos(htemp,newHypos,numNewHypos)
 100     CONTINUE
      ELSE
         DO 300 I = 1,numOldHypos
            htemp = oldHypos(I)
            DO 200 J = 1,htemp.numTracks
               IF (htemp.hypoTracks(J) .EQ. prev) THEN
                  htemp.hypoTracks(J) = cur
                  CALL CalculateHypothesisScore(htemp)
c                  IF ((trackList(cur).points(trackList(cur).length) .GE.
c     .                 37) .AND. (trackList(cur).points(trackList(cur).
c     .                 length) .LE. 46)) THEN
c                     WRITE (*,*) 'Track score: ',trackList(cur).score
c                     WRITE (*,*) 'Hypothesis score: ',htemp.score
c                     WRITE (*,*) 'Hypothesis complexity: ',
c     .                    htemp.complexity
c                  END IF
**** Adjust the using field of the track ****
                  DO 150 K = 1,htemp.numTracks
                     temp = htemp.hypoTracks(K)
 150                 trackList(temp).using = trackList(temp).using + 1
                  CALL AddToHypos(htemp,newHypos,numNewHypos)
               END IF
 200        CONTINUE
 300     CONTINUE
      END IF

      RETURN
      END  !**** IncorporateTrack ****

*-----------------------------------------------------------------------+
*     SUBROUTINE AddToHypos - add hypo to list so that there is no      |
*                             overflow                                  |
*-----------------------------------------------------------------------+

      SUBROUTINE AddToHypos(theHypo,hypos,numHypos)

      IMPLICIT REAL*8 (A-H,O-Z)

      PARAMETER (N=6,M=3,D2R=.017453293D0,R2D=1D0/D2R,
     .   pi=3.141592653897932D0, maxLength=100,maxObs=1000,
     .   maxSize=50,maxTrack=10000,IbigPrune=1000,IlittlePrune=10,
     .   nScan=5)

      COMMON /CDEBUG/Idebug
```

```
      COMMON /CFILES/IDATA, IDBUGO, IPICT, IMODEL
      COMMON /CTRACKLIST/trackList,numTracks

      STRUCTURE /TRACK/
        CHARACTER status
        INTEGER length,using,points(maxLength),new,misses
        REAL*8 score,predState(N),updState(N),predP(N,N),updP(N,N)
        END STRUCTURE
      RECORD /TRACK/trackList(maxTrack)
      INTEGER numTracks

      STRUCTURE /HYPOTHESIS/
        INTEGER numTracks,hypoTracks(maxSize)
        REAL*8 score,complexity
        END STRUCTURE
      RECORD /HYPOTHESIS/hypos(IbigPrune),theHypo

      INTEGER numHypos,index,I,temp
      REAL*8 max

      IF (numHypos .LT. IbigPrune) THEN
         numHypos = numHypos + 1
         hypos(numHypos) = theHypo
      ELSE
         index = 1
         max = hypos(1).score
         DO 100 I = 2,numHypos
            IF (hypos(I).score .GT. max) THEN
               index = I
               max = hypos(I).score
            END IF
 100     CONTINUE
**** Adjust using field of tracks in hypothesis that is to be deleted ****
         IF (max .GT. theHypo.score) THEN
            DO 200 I = 1,hypos(index).numTracks
               temp = hypos(index).hypoTracks(I)
 200           trackList(temp).using = trackList(temp).using - 1
            hypos(index) = theHypo
         ELSE
            DO 300 I = 1,theHypo.numTracks
               temp = theHypo.hypoTracks(I)
 300           trackList(temp).using = trackList(temp).using - 1
         END IF
      END IF

      RETURN
      END  !**** AddToHypos ****

*-----------------------------------------------------------------------+
*     FUNCTION Gate - does observation gate with the track              |
*-----------------------------------------------------------------------+

      FUNCTION Gate(updObs,trackNum)

      IMPLICIT REAL*8 (A-H,O-Z)

      PARAMETER (N=6,M=3,D2R=.017453293D0,R2D=1D0/D2R,
     .   pi=3.1415926535897932D0, maxLength=100,maxObs=1000,
     .   maxSize=50,maxTrack=10000,IbigPrune=1000,IlittlePrune=10,
     .   nScan=5)

      COMMON /CDEBUG/Idebug

      COMMON /CFILES/IDATA, IDBUGO, IPICT, IMODEL
      COMMON /CTRACKLIST/trackList,numTracks
      COMMON /COBSERVATIONS/Observations,numObs
      COMMON /CGATE/gatesize, distance
      COMMON /CMODEL/Q(N,N), R(M,M), P(N,N), V(M,M), Vinv(M,M)
      COMMON /CSENSOR/GBR(M), Xs(N), sensorTime

      DIMENSION Observations(maxObs,M+1)

      STRUCTURE /TRACK/
        CHARACTER status
        INTEGER length,using,points(maxLength),new,misses
        REAL*8 score,predState(N),updState(N),predP(N,N),updP(N,N)
        END STRUCTURE
      RECORD /TRACK/trackList(maxTrack)
      INTEGER numTracks

      INTEGER updObs,trackNum
      DIMENSION H(M,N),Xt(N),tempP(N,N),prod1(N,M),HPHT(M,M),
     .          rae(M),enu(M),resid(M)
      LOGICAL Gate
```

```
**** Calculate Observation Matrix H ****
      DO 100 I = 1,N
 100     Xt(I) = trackList(trackNum).predState(I)
      CALL HJRAE(Xt,Xs,H,N,M)

**** Calculate V (the innovations variance) ****
      DO 200 I = 1,N
         DO 200 J = 1,N
 200        tempP(I,J) = trackList(trackNum).predP(I,J)
      CALL DMXYTF(N,N,tempP,N,M,N,H,M,N,M,prod1,N)
      CALL DMRRRR(M,N,H,M,N,M,prod1,N,M,M,HPHT,M)
      DO 300 I = 1,M
         DO 300 J = 1,M
 300        V(I,J) = HPHT(I,J) + R(I,J)
      CALL DLINRG(M,V,M,Vinv,M)

**** Compute chi-square distance between prediction and measurement ****
      CALL PENUST(Xt,Xs,rae,enu)
      IF (rae(2) .GT. pi) rae(2) = rae(2) - 2.D0*pi
      IF (rae(2) .LT. -pi) rae(2) = rae(2) + 2.D0*pi
      DO 400 I = 1,M
 400     resid(I) = Observations(updObs,I) - rae(I)
      distance = DBLINF(M,M,Vinv,M,resid,resid)
      Gate = distance .LE. gatesize
c     distance = distance / 50.D0 !**** this fudge factor makes any ****
c                                 !**** track(length>1) score positive *
c               !**** no longer necessary with new scoring function ****

**** Print some debug info ****
      IF (Idebug .GE. 2) THEN
         IF (Gate) THEN
            WRITE (IDBUGO,*) '---Track ',trackNum,' gates with obs ',
     .                        updObs,'---'
         ELSE
            WRITE (IDBUGO,*) '---Track ',trackNum,' does NOT gate ',
     .                       'with obs ',updObs,'---'
         END IF
         WRITE (IDBUGO,*) 'Normalized distance: ',distance
      END IF

      RETURN
      END  !**** Gate ****

*-----------------------------------------------------------------------+
*     SUBROUTINE PrintSummary - print a summary of the tracks and       |
*                               hypotheses                              |
*-----------------------------------------------------------------------+

      SUBROUTINE PrintSummary(hypos,numHypos)

      IMPLICIT REAL*8 (A-H,O-Z)

      PARAMETER (N=6,M=3,D2R=.0174532930,R2D=1D0/D2R,
     .   pi=3.1415926535897932D0, maxLength=100,maxObs=1000,
     .   maxSize=50,maxTrack=10000,IbigPrune=1000,IlittlePrune=10,
     .   nScan=5)


      COMMON /CDEBUG/Idebug
      COMMON /CFILES/IDATA, IDBUGO, IPICT, IMODEL
      COMMON /CTRACKLIST/trackList,numTracks

      STRUCTURE /TRACK/
        CHARACTER status
        INTEGER length,using,points(maxLength),new,misses
        REAL*8 score,predState(N),updState(N),predP(N,N),updP(N,N)
        END STRUCTURE
      RECORD /TRACK/trackList(maxTrack),curTrack
      INTEGER numTracks

      STRUCTURE /HYPOTHESIS/
        INTEGER numTracks,hypoTracks(maxSize)
        REAL*8 score,complexity
        END STRUCTURE
      RECORD /HYPOTHESIS/hypos(IbigPrune),htemp

      INTEGER numHypos,TLOOP,HLOOP,numShow

      IF (numHypos .LT. 10) THEN
         numShow = numHypos
      ELSE IF (Idebug .EQ. -1) THEN
         numShow = 10
      ELSE
         numShow = numHypos
      END IF
```

```
      WRITE (IDBUGO,*) 'Number of Tracks:     ',numTracks
      WRITE (IDBUGO,*) 'Number of Hypotheses: ',numHypos
      WRITE (IDBUGO,*) ' Track Summary'
      WRITE (IDBUGO,*) ' Track       Observations
     .  Status      Quality'
      DO 520 TLOOP = 1,numTracks
        curTrack = trackList(TLOOP)
        IF (curTrack.status .EQ. 'B') THEN
           WRITE(IDBUGO,902) TLOOP,(curTrack.points(I),I=1,10)
     .          , curTrack.status, curTrack.score, curTrack.new
        ELSE
           WRITE(IDBUGO,900) TLOOP,(curTrack.points(I),I=1,10)
     .          , curTrack.status, curTrack.score
        END IF
        DO 510 J = 1,9
           IF (curTrack.length .GT. J*10)
     .        WRITE (IDBUGO,901) (curTrack.points(I),
     .              I=J*10+1,(J+1)*10)
510     CONTINUE
520   CONTINUE

      WRITE (IDBUGO,*)' '
      WRITE (IDBUGO,*)' Hypothesis Summary'
      WRITE (IDBUGO,*)' Hypothesis      Tracks
     .            Cost   Complexity'
      DO 540 HLOOP = 1,numShow
        htemp = hypos(HLOOP)
        WRITE (IDBUGO,920) HLOOP,(htemp.hypotracks(I),I=1,10),
     .      htemp.SCORE, htemp.complexity
        DO 530 J = 1,4
           IF (htemp.numTracks.GT.J*10)
     .        WRITE (IDBUGO,910) (htemp.hypotracks(I),
     .              I=J*10+1,(J+1)*10)
530     CONTINUE
540   CONTINUE
      WRITE (IDBUGO,*)' '
      WRITE (IDBUGO,*)' '

900   FORMAT (I4,'   |',10I4,' |  ',A,'  | ',F13.5)
901   FORMAT ('       |',10I4,' |     | ')
902   FORMAT (I4,'   |',10I4,' |  ',A,'  | ',F13.5,' orig:',I4)
910   FORMAT (13X,'|',10I4,' | ')
920   FORMAT (I6,7X,'|',10I4,' | ',F14.5,' ',F4.0)

      RETURN
      END  !**** PrintSummary ****

*-------------------------------------------------------------------------+
*     SUBROUTINE CheckMisses - check to see if any of the hypotheses are  |
*                      using the old tracks that existed before this      |
*                      scan was ever analyzed.  These tracks have a       |
*                      miss.                                              |
*-------------------------------------------------------------------------+

      SUBROUTINE CheckMisses(numOldTracks,hypos,numHypos)

      IMPLICIT REAL*8 (A-H,O-Z)

      PARAMETER (N=6,M=3,D2R=.017453293D0,R2D=1D0/D2R,
     .   pi=3.1415926535897932D0, maxLength=100,maxObs=1000,
     .   maxSize=50,maxTrack=10000,IbigPrune=1000,IlittlePrune=10,
     .   nScan=5)

      PARAMETER (DnewTarget=5.D-2,DfalseTarget=5.D-2,Pdetect=.95D0,
     .           penalty=-2.9957322735540D0,LTlimit=3,FAlimit=4.D-2)

      COMMON /CDEBUG/Idebug
      COMMON /CFILES/IDATA, IDBUGO, IPICT, IMODEL
      COMMON /CTRACKLIST/trackList,numTracks

      STRUCTURE /TRACK/
        CHARACTER status
        INTEGER length,using,points(maxLength),new,misses
        REAL*8 score,predState(N),updState(N),predP(N,N),updP(N,N)
        END STRUCTURE
      RECORD /TRACK/trackList(maxTrack)
      INTEGER numTracks

      STRUCTURE /HYPOTHESIS/
        INTEGER numTracks,hypoTracks(maxSize)
        REAL*8 score,complexity
        END STRUCTURE
      RECORD /HYPOTHESIS/hypos(IbigPrune),htemp
```

```
      INTEGER numOldTracks,numHypos,I,J,K

      DO 200 I = 1,numOldTracks
         IF ((trackList(I).using .GT. 0) .AND.
     .       (trackList(I).status .NE. 'F') .AND.
     .       (trackList(I).status .NE. 'L') .AND.
     .       (trackList(I).status .NE. 'D')) THEN
            trackList(I).misses = trackList(I).misses + 1
            trackList(I).length = trackList(I).length + 1
            trackList(I).points(trackList(I).length) = -1
            DO 100 J = 1,N
               trackList(I).updState(J) = trackList(I).predState(J)
               DO 100 K = 1,N
                  trackList(I).updP(J,K) = trackList(I).predP(J,K)
 100           CONTINUE
            IF ((trackList(I).status .EQ. 'P') .OR.
     .          (trackList(I).status .EQ. 'U')) THEN
               trackList(I).status = 'P'
               IF ((trackList(I).length .EQ. 3) .AND.
     .             (trackList(I).misses .EQ. 2)) THEN   !**** false alarm ****
                  trackList(I).status = 'F'
                  trackList(I).length = 1
               END IF
            ELSE   !**** if not a potential track ****
               IF ((trackList(I).misses .GE. LTlimit) .AND.
     .             ((trackList(I).status .EQ. 'T') .OR.
     .              (trackList(I).status .EQ. 'C'))) THEN !**** lost track ***
                  trackList(I).status = 'L'
               ELSE IF ((trackList(I).misses .GE. LTlimit) .AND.
     .                  (trackList(I).status .EQ. 'B')) THEN
                  trackList(I).status = 'D'
               END IF
            END IF
         END IF
 200  CONTINUE

**** Update all the hypothesis scores ****
      DO 300 I = 1,numHypos
         htemp = hypos(I)
         CALL CalculateHypothesisScore(htemp)
         hypos(I) = htemp
 250     CONTINUE
 300  CONTINUE

      RETURN
      END  !**** CheckMisses ****

*-----------------------------------------------------------------------+
*     SUBROUTINE PackTracks - delete any unused tracks and pack them    |
*         down to lowest index.  Change the numbers of the tracks in the|
*         hypotheses also.                                              |
*-----------------------------------------------------------------------+

      SUBROUTINE PackTracks(hypos,numHypos)

      IMPLICIT REAL*8 (A-H,O-Z)

      PARAMETER (N=6,M=3,D2R=.017453293D0,R2D=1D0/D2R,
     .   pi=3.1415926535897932D0, maxLength=100,maxObs=1000,
     .   maxSize=50,maxTrack=10000,IbigPrune=1000,IlittlePrune=10,
     .   nScan=5)

      PARAMETER (DnewTarget = 5.D-2, DfalseTarget = 5.D-2)

      COMMON /CDEBUG/Idebug
      COMMON /CFILES/IDATA, IDBUGO, IPICT, IMODEL
      COMMON /CTRACKLIST/trackList,numTracks

      STRUCTURE /TRACK/
        CHARACTER status
        INTEGER length,using,points(maxLength),new,misses
        REAL*8 score,predState(N),updState(N),predP(N,N),updP(N,N)
        END STRUCTURE
      RECORD /TRACK/trackList(maxTrack)
      INTEGER numTracks

      STRUCTURE /HYPOTHESIS/
        INTEGER numTracks,hypoTracks(maxSize)
        REAL*8 score,complexity
        END STRUCTURE
      RECORD /HYPOTHESIS/hypos(IbigPrune)


      INTEGER numHypos,I,new,temp
      DIMENSION NewLocations(maxTrack)
```

```
**** Pack the tracks saving the new locations for the moved tracks ****
      new = 0
      DO 100 I = 1,numTracks
         IF (trackList(I).using .LE. 0) THEN
            IF (Idebug .GE. 2) THEN
               IF (trackList(I).using .LT. 0) THEN
                  WRITE (IDBUGO,*) 'Track ',I,' is messed up.'
               END IF
               WRITE (IDBUGO,*) 'PackTracks - deleted track ',I
            END IF
         ELSE
            new = new + 1
            trackList(new) = trackList(I)
            NewLocations(I) = new
            IF (Idebug .GE. 2) THEN
               WRITE (IDBUGO,*) 'PackTracks - Track ',I,' is now ',new
            END IF
         END IF
 100  CONTINUE
      numTracks = new

**** Iterate through all of the hypotheses to change tracks that have moved **
      DO 300 I = 1,numHypos
         DO 200 J = 1,hypos(I).numTracks
            temp = hypos(I).hypoTracks(J)
            hypos(I).hypoTracks(J) = NewLocations(temp)
 200     CONTINUE
 300  CONTINUE

      RETURN
      END  ! **** PackTracks ****

*-----------------------------------------------------------------------+
*     SUBROUTINE PruneHypotheses - keep only the highest num hypotheses |
*                    and sort them ... highest score first              |
*-----------------------------------------------------------------------+

      SUBROUTINE PruneHypotheses(hypos,numHypos,num)

      IMPLICIT REAL*8 (A-H,O-Z)

      PARAMETER (N=6,M=3,D2R=.017453293D0,R2D=1D0/D2R,
     .   pi=3.1415926535897932D0, maxLength=100,maxObs=1000,
     .   maxSize=50,maxTrack=10000,IbigPrune=1000,IlittlePrune=10,
     .   nScan=5)

      COMMON /CDEBUG/Idebug
      COMMON /CFILES/IDATA, IDBUGO, IPICT, IMODEL
      COMMON /CTRACKLIST/trackList,numTracks

      STRUCTURE /TRACK/
        CHARACTER status
        INTEGER length,using,points(maxLength),new,misses
        REAL*8 score,predState(N),updState(N),predP(N,N),updP(N,N)
        END STRUCTURE
      RECORD /TRACK/trackList(maxTrack)
      INTEGER numTracks

      STRUCTURE /HYPOTHESIS/
        INTEGER numTracks,hypoTracks(maxSize)
        REAL*8 score,complexity
        END STRUCTURE
      RECORD /HYPOTHESIS/hypos(IbigPrune),htemp

      INTEGER num,numHypos,I,J,index,temp

**** Sort the hypotheses list ****
      DO 200 I = 1,numHypos-1
         htemp = hypos(I)
         index = I
         DO 100 J = I+1,numHypos
            IF (hypos(J).score .LT. htemp.score) THEN
               htemp = hypos(J)
               index = J
            END IF
 100     CONTINUE

**** Swap the I-th and index-th hypotheses ****
         hypos(index) = hypos(I)
         hypos(I) = htemp
 200  CONTINUE

**** Add extra hypotheses if their complexities are <= best hypothesis ****
 220  IF ((num .LT. numHypos) .AND.
```

```
     .    (hypos(num+1).complexity .LE. hypos(1).complexity)) THEN
          num = num + 1
          GOTO 220
       END IF
**** Erase rest of hypotheses ****
       DO 400 I = num+1,numHypos
          DO 300 J = 1,hypos(I).numTracks
             temp = hypos(I).hypoTracks(J)
 300         trackList(temp).using = trackList(temp).using - 1
          hypos(I).numTracks = 0
          hypos(I).score = 0D0
 400   CONTINUE
       IF (numHypos .GT. num) numHypos = num

       RETURN
       END  !**** PruneHypotheses ****

*-----------------------------------------------------------------------+
*      SUBROUTINE CalculateHypothesisScore - calculate the complexity and|
*                  the score of the hypothesis                          |
*-----------------------------------------------------------------------+

       SUBROUTINE CalculateHypothesisScore(htemp)

       IMPLICIT REAL*8 (A-H,O-Z)

       PARAMETER (N=6,M=3,D2R=.017453293D0,R2D=1D0/D2R,
      .   pi=3.1415926535897932D0, maxLength=100,maxObs=1000,
      .   maxSize=50,maxTrack=10000,IbigPrune=1000,IlittlePrune=10,
      .   nScan=5)

       COMMON /CDEBUG/Idebug
       COMMON /CFILES/IDATA, IDBUGO, IPICT, IMODEL
       COMMON /CTRACKLIST/trackList,numTracks

       STRUCTURE /TRACK/
         CHARACTER status
         INTEGER length,using,points(maxLength),new,misses
         REAL*8 score,predState(N),updState(N),predP(N,N),updP(N,N)
         END STRUCTURE
       RECORD /TRACK/trackList(maxTrack),tempTrack
       INTEGER numTracks

       STRUCTURE /HYPOTHESIS/
         INTEGER numTracks,hypoTracks(maxSize)
         REAL*8 score,complexity
         END STRUCTURE
       RECORD /HYPOTHESIS/htemp

       INTEGER BreakupOrigins(maxObs),I

       DO 100 I=1,maxObs
 100     BreakupOrigins(I) = 0
       htemp.score = 0D0
       htemp.complexity = 0D0

       DO 500 I=1,htemp.numTracks
          tempTrack = trackList(htemp.hypoTracks(I))
          htemp.score = htemp.score + tempTrack.score
          IF (tempTrack.status .EQ. 'F') THEN
             htemp.complexity = htemp.complexity + 4D0
          ELSEIF ((tempTrack.status .EQ. 'C') .OR.
      .           (tempTrack.status .EQ. 'T') .OR.
      .           (tempTrack.status .EQ. 'P') .OR.
      .           (tempTrack.status .EQ. 'U')) THEN
             htemp.complexity = htemp.complexity + 7D0
          ELSEIF (tempTrack.status .EQ. 'L') THEN
             htemp.complexity = htemp.complexity + 8D0
          ELSEIF (tempTrack.status .EQ. 'B') THEN
             IF (BreakupOrigins(tempTrack.new) .EQ. 0) THEN
                htemp.complexity = htemp.complexity + 4D0
                BreakupOrigins(tempTrack.new) = 1
             ELSE
                htemp.complexity = htemp.complexity + 3D0
             END IF
          ELSEIF (tempTrack.status .EQ. 'D') THEN
             IF (BreakupOrigins(tempTrack.new) .EQ. 0) THEN
                htemp.complexity = htemp.complexity + 6D0
                BreakupOrigins(tempTrack.new) = 1
             ELSE
                htemp.complexity = htemp.complexity + 4D0
             END IF
          ELSE
          END IF
          htemp.complexity = htemp.complexity + tempTrack.misses
```

```
 500  CONTINUE

      htemp.score = htemp.score + 2D0 * htemp.complexity

      RETURN
      END  !**** CalculateHypothesisScore ****

*-----------------------------------------------------------------------+
*     FUNCTION NewTrackPossibility - is the velocity part of the state  |
*                 vector small enough                                   |
*-----------------------------------------------------------------------+

      FUNCTION NewTrackPossibility(aTrack)

      IMPLICIT REAL*8 (A-H,O-Z)

      PARAMETER (N=6,M=3,D2R=.017453293D0,R2D=1D0/D2R,
     .   pi=3.141592653589793 2D0, maxLength=100,maxObs=1000,
     .   maxSize=50,maxTrack=10000,IbigPrune=1000,IlittlePrune=10,
     .   nScan=5)

      COMMON /CGATE/gatesize, distance

      STRUCTURE /TRACK/
        CHARACTER status
        INTEGER length,using,points(maxLength),new,misses
        REAL*8 score,predState(N),updState(N),predP(N,N),updP(N,N)
        END STRUCTURE
      RECORD /TRACK/aTrack

      LOGICAL NewTrackPossibility
      REAL*8 speed

      speed = aTrack.predState(4)**2D0 + aTrack.predState(6)**2D0
     .        + aTrack.predState(6)**2D0
      IF (speed .LT. (16D0 * gatesize)) THEN
         NewTrackPossibility = .TRUE.
      ELSE
         NewTrackPossibility = .FALSE.
      END IF

      RETURN
      END   !**** NewTrackPossibility ****

*-----------------------------------------------------------------------+
*     FUNCTION MakeBreakupTrack - make breakup track if possible with   |
*                 everything already estimated                          |
*-----------------------------------------------------------------------+

      FUNCTION MakeBreakupTrack(breakup,orig,newMeas,T1,T2)

      IMPLICIT REAL*8 (A-H,O-Z)

      PARAMETER (N=6,M=3,D2R=.017453293D0,R2D=1D0/D2R,
     .   pi=3.141592653589793 2D0, maxLength=100,maxObs=1000,
     .   maxSize=50,maxTrack=10000,IbigPrune=1000,IlittlePrune=10,
     .   nScan=5)

      COMMON /CMODEL/Q(N,N), R(M,M), P(N,N), V(M,M), Vinv(M,M)

      STRUCTURE /TRACK/
        CHARACTER status
        INTEGER length,using,points(maxLength),new,misses
        REAL*8 score,predState(N),updState(N),predP(N,N),updP(N,N)
        END STRUCTURE
      RECORD /TRACK/breakup,orig

      LOGICAL MakeBreakupTrack
      INTEGER newMeas,I,J,K
      REAL*8 T1,T2

      DIMENSION state(N),first(N),diff(M),pred(N)

      CALL FindObsState(newMeas,state)
      IF (newMeas .EQ. 13) THEN
         WRITE (*,*) 'State for meas. 13: ',(state(J),J=1,N)
      END IF
      CALL FindObsState(orig.points(orig.length),state)
      IF (newMeas .EQ. 13) THEN
         WRITE (*,*) 'State for last meas. of track ',
     .        orig.points(orig.length),': ',(state(J),J=1,N)
      END IF
      CALL FindObsState(newMeas,state)
      IF (newMeas .EQ. 13) THEN
         WRITE (*,*) 'State for meas. 13 again: ',(state(J),J=1,N)
```

```
      END IF

      IF (orig.points(orig.length) .NE. -1) THEN
         breakup = orig

**** First predict velocity ****
         CALL FindObsState(orig.points(orig.length),breakup.predState)
         DO 150 J=1,M
            first(J) = breakup.predState(J)
 150        first(J+M) = 0D0
            IF (newMeas .EQ. 13) THEN
               WRITE (*,*) 'initial State for first: ',
     .              (first(K),K=1,N)
            END IF

         DO 200 J=1,M
 200        diff(J) = (state(J) - first(J))/(T2-T1)

 250     CONTINUE
         DO 300 J=1,M
 300        first(J+M) = diff(J) + first(J+M)
         DO 350 J=1,M
 350        first(J) = breakup.predState(J)
         CALL MOVESV(T1,first,T2,pred)
         DO 400 J=1,M
 400        diff(J) = (state(J)-pred(J))/(T2-T1)
         IF ((diff(1) .GE. 1D-5) .OR.
     .        (diff(2) .GE. 1D-5) .OR.
     .        (diff(3) .GE. 1D-6)) THEN
            GOTO 250
         END IF
         DO 450 J=1,N
            breakup.predState(J) = first(J)
 450        breakup.updState(J) = first(J)

         IF (newMeas .EQ. 13) THEN
            WRITE (*,*) 'State for first: ',(first(J),J=1,N)
         END IF


         IF (NewTrackPossibility(breakup)) THEN
            MakeBreakupTrack = .TRUE.
            breakup.status = 'B'
            breakup.length = 1
            breakup.using = 0
            breakup.new = orig.points(orig.length)
            breakup.misses = 0
            breakup.score = 0D0
            breakup.points(1) = newMeas
            DO 500 J=2,maxLength
 500           breakup.points(J) = 0
            DO 600 I = 1,N
               DO 550 J = 1,N
                  breakup.predP(I,J) = P(I,J)
 550              breakup.updP(I,J) = P(I,J)
 600        CONTINUE
c            CALL MOVESV(T1,first,T2,breakup.predState)
c            DO 700 I = 1,N
c 700           breakup.updState(I) = breakup.predState(I)
         ELSE
            MakeBreakupTrack = .FALSE.
         END IF
      ELSE
         MakeBreakupTrack = .FALSE.
      END IF

      RETURN
      END  !**** MakeBreakupTrack ****
```

Bibliography

[1] H. Akaike. A new look at the statistical model identification. *IEEE Trans. on Automat. Contr.*, AC–19(6):716–722, December 1974.

[2] Y. Bar-Shalom. Extension of the probabilistic data association filter in multi-target tracking. In *Proc. 5th Symp. on Nonlinear Estimation*, pages 16–21, Sept. 1974.

[3] Y. Bar-Shalom and E. Tse. Tracking in a cluttered environment with probabilistic data association. *Automatica*, 11:451–460, 1975.

[4] S. S. Blackman. *Multiple-Target Tracking with Radar Applications.* Artech House, Inc., Dedham, MA, 1986.

[5] A. Gelb, editor. *Applied Optimal Estimation.* The Analytic Sciences Corporation, Cambridge, MA, 1984.

[6] G. Minkler and J. Minkler. *Aerospace Coordinate Systems and Transformations.* Magellan Book Company, Baltimore, MD, 1990.

[7] J. B. Pearson. On nonlinear least-squares filtering. *Automatica*, 4:97–105, 1967.

[8] D. B. Reid. An algorithm for tracking multiple targets. *IEEE Trans. on Automat. Contr.*, AC–24(6):843–854, December 1979.

[9] J. Rissanen. *Stochastic Complexity in Statistical Inquiry.* World Scientific Publishing Co Pte Ltd., Singapore, 1989.

[10] A. S. Willsky and H. L. Jones. A generalized likelihood ratio approach to state estimation in linear systems subject to abrupt changes. In *Proc. 1974 IEEE Conf. Decision and Control*, pages 846–853, November 1974.