INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

An IMM-JVC Algorithm for Multi-Target Tracking with Asynchronous Sensors

Benoit Jarry

Department of Electrical Engineering

McGill University, Montreal

August 2000

A Thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements of the degree of Master of Engineering

Abstract

The tracking of closely maneuvering targets represents a challenge for both the contact-to-track association and the positional estimation algorithms. This thesis describes an IMM-JVC algorithm, its implementation, and demonstrates its performance on some simulated scenarios.

The Interacting Multiple Model (IMM) estimator is integrated with the Jonker-Volgenant-Castanon (JVC) contact-to-track association optimization procedure in the IMM-JVC formulation described. The tracking accuracy provided by the IMM has been recognized as superior to that obtained from other single-scan positional estimators. The IMM provides a probabilistically weighted combination of the results of multiple Kalman filters. The JVC method also uses probability weighting in a global nearest neighbor approach to contact-to-track association. Some emphasis is put on tackling the problem of asynchronous sensors, which may be encountered in the case of multiple sensors.

A number of different approaches, along with the IMM-JVC, are simulated to illustrate their characteristic properties and provide a basis for comparison.

Résumé

Suivre la trajectoire d'une cible manoeuvrante représente un défi pour les algorithmes de l'association de trajectoires et contacts, et de l'estimation de position. Cette thèse présente un algorithme IMM-JVC, sa réalisation, et démontre sa performance avec des scénarios simulés.

L'algorithme d'estimation IMM est intégré avec l'algorithme d'association optimal dans le IMM-JVC. C'est reconnue que l'IMM offre une meilleure précision que d'autres algorithmes de son genre. L'IMM fourni une combinaison pondéré des résultats de plusieurs filtre de Kalman. La méthode JVC utilise une transformation de probabilité faisant plus probable l'association de trajectoires et contacts d'écartement minime. Dans le cas de multiples détecteurs, le problème de détecteurs asynchrones est considérées.

Un nombre de méthodes differentes, en plus du IMM-JVC, sont simulées par ordinateur afin d'en illustrer leurs propriétés caractéristiques et pour réaliser la comparaison.

Acknowledgements

I would like to express my honest gratitude and deepest appreciation to my research supervisor, Prof. Hannah Michalska, without whom this work would not have been possible. I extend my thanks to Lockheed Martin Canada, and my supervisor there, Alexandre Jouan, with whose assistance and guidance this project was conducted and who lent me an LMC terminal for the purposes of my research. I also extend my thanks to LMC and FCAR for generous financial support.

Contents

Abstract i
Résumé ii
Acknowledgements iii
Contents iv
List of Figures v
1 INTRODUCTION 1
1.1 BACKGROUND 1
1.1.1 Estimation, Decision and Tracking 1
1.1.2 Surveillance Systems 4
1.2 TRACKING AND DATA ASSOCIATION 5
1.2.1 Tracking 5
1.2.2 Data Association 6
1.3 THE KALMAN FILTER 8
1.3.1 Kalman Filter Overview 8
1.3.2 Extended Kalman Filter 12
1.4 THE MULTIPLE MODEL APPROACH 14
1.4.1 Overview 14
1.4.2 The Interacting Multiple Model Estimator 15
1.5 REFERENCES 19
2 THE IMM-JVC ALGORITHM 21
2.1 INTRODUCTION 21
2.2 JVC ASSOCIATION 22
2.3 THE IMM MODULE 25
2.4 INTEGRATION OF THE IMM AND JVC 25
2.5 THE "OUT-OF-SEQUENCE" MEASUREMENTS 27
2.6 SIMULATION EXPERIMENTS 28
2.6.1 Scenario 1 – Aircraft Closely Maneuvering 29
2.6.2 Scenario 2 – Noise-free Retrodiction 31
2.6.3 Scenario 3 – Aircraft Closely Maneuvering with Clutter 32
2.7 REFERENCES 38
3 DATA TYPES & AGENTS 40
3.1 THE DECISION AIDS FOR AIRBORNE SURVEILLANCE PROJECT (DAAS) 40
3.2 TESTBED ARCHITECTURE 40
3.3 IMM-JVC IMPLEMENTATION 44
3.4 CONCLUSION 46
3.5 REFERENCES 47

List of Figures

Figure 1.1-1 Mathematical view of state estimation 3
Figure 1.2-1 Components of a tracking system 6
Figure 1.3-1 One cycle of KF 11
Figure 1.3-2 One cycle of EKF 13
Figure 1.4-1 One cycle of the IMM ($i,j = 1,\ldots,r$) 17
Figure 2.6-1 NN association, estimation with adaptive KF or IMM 30
Figure 2.6-2 JVC association, estimation with IMM 31
Figure 2.6-3 Track covariance in presence of out-of-sequence measurements 32
Figure 2.6-4 JVC association, IMM-CVCA with clutter 33
Figure 2.6-5 MSDF track creation and life duration 34
Figure 2.6-6 2D mapping of the number of contacts in track gates 35
Figure 2.6-7 Number of contacts in buffers 36
Figure 3.2-1 Test bed architecture 41
Figure 3.2-2 KBS BB architecture 42
Figure 3.2-3 MSDF components 44

1 INTRODUCTION

1.1 BACKGROUND

In this paper an IMM-JVC algorithm, and its implementation, for multi-target, multi-sensor tracking is described. This algorithm has proven, through computer simulation, to be highly accurate and computationally efficient. Chapter one of this paper presents terms and concepts basic to target tracking. Chapter two of this paper is drawn from two papers co-authored by the current author and presented at separate conferences. These papers describe, for the first time, the IMM-JVC combination.

B. Jarry, A. Jouan, H. Michalska. An IMM-JVC Algorithm for Multi-Target Tracking with Asynchronous Sensors. *Proceedings of the 19th IASTED Conference on Modelling, Information and Control*, Innsbruck, Austria, 2000, pp. 603-607.

A. Jouan, B. Jarry, H. Michalska. Tracking Closely Maneuvering Targets in Clutter with an IMM-JVC Algorithm. *FUSION 2000*, in press.

Chapter three describes the software used for the IMM-JVC implementation.

1.1.1 Estimation, Decision and Tracking

Estimation is the process of inferring the value of a quantity of interest imperfect observations. More rigorously, estimation is the process of selection of a point from a continuous space, as in the "best estimate". Estimation exists to generate information of higher quality than raw measurements and which contains information not directly available in the measurements.

Decision is the selection of one out of a set of discrete alternatives, as in the "best choice" from a discrete space. However, one can consider a discrete-valued estimation with the possibility of obtaining some conditional probabilities for the various alternatives rather than making a definite choice from those alternatives. This information, that is the conditional probabilities, may then be used without making a "hard decision". Therefore, estimation and decision are overlapping, allowing techniques from both areas to be used simultaneously in many practical problems.

Tracking is the estimation of the state of a moving object [1]. This may be done using one or more sensors at fixed locations or on moving platforms, such as on a moving (or stationary) aircraft.

Tracking is wider in scope than estimation. It uses all the tools from estimation but also requires extensive use of statistical/probabilistic decision theory. This is especially true in the case of *data association*. Data association is the process of determining which measurement should be associated with the state of the moving object of interest.

Filtering, in the tracking context, is the estimation of the (current) state of a dynamic system from noisy data [2]. This process amounts to "filtering out" the noise.

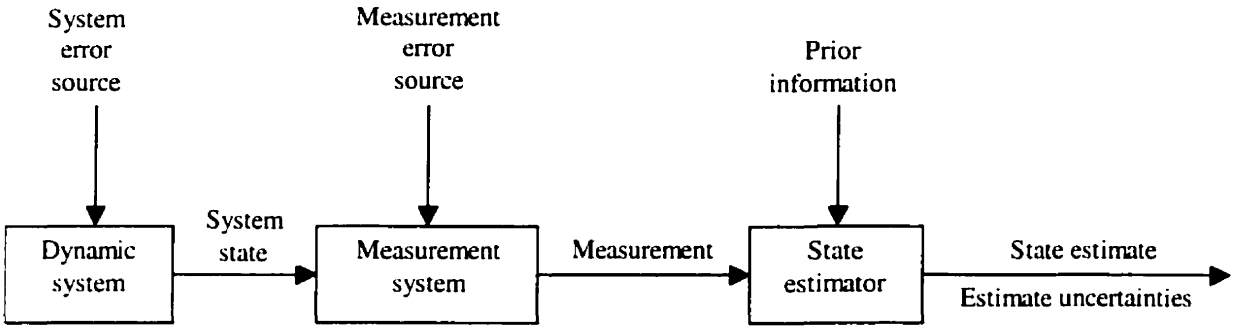

Figure 1.1-1 Mathematical view of state estimation

Figure 1.1.1-1 presents a concise block diagram that illustrates state estimation. The only variables available to the estimator are the measurements. The effects of the error sources on the measurements are accounted for as "noise". In obtaining the state estimate and its uncertainties, the estimator may use knowledge about:

- The evolution of the state (the system dynamics),
- The sensor (measurement system),
- The probabilistic characterization of the various random factors (uncertainties in the system and measurement models and error sources) and of the prior information (earlier measurements).

An *optimal estimator* is a computational algorithm that processes observations (measurements) in order to yield an estimate of a variable of interest that minimizes the selected error criterion (minimum mean square error, etc). An optimal estimator makes best utilization of the data and of knowledge of the system and its disturbances. However, an optimal estimator is possibly sensitive to modeling errors and may be computationally expensive[1].

1.1.2 Surveillance Systems

Due to the widespread use and increasing sophistication of military and civilians surveillance systems there has been a much interest in algorithms capable of tracking large numbers of targets using measurement data from many, possibly diverse, sensors. The effort expended to track n targets can be much more costly than n times the effort expended for a single target due to the complexity of establishing the correspondence between targets and measurements, the data association mentioned above.

From figure 1.1.1-1, the requirement for data association can be seen as occurring between the first and second blocks due to uncertainty in the origin of observations obtained by the sensor. This uncertainty usually arises for a number of factors relating to the nature of the sensors involved. Such sensors, for example radar, sonar or optical, sense energy emitted from or reflected by an object (or several objects) of interest, as well as from other spurious sources of energy, for instance active countermeasures or chaff.

Additional uncertainty in tracking targets can be associated with the origin of the measurements in addition to *inaccuracy*, which is usually modeled as additive noise [4]. Inaccuracy arises due such things as sensor resolution. Uncertainty related to the origin of measurements occurs in a surveillance system when a radar, sonar, or optical sensor is operating in the presence of: clutter, countermeasures, or false alarms. Further, the probability of obtaining a measurement from a target, the *target detection probability*, is usually less than unity.

Origin uncertainty may also occur when several targets are closely spaced and, while it is possible to resolve the each detection, it is not possible to associate these

detections to their sources with certainty. For instance, using radar one may obtain three separate detections for three closely maneuvering aircraft; however, due to the inaccuracy of the measurements it is impossible to determine the correspondence between aircraft and detection. A similar situation occurs in track formation when there are several targets but their number is unknown and some measurements may be spurious.

Application of standard estimation algorithms using a *nearest neighbor* approach can lead to poor results is situations where spurious measurements occur frequently or when separate tracks lie within the range of sensor accuracy. Such an approach does not take into account that the measurement used may not have originated from the target of interest.

1.2 TRACKING AND DATA ASSOCIATION

1.2.1 Tracking

Tracking, as mentioned above, is the processing of measurements obtained from a target in order to maintain an estimate of its current *state*, which is typically:

- Kinematics – position, velocity, acceleration, turn rate, etc.
- Features – radiated signal strength, spectral characteristics, radar cross-section, target classification, etc.
- Constants or slowly varying parameters – aerodynamic parameters, etc.

Measurements are noise-corrupted observations related to the state of a target, such as:

- Direct estimate of position (usually range, azimuth, and elevation),
- Range and azimuth (bearing),
- Bearing only,
- Imaging.

The measurements of interest are not raw data points but are usually the outputs of signal processing and detection subsystems as shown in figure 1.2.1-1.

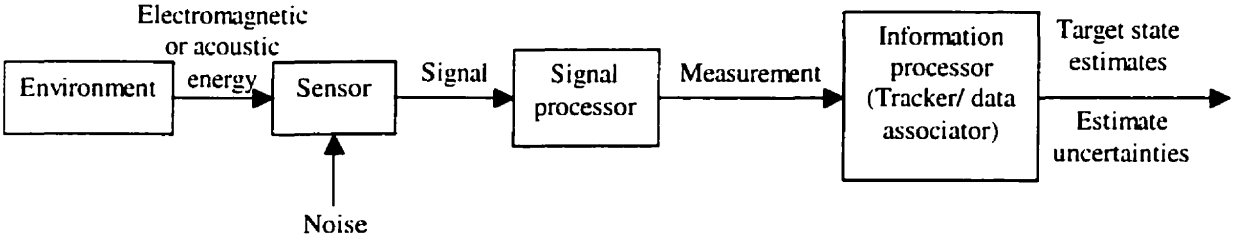

Figure 1.2-1 Components of a tracking system

The *sensor* is a device that observes the environment wherein the target is found through reception of some signals, a frequent example in aircraft surveillance is radar. The raw signal from the sensor is transformed into a measurement by use of a signal processor. For instance, the radiation received by the radar antenna is converted into a measurement providing the range and bearing of a target. To make this process discrete, a *frame* or *scan* is a snapshot of the environment, obtained by the sensor during the sampling period or at the sampling time.

1.2.2 Data Association

A *track* is a state trajectory estimated from a set of measurements -- the *data* in data association -- that has been associated with the same targets [1]. The association process is called for in situations involving multiple targets for measurements of

uncertain origin due to random false alarms, clutter caused by spurious reflectors near the target, other interfering targets, or in the presence of decoys and countermeasures.

Varieties of data association are:

- Measurement to measurement association – for track formation,
- Measurement to track association – for track maintenance and updating,
- Track to track association – for track fusion in multi-sensor systems.

Track formation is the detection of a target, through the processing of measurements from a number of sampling times to determine the presence of a target, and the initiation of its track. *Track maintenance* or *updating* is the association and incorporation of measurements from a sampling time into tracks.

Further, there are two different approaches to data association [2]:

- Non-Bayesian – this approach makes an association decision using statistical methods then ignores that the decision made may not be correct for later decisions,
- Probabilistic (Bayesian) – this approach evaluates probabilities for association and uses these probabilities throughout estimation.

When determining the association probabilities, even for a nearest neighbor approach, the evaluation process relies on models of the targets and sensors providing the measurements. Target models can be easily invalidated through a change of target behavior such as a maneuver, thereby increasing the difficulty in data association.

A number of data association algorithms have been developed, among these are:

- Nearest neighbor – associate one-to-one each measurement/track to the (statistically) nearest track/measurement,
- Global nearest neighbor – associate one-to-one by minimizing a function of the distances between measurements and tracks; one such method is the JVC, described below,
- Probabilistic data association filter – the PDAF comes in a number of flavors, such as PDAF, JPDAF, IPDAF, etc; all depend on a probabilistically weighted all-neighbors approach [1].

1.3 THE KALMAN FILTER

A commonly used approach to filtering and prediction for multi-target tracking is the Kalman filter. Kalman filtering generates time-variable tracking coefficients that are determined by *a priori* models for the statistics of measurement noise and target dynamics [3]; that is, the state is predicted using known models for the noise and dynamics. With expanding computer capabilities, the high-accuracy tracking of the Kalman filter is giving way to more accurate, flexible, and complex tracking methods based upon the Kalman filter.

1.3.1 Kalman Filter Overview

Modeling Assumptions

The Kalman filter (KF) makes a number of fundamental assumptions regarding the system and its inputs. These are:

- The system state x_k evolves according to a known linear plant equation (dynamics) driven by a known input u_k and an additive process noise w_k. This noise is a zero-mean white (uncorrelated) process with known covariance Q_k.
- The measurement z_k is a known linear function of the state with an additive measurement noise v_k. This noise is a zero-mean white process with known covariance R_k.
- The initial state if unknown is assumed to be a random variable with known mean (the initial estimate x_0) and covariance (the initial uncertainty P_0).
- The initial error, the process noise, and the measurement noise are mutually uncorrelated.

$$\begin{aligned} x_{k+1} &= F_k x_k + G_k u_k + w_k \\ z_k &= H_k x_k + v_k \end{aligned} \qquad (1.3.1\text{-}1)$$

Hence, a state space formulation for the system is, with variables described above:

Also, F_k is the state transition matrix, G_k is the input matrix, H_k is the measurement matrix, and k is the sample index.

Algorithm Flowchart

At every stage k the entire past of the state trajectory is summarized by the state estimate $\hat{x}_{k|k}$ and its associated covariance $P_{k|k}$. The state estimation cycle consists of:

1. State and measurement prediction (also called the "time update"),
2. State update (also called the "measurement update").

The input, which is known and may be due to platform motion or sensor pointing, is used during state prediction. The state update requires the filter gain W_k, which is obtained from the covariance calculations.

By assuming that the initial state error and all the noises have Gaussian distributions, the KF becomes the minimum mean square error (MMSE) estimator. If the random variables are not Gaussian, and only their first and second moments are available, the KF is the best linear MMSE estimator. Hence, the KF can be said to be optimal in the sense of minimum mean square error.

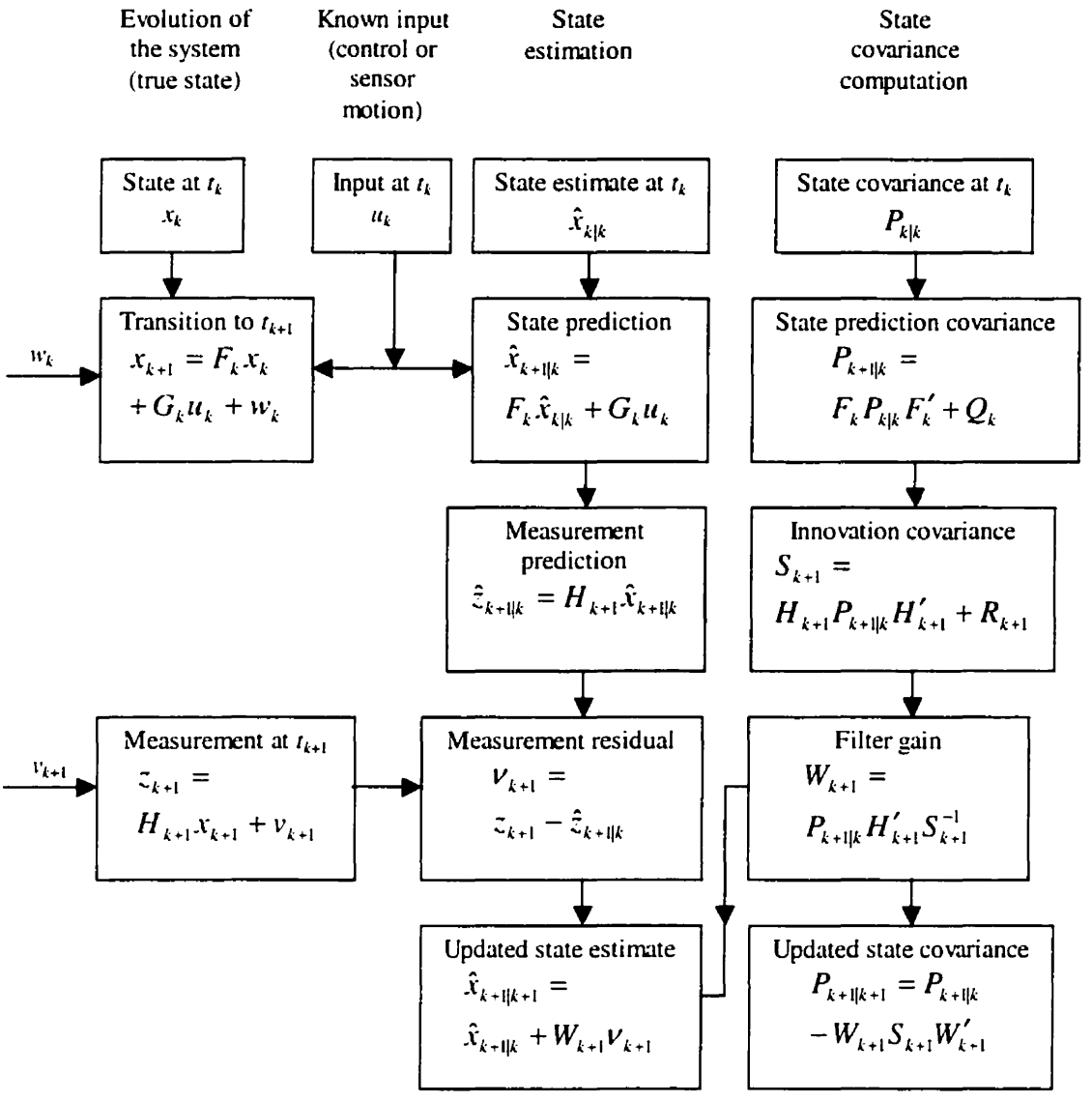

Figure 1.3-1 One cycle of KF

The assumptions made above limit the applicability of the basic Kalman filter to a number of applications, such as:

- Non-linear motion models or measurement equations – these can be resolved through linearization in the extended Kalman filter; a coordinate conversion, such as polar to Cartesian, may also resolve difficulties from non-linear dynamics,

- Unknown inputs and/or changes in the motion model, referred to as mode changes,
- Auto- or cross-correlated noise,
- Uncertainty as to the origin of measurements, in terms of data association not inaccuracy,
- Multiple targets.

More complex filtering methods, most incorporating the KF as their base, have evolved to resolve these difficulties. A discussion of the conditions for stability of the Kalman Filter may be found in [11].

1.3.2 Extended Kalman Filter

A sub-optimal method devised to resolve the problem of non-linear motion models or measurement equations is the extended Kalman filter (EKF).

Modeling Assumptions

The assumptions for the EKF are the same as for the KF except that, while the noises are still assumed to enter additively, the dynamic and/or measurement equations are now non-linear functions. The system equations can be inferred from the flowchart of the EKF, which follows.

Algorithm Flowchart

The EKF uses the Taylor expansion of the non-linear functions in the prediction stage. The main difference from the KF is the evaluation of Jacobians for the state transition and measurement equations; therefore, the covariance computations are no

longer decoupled from the state estimation calculations. Linearization occurs at the latest estimate, or along a nominal trajectory (which decouples the covariance computations).

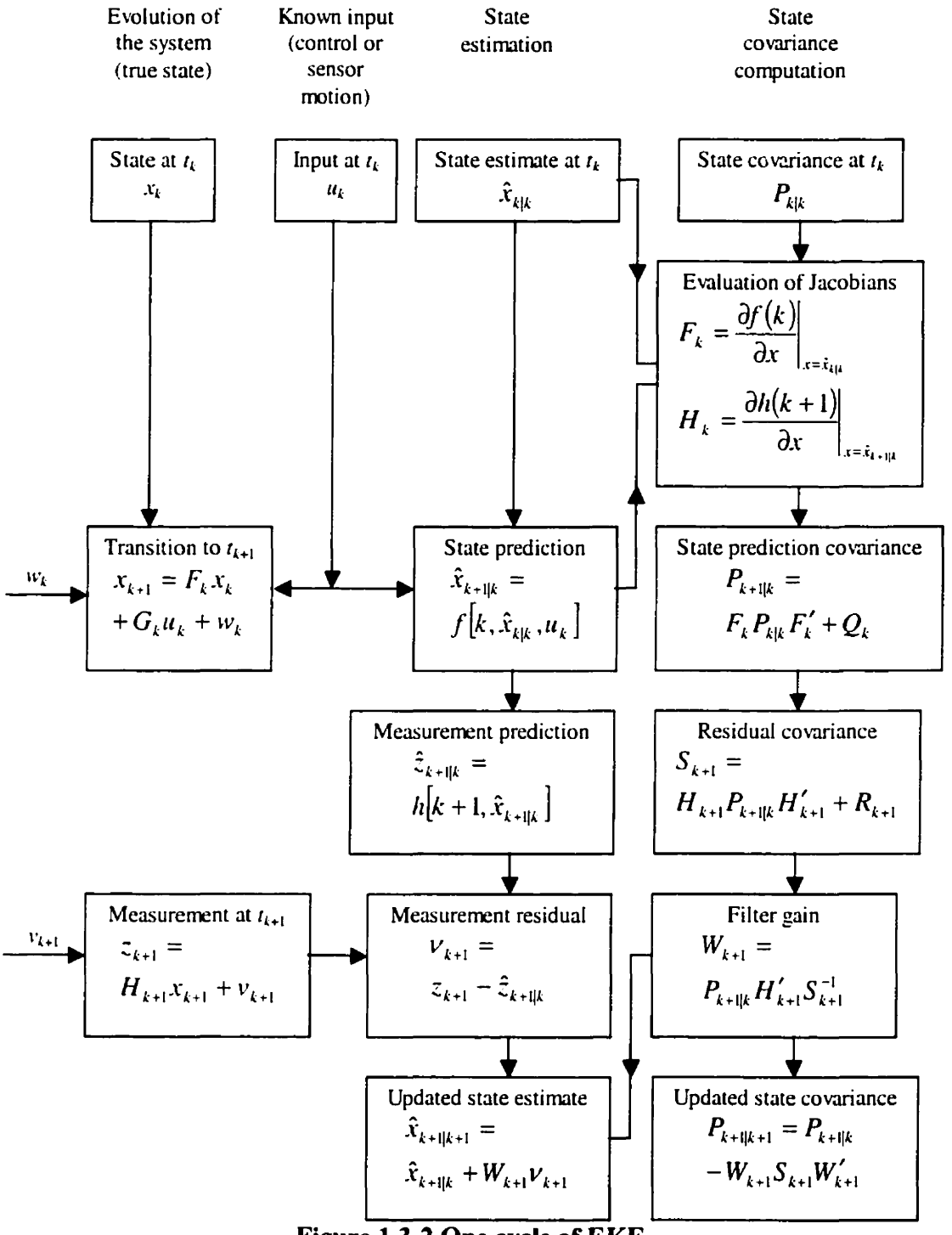

Figure 1.3-2 One cycle of EKF

The series expansion may introduce un-modeled errors that violate some assumptions regarding the prediction errors. The non-linear transformation will introduce a bias (a constant error) and the covariance calculation based on the series expansion is not always accurate. Furthermore, since the expansion is of first order, higher order terms are neglected. Finally, the EKF is very sensitive to the accuracy of initial conditions. A simple, though incomplete, way of compensating for un-modeled errors is to introduce artificial noise by increasing the process and/or measurement noise covariance, or to directly increase the filter-calculated state covariance.

In practice, if the initial errors and noises are small (small being a relative term to the state and measurement) then the EKF performs well.

1.4 THE MULTIPLE MODEL APPROACH

1.4.1 Overview

The multiple model approach is well suited for estimation in systems whose behavior changes with time. One typical such system, relevant in this project, is a maneuvering aircraft.

A multiple model system is assumed to obey, at any time, one of a finite number r of motion models. Hence, this system contains both continuous, from noise, uncertainties and discrete, pertaining to the different models or *modes*, uncertainties. A Bayesian approach is adopted in that the posterior mode probabilities (the probabilities that each model was in effect at the current time step given the measurements obtained) are obtained using their prior values (the probabilities that each model was in effect at the

prior time step). As the system evolves the filter switches between modes according to a Markov chain in order to track the target of interest.

An optimal multiple model estimator carries all the mode sequence hypotheses (mode combinations, such as: mode 1 at time k, mode 3 at time k+1, etc.). Clearly, such an approach entails maintaining r^k possibilities at time k. The complexity of such an algorithm then increases exponentially with time; that is, it is of complexity $O(r^k)$.

1.4.2 The Interacting Multiple Model Estimator

The interacting multiple model (IMM) estimator is the best-known und most widely used of the multiple model approaches. This method mixes track hypotheses with unit depth at the start of each cycle; therefore, it provides superior tracking performance, compared to the Kalman or Extended Kalman Filter, and reduced computational complexity/cost, compared to the optimum multiple model estimator or multi-hypothesis tracking [3]. "Mixing track hypotheses with unit depth" means that a weighted combination of the one step mode sequence hypotheses provides the tracking result. More likely hypotheses are more heavily weighted, thereby producing an algorithm of complexity $O(r)$.

Modeling Assumptions

The state model is:

$$\begin{aligned} x_k &= F[M(k)]x_{k-1} + w[k-1, M(k)] \\ z_k &= H[M(k)]x_k + v[k, M(k)] \end{aligned} \qquad (1.4.2\text{-}1)$$

Where $M(k)$ denotes the mode at time k; that is, the mode in effect during the sampling period ending at k.

There are finite number of modes r, such that:

$$M(k) \in \{M_j\}_{j=1}^{r} \tag{1.4.2-2}$$

The structure of the system and/or the statistics of the noise, which is assumed to be gaussian, can differ from mode to mode:

$$\begin{aligned} F[M_j] &= F_j \\ w(k-1, M_j) &\sim N(u_j, Q_j) \end{aligned} \tag{1.4.2-3}$$

The mode jump process is a Markov chain (the probability of a random process at time k given all prior values of the process up to and including the value at the k-1th time is the probability of that random variable at time k given only the k-1th time) with known mode transition probabilities.

$$P\{M(k) = M_j \mid M(k-1) = M_i\} = p_{ij} \tag{1.4.2-4}$$

Algorithm Flowchart

The algorithm contains four components:

- Interaction/mixing: Mixing of the previous cycle mode-conditioned state estimates and covariance, using the mixing probabilities, to initialize the current cycle of each mode-conditioned filter,

- Mode-conditioned filtering: Calculation of the state estimates and covariances conditioned on a mode being in effect, as well as of the mode likelihood function; hence, there are r parallel filters, usually KFs or some variation thereof,
- Probability evaluation: Computation of the mixing and the updated mode probabilities,
- Overall state estimation and covariance: Combination of the latest mode-conditioned state estimates and covariances to produce a single output; this result is not used in the next cycle of the algorithm and is for output only.

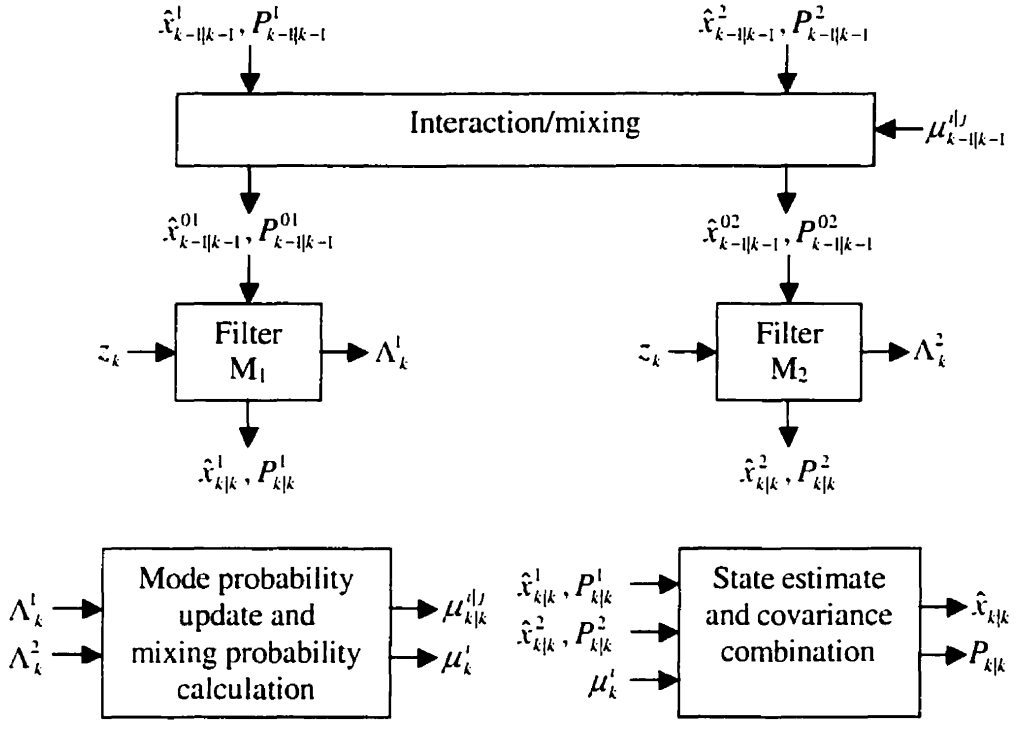

Figure 1.4-1 One cycle of the IMM ($i,j = 1,\ldots,r$)

Steps in IMM Estimation

1. Mixing probability calculation: These are the probabilities that mode M_i was in effect at time k-1 given that M_j is in effect at time k conditioned on the set of

measurements Z_{k-1}. These are calculated, using the assumptions, for $i,j = 1,\ldots,r$, as:

$$\mu^{i|j}_{k-1|k-1} = P\{M_{k-1} = M_i \mid M_k = M_j, Z_{k-1}\}$$
$$= \frac{1}{\bar{c}_j} P\{M_k = M_j \mid M_{k-1} = M_i\} P\{M_{k-1} = M_i \mid Z_{k-1}\}$$
$$= \frac{1}{\bar{c}_j} p_{ij} \mu^i_{k-1} \quad \text{where} \quad \bar{c}_j = \sum_{i=1}^{r} p_{ij} \mu^i_{k-1} \tag{1.4.2-5}$$

2. Interaction/Mixing: Starting with the previous state estimates $\hat{x}^i_{k-1|k-1}$ obtained from the r different modes (the filter blocks in the above figure) and the corresponding covariance matrices $P^i_{k-1|k-1}$, a mixed initial condition for the filter M_j at time k is calculated, for $j = 1,\ldots,r$, as:

$$\hat{x}^{0j}_{k-1|k-1} = \sum_{i=1}^{r} \hat{x}^i_{k-1|k-1} \mu^{i|j}_{k-1|k-1} \tag{1.4.2-6}$$

$$P^{0j}_{k-1|k-1} = \sum_{i=1}^{r} \mu^{i|j}_{k-1|k-1} \left\{ P^i_{k-1|k-1} + \left[\hat{x}^i_{k-1|k-1} - \hat{x}^{0j}_{k-1|k-1}\right]\left[\hat{x}^i_{k-1|k-1} - \hat{x}^{0j}_{k-1|k-1}\right]^T \right\} \tag{1.4.2-7}$$

3. Mode-conditioned filtering: The mixed initial condition obtained in step 2 as well as the new measurement z_k are now used as inputs to each filter M_j at time k, producing new modal estimates $\hat{x}^j_{k|k}$ and $P^j_{k|k}$. The filters also produce the modal measurement prediction $\hat{z}^j_{k|k-1}$ and the corresponding innovation covariance S^j_k. The conditional probability densities for the r filters are, as mentioned above, assumed to be gaussian and are denoted, for $j = 1,\ldots,r$, as:

$$\Lambda^j_k = N\left(z_k; \hat{z}^j_{k|k-1}, S^j_k\right)$$
$$= \frac{1}{(2\pi)^{n/2}\left|S^j_k\right|^{1/2}} \exp\left\{-\tfrac{1}{2}\left[z_k - \hat{z}^j_{k|k-1}\right]^T \left(S^j_k\right)^{-1}\left[z_k - \hat{z}^j_{k|k-1}\right]\right\} \tag{1.4.2-8}$$

4. Mode probability update: The new, post filtering, mode probabilities, for j = 1,...r, are:

$$\mu_k^j = P\{M_k = M_j \mid Z_k\}$$
$$= \frac{1}{c}\Lambda_k^j \sum_{i=1}^{r} P\{M_k = M_j \mid M_{k-1} = M_i, Z_{k-1}\} P\{M_{k-1} = M_i \mid Z_{k-1}\}$$
$$= \frac{1}{c}\Lambda_k^j \sum_{i=1}^{r} p_{ij}\mu_{k-1}^i = \frac{1}{c}\Lambda_k^j \bar{c}_j \quad \text{where} \quad c = \sum_{j=1}^{r} \Lambda_k^j \bar{c}_j \qquad (1.4.2\text{-}9)$$

5. State estimate and covariance combination: The next point along an established track, for output purposes, is generated from:

$$\hat{x}_{k|k} = \sum_{j=1}^{r} \hat{x}_{k|k}^j \mu_k^j \qquad (1.4.2\text{-}10)$$

$$P_{k|k} = \sum_{j=1}^{r} \mu_k^j \left\{ P_{k|k}^j + \left[\hat{x}_{k|k}^j - \hat{x}_{k|k}\right]\left[\hat{x}_{k|k}^j - \hat{x}_{k|k}\right]^T \right\} \qquad (1.4.2\text{-}11)$$

A number of parameters in the IMM are left to the user to set, these are:

- The types of filters used – such as, linear (KF) or non-linear (EKF),
- The values of the process noise variances,
- The values of the mode transition probabilities p_{ij}.

Finally, for a 3 linear model IMM the computational requirements are approximately 4 times higher than for the KF.

1.5 REFERENCES

[1] Y. Bar-Shalom and X.R. Li. *Multitarget-Multisensor Tracking: Principles and Techniques*. Storrs, CT, YBS Publishing, 1995.

[2] Y. Bar-Shalom and X.R. Li. *Estimation and Tracking: Principles, Techniques, and Software*. Boston, MA, Artech House, 1993.

[3] S.S. Blackman. *Multi-Target Tracking with Radar Application*. Artech House, 1986.

[4] R.W. Sittler. An Optimal Data Association Problem in Surveillance Theory. *IEEE Trans. Mil. Electron.*, MIL-8:125-139, Apr. 1964.

[5] Y. Bar-Shalom and T.E. Fortmann. *Tracking and Data Association*. Academic Press, San Diego, 1988.

[6] R.E. Helmick and G.A. Watson. IMM-IPDAF for Track Formation on Maneuvering Targets in Cluttered Environments, *SPIE*, Vol. 2234, pp. 460-471, 1994.

[7] H. Leung et al. Evaluation of Multiple Radar Target Trackers in Stressful Environments, *IEEE Trans. Aero. Elec. Sys.*, Vol. 35, No. 2, pp. 663-673, 1999.

[8] S.S. Blackman and M. Busch. Evaluation of IMM Filtering for an Air Defense System Application, *SPIE*, Vol. 2561, pp. 435-445, 1995.

[9] G.A. Watson and W.D. Blair, Tracking Maneuvering Targets with Multiple Sensors Using the Interacting Multiple Model Algorithm, *Proc. Signal and Data Processing of Small Targets*, 1954, pp. 438-449, 1993.

[10] G.A. Watson, IMAM Algorithm for Tracking Maneuvering Targets in Clutter, *Proc. Signal and Data Processing of Small Targets*, Vo. 2759, pp. 304-315, 1996.

[11] P.E. Caines, *Linear Stochastic Systems*, J. Wiley, NYC, 1988.

2 THE IMM-JVC ALGORITHM

2.1 INTRODUCTION

The tracking accuracy provided by the Interacting Multiple Model state estimator (IMM), [1], [2], has long been recognized as superior to that obtained from other single-scan positional estimators such as the adaptive Kalman filter or rule-based maneuver detectors when operating in environments with maneuvering targets. Although it employs certain simplifying assumptions, the IMM is the least computationally demanding algorithm of the multiple model filter family. It also allows for the incorporation of non-linear motion models (through the EKF) such as coordinated turns. It has thus proven to be more reliable than an adaptive version of the Kalman filter in detecting maneuvering periods; for comparisons, see [3] and [4].

However, when dealing with closely maneuvering targets, the most critical role is played by the chosen method of data association. Without effective association, state estimation is at risk. Hence, for the purposes of this paper, an IMM state estimation algorithm was combined with data association based upon the Jonker-Volgenant-Castanon (JVC) optimization procedure. This IMM-JVC algorithm was developed, implemented, and tested within an aircraft surveillance program. The combined algorithm may take a favorable place among other approaches, such as the IMM-JPDAF, [6], or the multi-hypothesis tracking (MHT) [7].

While constructing the IMM-JVC algorithm, a special emphasis was put on tackling the important problem of asynchronous sensors, which are likely to deliver measurements in an out-of-sequence fashion, [8]. This situation is frequently encountered in the case of multiple radars working at various scanning rates and sending scans (see chapter one for a definition of scan) of data at a pace depending on target density. Since the data association sub-algorithm and the modes of the IMM filter employ the most recent measurements to update the tracks, such "out-of-sequence" measurements, if neglected, can result in: (a) erroneous state and covariance estimates, (b) improper measurement to track association, or (c) a complete loss of track. To increase the reliability of the IMM-JVC algorithm, an approach allowing for the inclusion of such measurements in the state estimation and data association modules is suggested. This approach is similar to that found in [9] but results in a different covariance matrix update.

2.2 JVC ASSOCIATION

The Jonker-Volgenant-Castanon (JVC) algorithm is a global nearest neighbor data association algorithm. It yields a unique pairing of measurements and tracks. While a nearest neighbor approach would, for every track or measurement, associate each track (measurement) with the nearest measurement (track), a global nearest neighbor approach seeks to minimize the total distance, or a function thereof, between tracks and measurements.

In terms of the notation put forward in presenting the KF, let $\hat{z}_{k+1|k}$ denote the predicted mean of the measurement z_{k+1} conditioned on the k past measurements $Z_k = \{z_0, \ldots, z_k\}$; that is, $\hat{z}_{k+1|k} = H_{k+1}^T \hat{x}_{k+1|k}$ with $\hat{x}_{k+1|k} = E[x_{k+1} \mid Z_k]$. Let S_{k+1} denote the

associated covariance matrix, that is, $S_{k+1} = H_{k+1}^T P_{k+1|k} H_{k+1} + R_{k+1}$ where $P_{k+1|k} = E\left[\left(x_{k+1} - \hat{x}_{k+1|k}\right)\left(x_{k+1} - \hat{x}_{k+1|k}\right)^T \mid Z_k\right]$.

Let i denote variables belonging to the i^{th} track and j denote variables belonging to the j^{th} measurement. Under the assumption that the probability distribution of the j^{th} measurement conditioned on the past Z_k^i is gaussian with mean $\hat{z}_{k+1|k}^i$ and covariance S_{k+1}^i (these values are obtained using the KF or equivalent), the normalized innovation squared $d_{ij}^2 = \left(\nu_{k+1}^{ij}\right)^T \left(S_{k+1}^i\right)^{-1} \nu_{k+1}^{ij}$, with $\nu_{k+1}^{ij} = z_{k+1}^j - \hat{z}_{k+1|k}^i$, is chi-square distributed with number of degrees of freedom equal to the dimension of the measurement. With n denoting the number of tracks and measurements (the algorithm easily generalizes to the case of fewer tracks than measurements), the JVC module begins with the evaluation of the weights c_{ij}, $i,j = 1,\ldots,n$, reflecting the probabilistic distances in all possible pairings of the tracks and the measurements, such that:

$$c_{ij} = P\left\{\chi^2 > d_{ij}^2\right\} \qquad (2.2\text{-}1)$$

The weights reflect the probability that the system states do <u>not</u> fall within the observation gates (regions) delimited by the normalized innovations; hence, the weights are probabilities of non-association. These weights may be placed within an assignment matrix, allowing the use of a minimizing optimization algorithm as described below. As described in [11], the JVC algorithm comprises of two sequential steps, the first being similar to an auction algorithm [12], and the second being a modified version of the Munkres algorithm [13] for sparse matrices.

A unique one-to-one track to measurement pairing is then derived as the solution $\hat{x}_{ij}$ to the following linear program:

$$\min\left\{\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}\right\} \tag{2.2-2}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \quad \sum_{j=1}^{n} x_{ij} = 1 \tag{2.2-3}$$

$$0 \le x_{ij} \le 1 \quad \forall i, j \tag{2.2-4}$$

Clearly, $\hat{x}_{ij}$ is integer valued, with unity signifying association of a measurement to a track.

The following procedure is used to resolve the matter of track formation. Since every measurement is considered to be a possible track in this procedure, measurements are referred to as *contacts*. When the number of contacts in the buffer is larger than the number of tracks to be updated, the contacts left unassigned will contribute to the creation of a new track with a status left as NO-STATUS. If, at a later time, a second contact is assigned to a NO-STATUS track, then the track, then the track is promoted as INITIATED. If a third contact is later assigned to an INITIATED track, the track is promoted as TENTATIVE. It is only at the fourth assigned contact that the status of a track is declared as FIRM.

When building the JVC assignment matrix, two processes should be considered since they may alter the performance of the JVC optimization. The first of these two processes is the buffering scheme chosen to present the sensor reports to the association module (the JVC module) of the MSDF test-bed. Most of the time, the characteristics of these buffers (time duration, scanned volume) are intimately related to built-in rules characterizing each sensor. Those rules are based on factors such as scan duration,

scanned angular range, estimated target density, etc. Buffers of sensor reports built according to those rules are very often used without modification in the contact-to-track association module. Inadequate buffering could seriously degrade tracking performance in a dense target environment, especially one involving multiple sensors; scenario 3 below discusses this further.

The second of these two processes is the selection from the MSDF track database of those tracks that are likely to be updated with a subset of the incoming contacts. For MSDF systems having real-time constraints, specific track selection strategies may be required to reduce the processing time. For instance, there may not be a need to include in the assignment matrix a track that is distantly removed from the buffered contacts when nearby tracks are quite certainly the only possibilities for matching. Filtering out these impossible possibilities would reduce the dimension of the assignment matrix, thereby reducing the computing time for association without risking accuracy.

2.3 THE IMM MODULE

The IMM is described above in section 1.4.2. The only modifications made to the IMM module are to integrate it with the JVC module.

2.4 INTEGRATION OF THE IMM AND JVC

As explained above, step 3 of the IMM algorithm employs the measurement z_k. It is at this point that data association should be incorporated into tracking. Since the state estimation filter is the IMM, there are, at this step in the estimation process, r different modal estimates. Rather than matching modal estimates with measurements, a simpler

and faster approach is to obtain a one-step prediction output value from the IMM, analogous to the updated output in step 5.

The KF/EKF prediction equations, as in the section 1.3.1 or section 1.3.2 flowchart, for the modes $j = 1,\ldots,r$, are:

$$\hat{x}^{0j}_{k|k-1} = F^j_k \hat{x}^{0j}_{k-1|k-1} \tag{2.4-1}$$

$$P^{0j}_{k|k-1} = F^j_k P^{0j}_{k-1|k-1} \left(F^j_k\right)^T + Q^j_k \tag{2.4-2}$$

The weighting for an IMM one-step predictor can be found by introducing the prediction probabilities $\mu^j_{k|k-1} = P\{M_k = M_j \mid Z_{k-1}\}$, which are calculated as:

$$\mu^j_{k|k-1} = \sum_{i=1}^{r} P\{M_k = M_j \mid M_{k-1} = M_i, Z_{k-1}\} P\{M_{k-1} = M_i \mid Z_{k-1}\}$$

$$= \sum_{i=1}^{r} p_{ij} \mu^j_{k-1} \tag{2.4-3}$$

The individual modal estimates 2.4-1,2 are now mixed, as a gaussian mixture [2, §1.4.16], using the prediction probabilities 2.4-3 to produce the IMM one-step predicted state and covariance, which identify the tracks for the purpose of measurement to track association.

$$\hat{x}_{k|k-1} = \sum_{i=1}^{r} \hat{x}^{0j}_{k|k-1} \mu^j_{k|k-1} \tag{2.4-4}$$

$$P_{k|k-1} = \sum_{i=1}^{r} \mu^j_{k|k-1} \left\{ P^{0j}_{k|k-1} + \left[\hat{x}^{0j}_{k|k-1} - \hat{x}_{k|k-1}\right]\left[\hat{x}^{0j}_{k|k-1} - \hat{x}_{k|k-1}\right]^T \right\} \tag{2.4-5}$$

These results are readily used as inputs to the JVC module to produce the innovations and the corresponding probabilistic distances, finally yielding the desired measurement to track associations. The execution of step 3 of the IMM module is thus enabled, and the two modules, JVC and IMM, are then integrated into one algorithm.

2.5 THE "OUT-OF-SEQUENCE" MEASUREMENTS

When the measurements from a bank of sensors arrive to a central processor in batches (such as, corresponding to radar scans) it is a frequent situation that an earlier measurement arrives after a later one. This calls for a "corrective" procedure whose purpose would be to incorporate the earlier measurement (arriving out-of-sequence) in the current state estimate and the corresponding covariance. In this context, a possibility is presented below.

The Retrodiction Approach

Suppose that a measurement z_{k+1} arrives at time t_{k+1} such that $t_{k+1} - t_k < 0$; that is, z_{k+1} is an out-of-sequence measurement. Begin by regarding all the previous measurements Z_k as no longer available, that is, the previous measurements were not stored in memory. This assumption is usually the case since recursive algorithms such as the KF or IMM do not require the measurement history to function, thereby producing a saving in the amount of computer memory required. The retrodiction approach incorporates the measurement z_{k+1} when it arrives to produce a "corrected" state estimate and covariance at time t_k:

$$\hat{x}_{k|k+1} = E[x_k \mid Z_k, z_{k+1}], \text{ and } P_{k|k+1} = E\left[(x_k - \hat{x}_{k|k+1})(x_k - \hat{x}_{k|k+1})^T \mid Z_k, z_{k+1}\right].$$

For this to be possible, a simplifying assumption is necessary which states that the process noise at time t_{k+1} is insignificant and thus can be regarded as being equal to zero. (Otherwise, terms such as $E[w_{k+1} \mid Z_k] \neq 0$ will appear in the calculations, which are virtually impossible to account for.) The calculation of the corrected state and covariance estimates is then straightforward and proceeds as follows.

The expectations and covariances of the random variables x_k and z_{k+1} conditioned on the previous measurements Z_k are either available as estimates at time t_k, or else easily calculated as:

$$\begin{aligned}
\hat{x}_{k|k} &= E[x_k \mid Z_k] \quad \text{- avail.} \\
\hat{z}_{k+1|k} &= E[z_{k+1} \mid Z_k] = H_{k+1}^T F_{k+1}^{-1} \hat{x}_{k|k} \\
\Sigma_{xx} &= E\left[(x_k - \hat{x}_{k|k})(x_k - \hat{x}_{k|k})^T \mid Z_k\right] = P_{k|k} \quad \text{- avail.} \\
\Sigma_{xz} &= E\left[(x_k - \hat{x}_{k|k})(z_{k+1} - \hat{z}_{k+1|k})^T \mid Z_k\right] \\
&= P_{k|k}\left(F_{k+1}^{-1}\right)^T H_{k+1} \\
\Sigma_{zz} &= E\left[(z_{k+1} - \hat{z}_{k+1|k})(z_{k+1} - \hat{z}_{k+1|k})^T \mid Z_k\right] \\
&= H_{k+1}^T F_{k+1}^{-1} P_{k|k}\left(F_{k+1}^{-1}\right)^T H_{k+1} + R_{k+1}
\end{aligned} \tag{2.5-1}$$

F_{k+1}^{-1} is the system matrix providing for the state transition backwards in time from x_{k+1} to x_k. In terms of the above, the desired estimates are:

$$\begin{aligned}
\hat{x}_{k|k+1} &= \hat{x}_{k|k} + \Sigma_{xz}\Sigma_{zz}^{-1}\left(z_{k+1} - \hat{z}_{k+1|k}\right) \\
P_{k|k+1} &= P_{k|k} - \Sigma_{xz}\Sigma_{zz}^{-1}\Sigma_{xz}^T
\end{aligned} \tag{2.5-2}$$

For the purposes of data association in the JVC module:

$$\begin{aligned}
\hat{x}_{k+1|k} &= F_{k+1}^{-1}\hat{x}_{k|k} \\
P_{k+1|k} &= F_{k+1}^{-1} P_{k|k}\left(F_{k+1}^{-1}\right)^T
\end{aligned} \tag{2.5-3}$$

These expressions are readily used in the next cycle of the IMM-JVC algorithm.

2.6 SIMULATION EXPERIMENTS

The following experimental results were obtained on the agent-based Multi-Sensor Data Fusion Testbed MSDF developed at Lockheed Martin Canada (Montreal) LMC using a scenario simulator developed by the Canadian Defense Research Establishment Agency at Valcartier (Quebec) DREV. The algorithm was implemented in the agent-based MSDF and that implementation is described below in detail.

2.6.1 Scenario 1 – Aircraft Closely Maneuvering

This scenario shows two aircraft flying along straight paths at 100m/s along a 45° converging path scanned by radar at a rate of 30rpm. When the distance between the two aircraft is of 200m they both perform a 90° turn with an acceleration of 3g during 5s. This trajectory can be seen in Figure 2.6-2.

Simulation shows that when using a nearest neighbor algorithm for association, the aircraft maneuvers were not detected regardless of the positional estimator algorithm used (adaptive Kalman or IMM using a constant velocity and a constant acceleration motion models). For the data fusion module, each aircraft pursues an incorrect rectilinear (straight line) course. Figure 2.6-1 shows the result obtained when using the nearest neighbor association algorithm with the IMM for positional estimation. The same problem occurs when the JVC algorithm is used for association and combined with an adaptive KF for positional estimation.

As shown in Figure 2.6-2, the two aircraft were correctly tracked when using an IMM-JVC combination where the IMM motion models are one constant velocity CV and one constant acceleration CA (hence CVCA). The IMM transitional probabilities (the off-diagonal entries in the mode transition matrix; that is, p_{ij} where $i \neq j$) were set for 10% switching at 2s measurement intervals (2s being the length of a single radar scan).

One may note that the performance obtained in Figure 2.6-1 would also represent the expected performance when using the JVC association with buffers of contacts containing only one sensor report. In such a case, the JVC assignment matrix is of dimension $1 \times m$, where m is the number of tracks – selected from the track database –

which are likely to be updated with the incoming contact. Such a situation would occur when the measurement report from each aircraft is received for processing separately, as in the instance of a multi-sensor setup. This situation demonstrates the importance of proper buffering in a multi-sensor setup.

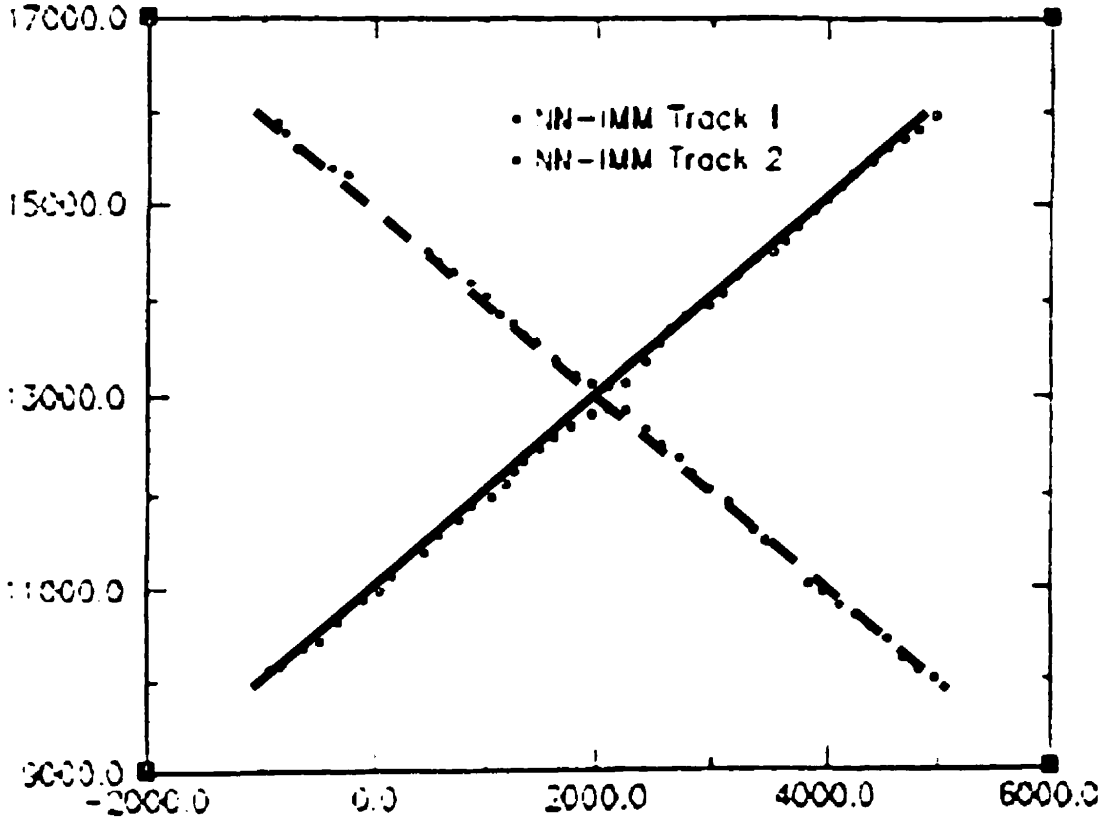

Figure 2.6-1 NN association, estimation with adaptive KF or IMM

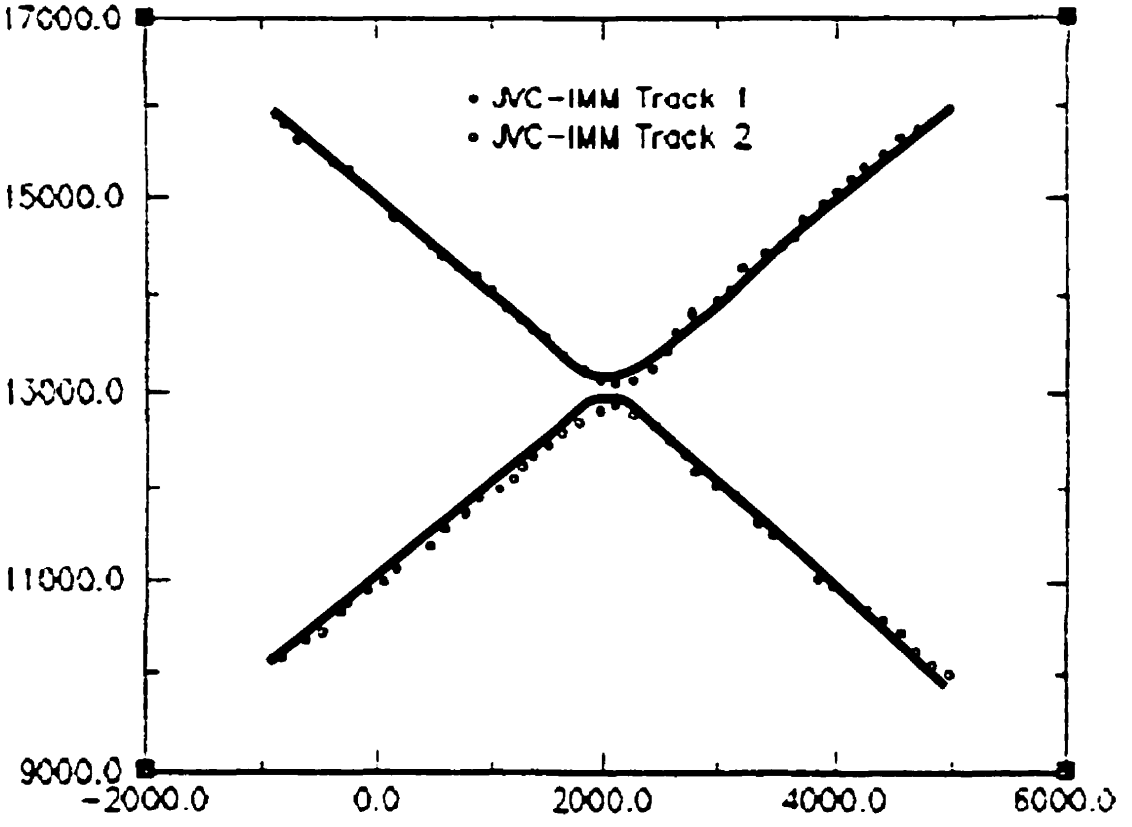

Figure 2.6-2 JVC association, estimation with IMM

2.6.2 Scenario 2 – Noise-free Retrodiction

This scenario is of one aircraft flying at 100m/s along a straight path. It is tracked using a long range radar operating at 12rpm, then enters the range of a short range radar operating at 60rpm, and finally exits the short range coverage after 750s. Out-of-sequence measurements were simulated by introducing a delay of 0.5s in the fusion of the slowest (long range) radar reports.

Figure 2.6-3 shows the covariance of the track in the y-coordinate. The degradation of the track shown by the covariance increase (the dotted curve) when the aircraft exits the short range is avoided when using the noise-free retrodiction algorithm (the dashed curve). This figure also shows that the curve corrected by the noise free

retrodiction is closely superimposed to that curve obtained when no time shift is introduced.

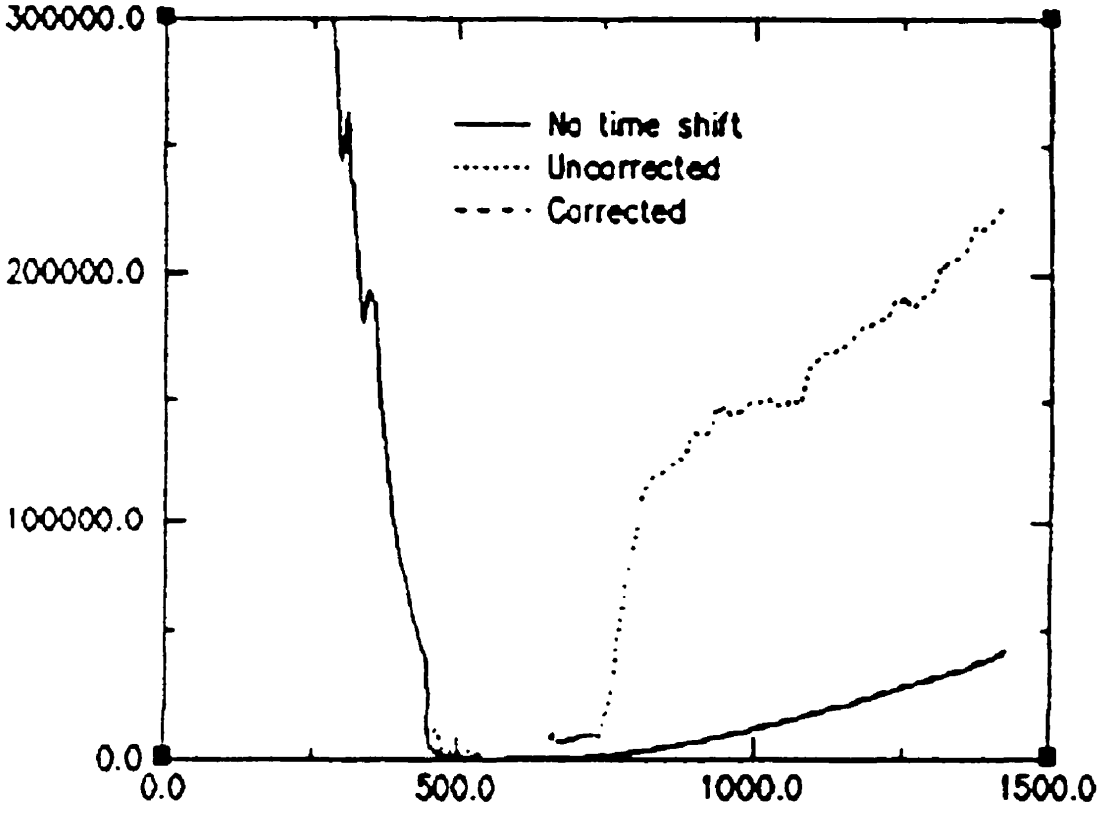

Figure 2.6-3 Track covariance in presence of out-of-sequence measurements

2.6.3 Scenario 3 – Aircraft Closely Maneuvering with Clutter

In order to demonstrate the capabilities of the JVC module in data association, the following scenario is presented. In this scenario, the two aircraft of scenario 1 are again flying along the same path. After the first 20s of the scenario, where each track is of FIRM status, random clutter is added within each buffer. For each clutter-free buffer of N_t target contacts, a random number N_c between 0 and 3 of clutter contacts is randomly generated. Then each of the N_c clutter reports is assigned a randomly generated time

within a radar scan and a randomly generated range-bearing measurement within the radar field of view. Figure 2.6-4 shows the resulting tracks with clutter.

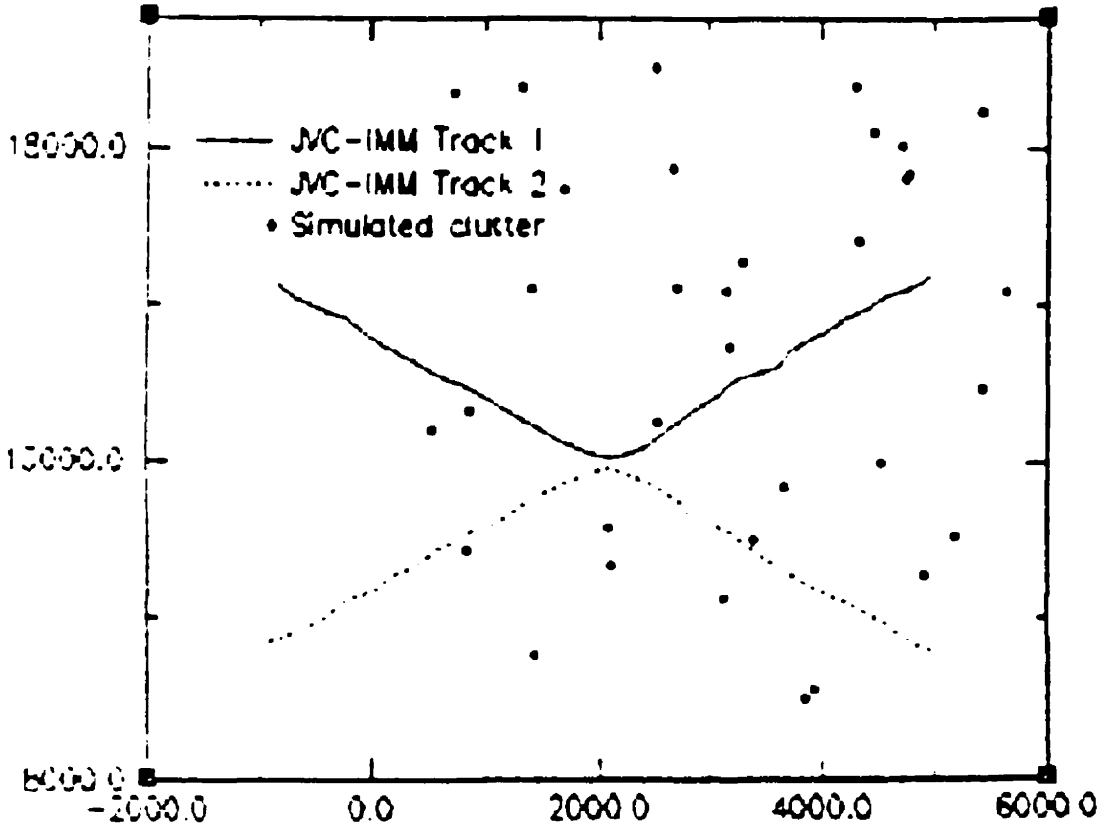

Figure 2.6-4 JVC association, IMM-CVCA with clutter

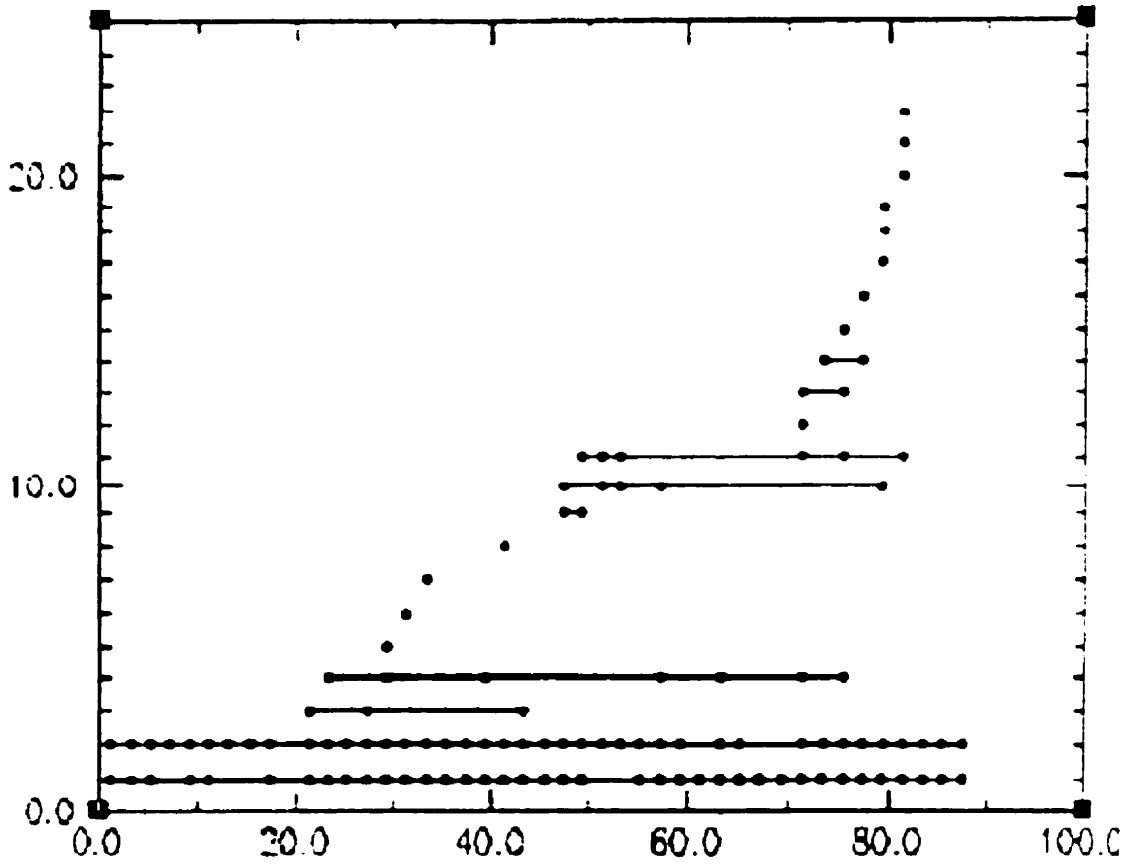

Figure 2.6-5 MSDF track creation and life duration

Figure 2.6-5 illustrates the life duration and status of the tracks that have been created in the track database. The line length (along the x-axis) represents the track duration while the track number in the database is given on the y-axis. When only one dot appears, the track has been created but has a NO-STATUS. When two, three, four, and more than four dots appear, the status of the track is respectively INITIATED, TENTATIVE, and FIRM. This figure clearly shows that among the 22 tracks that have been created, 13 did not rise above NO-STATUS, 3 stopped at INITIATED, 1 at TENTATIVE, and only 5 reached FIRM status. The *KillOldTrack* mechanism, which removes from the track database all the tracks that have not been recently updated, was deactivated for this simulation. At a scan rate of 30rpm, one might decide to remove

tracks from the database after 5s (after 2.5 full scans without update). With the mechanism active, the figure shows that no tracks other than those corresponding to the two aircraft would have achieved FIRM status.

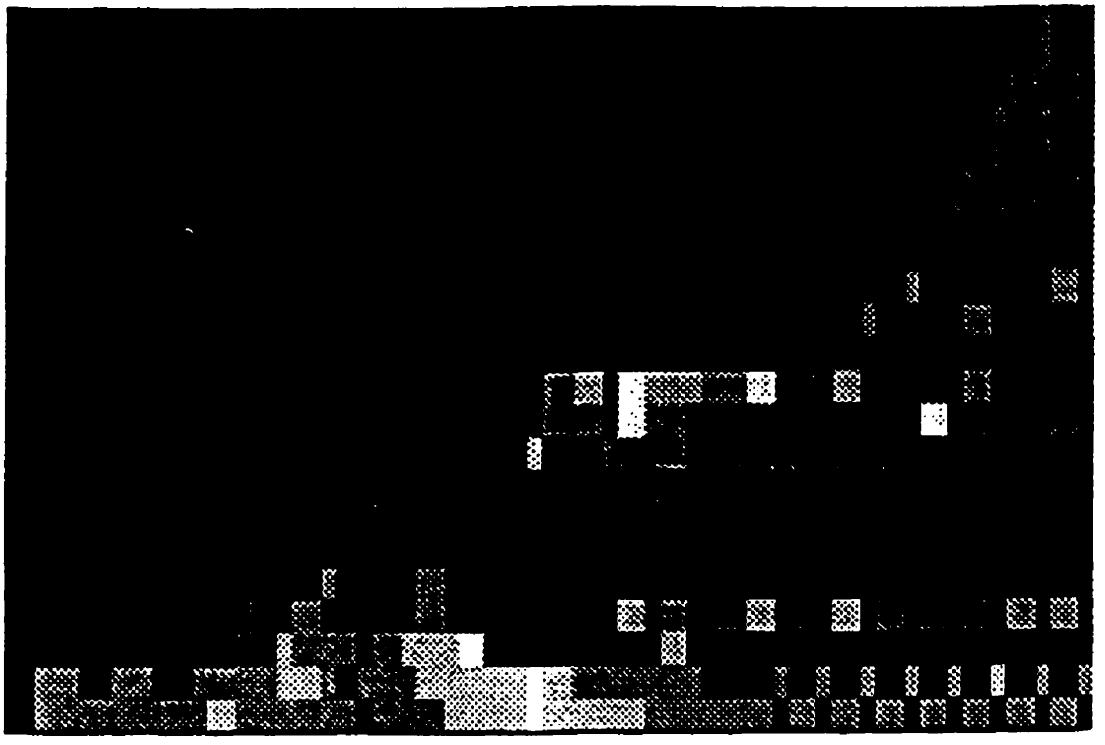

Figure 2.6-6 2D mapping of the number of contacts in track gates

Figure 2.6-6 shows a two-dimensional mapping of the number of contacts in track gates. Time is along the horizontal, and the 22 tracks along the vertical. The gray-level coding in this 2D mapping is interpreted as follows. All pixels in black can be ignored in that the corresponding track is not selected to build the assignment matrix. Then, 3 brightening levels of gray are used when the corresponding track has: an infeasible assignment (track is selected but the probability of non-association resulting from the calculation of the statistical distance is higher than a given threshold), one feasible assignment (one contact in the track gate), and two feasible assignments. White

represents the case of 3 contacts within a track gate. Note that the aircraft tracks (the bottom two) peak in intensity around the scenario's midpoint when they are closest to each other. This 2D mapping allows one to identify when JVC optimization is dissimilar to nearest-neighbor assignment.

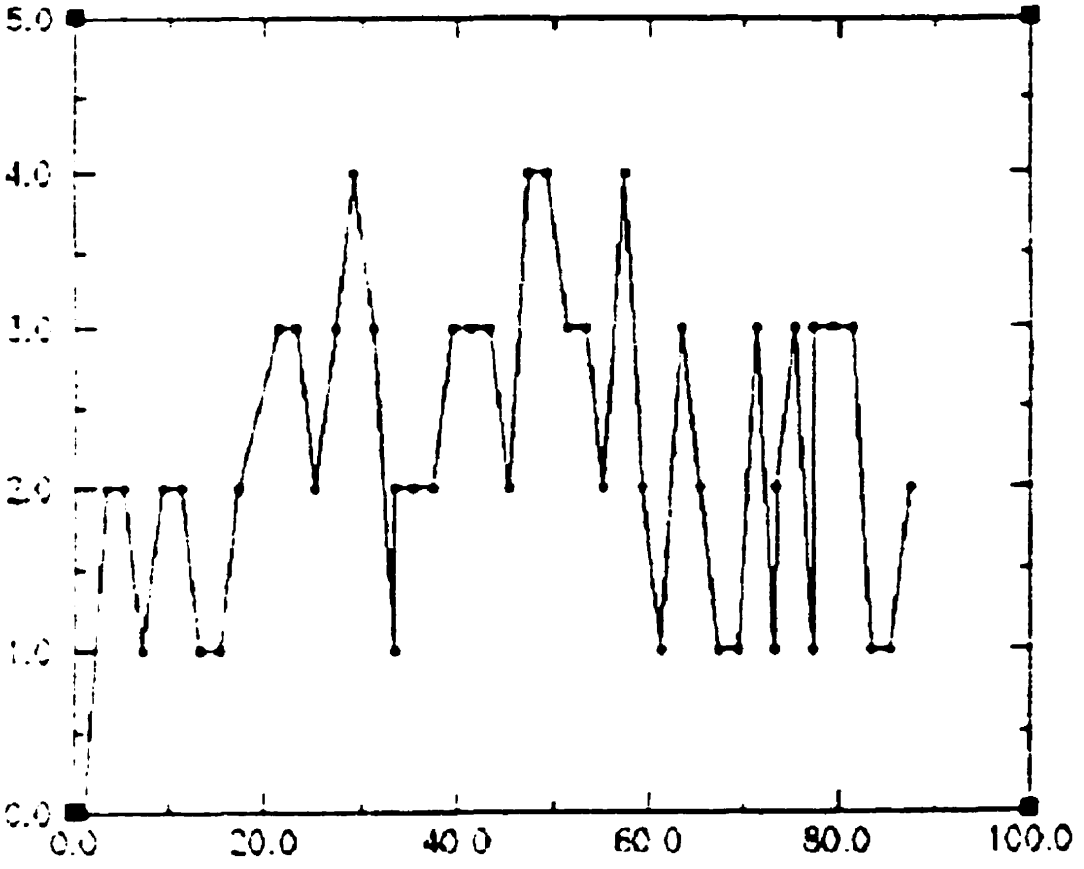

Figure 2.6-7 Number of contacts in buffers

Further information may be obtained by considering Figure 2.6-7. This figure displays the actual number of incoming contacts in the buffer. Comparison with Figure 2.6-6 allows an evaluation of the time at which track gates overlap. For example, at time 43.31s, Figure 2.6-7 shows that there are three contacts in the buffer (two from the aircraft, one from clutter) and the 2D mapping shows that 2 contacts fall in the gates of tracks 1 and 2, and 3 contacts fall in the gate of track 3. From Figure 2.6-5, one sees that

the aircraft contacts are correctly assigned and the clutter contact is assigned to track 3 with a status promotion from INITIATED to TENTATIVE. If the *KillOldTrack* mechanism had been active, track 3 would have been previously removed and the clutter contact would have triggered the creation of a new track.

This scenario highlights the critical role played by the association algorithm for tracking closely maneuvering targets. The global assignment problem, that JVC optimization is made to solve, yields an excellent tracking accuracy with a reasonable computer execution time. Track management functionalities such as the *KillOldTrack* mechanism also has a role to play to reduce the execution time since it will lead to assignment matrices smaller in size.

2.7 REFERENCES

[1] H.A.P. Blom and Y. Bar-Shalom. The interacting multiple model algorithm for systems with Markovian switching coefficients. *IEEE Transactions on Automatic Control*, Vol. 33 No. 8, 1988, pp. 780-783.

[2] Y. Bar-Shalom and X.R. Li. *Estimation and Tracking: Principles, Techniques, and Software*. Boston, MA, Artech House, 1993.

[3] K. Kastella and M. Biscuso. Tracking algorithms for air traffic control applications. *Air Traffic Control Quaterly*, Vol. 3, No. 1, 1996, pp. 19-43.

[4] A. Jouan, E. Bosse and al. Comparison of various schema of filter adaptivity for the tracking of maneuvering targets. *Proceedings of Aerosense '98: Signal and Data Processing of Small Targets*, Vol. 3373, 1998, pp. 247-258.

[5] R. Jonker, A. Volgenant. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, Vol. 38, 1987, pp. 325-340.

[6] M. de Feo, A. Graziano, R. Miglioli, A. Farina, IMMJPDA versus MHT and Kalman filter with NN correlation: performance comparison. *Proceedings of IEE on Radar, Sonar Navig.*, Vol. 144, No. 2, 1997, pp. 49-56.

[7] Y. Bar-Shalom and X.R. Li. *Multitarget-Multisensor Tracking: Principles and Techniques*. Storrs, CT, YBS Publishing, 1995.

[8] M. Yeddanapudi, Y. Bar-Shalom and K.R. Patipati. IMM estimation for multitarget multisensor air traffic surveillance. *Proceedings of the IEEE*, Vol. 85, No. 1, 1997, pp. 80-94.

[9] H. Wang, T. Kirubarajan and Y. Bar-Shalom. Precision large scale air traffic surveillance using IMM/assignment estimators. *IEEE Transactions on Aerospace and Electronic Syst.*, Vol. 35, No. 1, 1999, pp. 255-265.

[10] S.S. Blackman. *Multiple-Target Tracking with Radar Applications*. Artech House, 1986.

[11] O.E. Drummond, D.A. Castanon, M.S. Bellovin. Comparison of 2-D Assignment for Sparse, Rectangular, Floating Point, Cost Matrix. *Journal of the SDI Panels on Tracking*, Institute for Defense Analyses, Issue No. 4, 1990, pp. 4-81 to 4-97.

[12] D.P. Bertsekas. The Auction Algorithm: A Distributed Relaxation Method for the Assignment Problem. *Annals of Operations Research*, Vol. 14, 1988, pp. 103-123.

[13] J. Munkres. Algorithms for the Assignment and Transportation Problems. *J. Soc. Indust. Applied Math.*, Vol. 5, No. 1, 1957, pp. 32-38.

[14] B. Jarry, A. Jouan, H. Michalska. An IMM-JVC Algorithm for Multi-Target Tracking with Asynchronous Sensors. *Proceedings of the 19th IASTED Conference on Modelling, Information and Control*, Innsbruck, Austria, 2000, pp. 603-607.

[15] A. Jouan, B. Jarry, H. Michalska. Tracking Closely Maneuvering Targets in Clutter with an IMM-JVC Algorithm. *FUSION 2000*, in press.

3 DATA TYPES & AGENTS

3.1 THE DECISION AIDS FOR AIRBORNE SURVEILLANCE PROJECT (DAAS)

The Decision Aids for Airborne Surveillance DAAS research project aims at building a multi-sensor data fusion MSDF test-bed emulating the fusion of the imaging and non-imaging sensors onboard a surveillance aircraft. The DAAS is a project undertaken by Lockheed Martin Canada LMC. This test-bed is implemented upon a Knowledge-Based System KBS shell, in a very efficient, modular, agent-based architecture. The contribution of this paper to the DAAS project is focused upon the enhancement of the test-bed by introducing and improving upon existing algorithms for tracking, namely, the development of the IMM-JVC algorithm. This chapter describes the test-bed implementation of the IMM-JVC algorithm.

3.2 TESTBED ARCHITECTURE

The architecture developed by LMC in collaboration with DREV uses a Knowledge Based System KBS shell based on a BlackBoard-based BB problem-solving paradigm. This architecture is also shown, along with blocks for input, output, simulation, and scenario generation, in Figure 3.2-1. The STA (Situation & Threat Assessment) and RM (Resource Management) agents are outside the scope of this paper. The test bed includes:

- Infrastructure comprising simulation, performance evaluation, user interaction (HCI block), and a scenario generator,
- MSDF agents – tracking and data association algorithms, as well as identification algorithms.

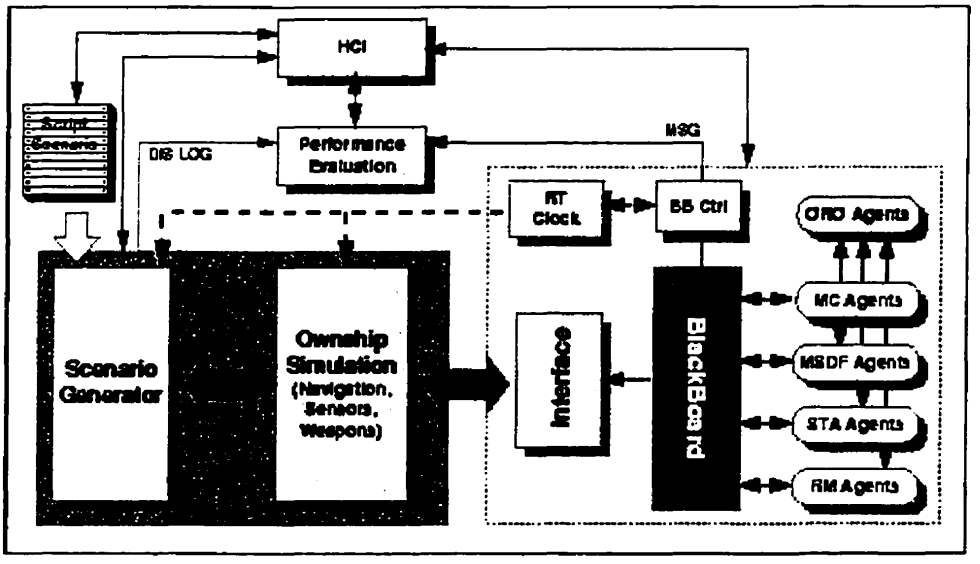

Figure 3.2-1 Test bed architecture

The KBS architecture presents a number of advantages, such as:

- A distributed (multiprocessor) environment,
- Design flexibility, allowing the integration of diversified knowledge sources and heterogeneous problem-solving mechanisms under a common framework,
- Data- and goal-driven problem solving strategies.

A diagram of the BlackBoard is shown in Figure 3.2-2:

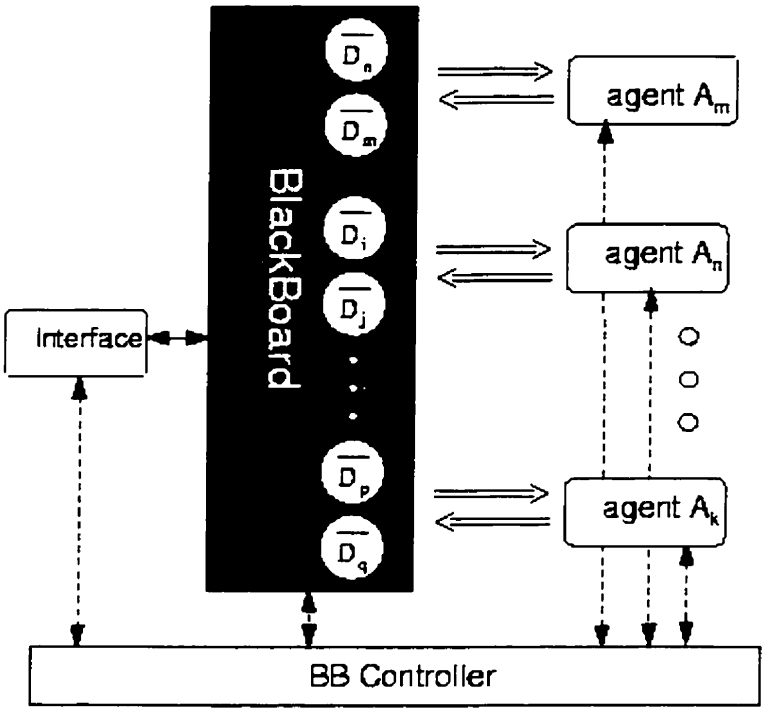

Figure 3.2-2 KBS BB architecture

Generally speaking, a BlackBoard system, like any expert system computer program, is a problem-solving engine whose architecture is based on three main components:

- The knowledge sources,
- A global data base (in this case, the BlackBoard),
- A control structure.

The Knowledge Sources (also referred to as Agents) are the parts of the system that contain the algorithms, facts, logical rules or heuristics representation needed to perform the ultimate task of the engine, namely to find a solution to a given problem. The goal of each knowledge source is to contribute pieces of information by modifying some control or data structure that is present on the blackboard that eventually leads to a solution to the problem. This is where the agents reside.

The BlackBoard is a global structure that is available to all knowledge sources – the agents – in the system. It contains problem-specific data objects needed by, or produced by, the knowledge sources, and that are part of the solution space to the problem. These data objects can then be connected to others through named links or relations, or organized into more refined structures or hierarchies, for example by partitioning or introducing layers on the BlackBoard. This is where the data types reside.

The BlackBoard is a global database accessible by all agents. It contains all the active processing data (interfaced, generated and modified by the agents). The agents performing knowledge transformation on the blackboard are independent from each other and are self-activating or data-driven, in the sense that their activation is triggered by new pieces of data being put on the blackboard. The agents examine the state of the blackboard, create new data structures and, when it is necessary, modify them by taking a data structure of one type and producing a data structure of another type.

The Control Structure is arguably the most delicate part of the system, since any action can create new conditions that in turn require drastically different problem solving strategies. It is well known that the solution reached by a complex system generally depends on the sequence of intervention of the knowledge sources – the timing in agent execution. A lot of control mechanisms are available to the developer; typically, they involve choosing a focus of attention that is triggered in priority, according to a ranking based on pre-defined evaluation criteria. This is where the context functions reside.

A BlackBoard environment such as the one described above can be viewed as a high-level set of routines and instructions, similar to a programming language, than can be used by a developer to implement knowledge rules in order to solve a specific

problem. The BlackBoard environment has been implemented in C++ rather than in a higher-level language (such as LISP or Smalltalk) to satisfy the real-time requirement of the DAAS project.

Figure 3.2-3 displays the MSDF components within the BlackBoard architecture.

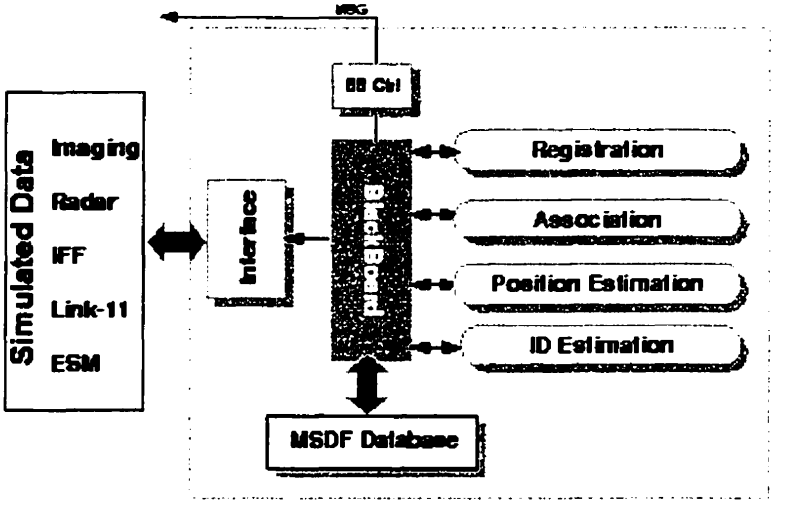

Figure 3.2-3 MSDF components

3.3 IMM-JVC IMPLEMENTATION

The relevant code for the IMM-JVC implementation is included following this section. This implementation uses one constant velocity CV model and one constant acceleration CA model for Cartesian co-ordinates and is written in C++. For a complete description of the data types and agents in the MSDF testbed see [3]. What follows is a description of one cycle of the algorithm.

- Contacts are received, sorted by type (radar, ESM, etc.), and co-ordinates are converted. The resulting contacts are available on the BB through the contact list. Existing tracks are available through the track list.

- All possible contact/track pairs are formed using the CreatePairs agent in CONTACT_TRACK_PAIR data type form. This data type, in addition to the contact and the track, contains instructions regarding what should be done next to the pair. When a CONTACT_TRACK_PAIR is placed on the BB (see line 183 of A_CreatePairs.C), those instructions are triggered. One may wonder at this point the difference between a data type and a C++ class; a data type may be defined by a class but a data type is used by the BB to trigger agents as well as to store information. The CONTACT_TRACK_PAIR data type is a derived data type, meaning that it is made up of other data types which, when active on the BB, can also trigger agents. If there are no tracks in the track list, the CreateXYTrack agent is triggered.

- Since, depending on its status, a pair can have different instructions the context function PairUpdate is used to determine which agent currently is needed to service the pair.

- The first agent to service the pair is usually that which time updates the track in the pair. For the IMM-JVC this agent is IMMTimeUpdateXYTrack. This agent does steps 1, 2, and the prediction portion of 3 in the IMM algorithm (see section 1.4.2). Since the code for this agent is fairly long it has been broken up into a number of functions included in IMM_CVCA.C and IMMRoutines.C.

- Once the time update is complete, PairUpdate sends the pair to data association. Data association is a two-step process which first uses the GateXY_XY agent then the JVCAssignment agent. Following data association, invalid pairs are

deleted using the DeletePair agent. Unassigned contacts trigger the use of the CreateXYTrack agent.

- Valid pairs are ready to have the track updated using the contact; that is, the measurement update portion of step 3, and steps 4 and 5 of the IMM algorithm may be performed using the IMMFilterXY_XY agent. The pair status is moved to ID_UPDATE which triggers the ID updating agent, which is followed by the eventual deleting of the pair through DeletePair.

- Both the IMMFilterXY_XY agent and the IMMCreateXYTrack agent spawn instances of the IMM_COMBINE_STATE data type on the BB. This data type triggers the IMMGenerateTrack agent which either instantiates a new instance of the TRACK data type on the BB or finds, updates, and activates the proper existing instance of the TRACK data type. The TRACK data type triggers an agent that updates its ID, after which the track is ready for the next contact to arrive.

As can be seen from the above description, the BB works through a set of events where a data type triggers an agent which spawns a data type which triggers an agent which ...

3.4 CONCLUSION

In this paper an algorithm, and its implementation, for tracking has been described. Chapter two of this paper is drawn from two papers co-authored by the current author and presented at separate conferences [4,5]. These papers describe, for the first time, the IMM-JVC combination.

3.5 REFERENCES

[1] E. Shahbazian et al. A Blackboard Architecture for Incremental Implementation of Data Fusion Applications. *Report*, Lockheed Martin Canada.

[2] E. Shahbazian et al. Fusion of Imaging and Non-Imaging Data for Surveillance Aircraft. *SPIE*, 1997.

[3] Demonstration of Image Analysis and Object Recognition Decision Aids for Airborne Surveillance, Detailed Design Document (DDD), Year 1. *Report*, Lockheed Martin Canada, 26 October 1998.

[4] B. Jarry, A. Jouan, H. Michalska. An IMM-JVC Algorithm for Multi-Target Tracking with Asynchronous Sensors. *Proceedings of the 19th IASTED Conference on Modelling, Information and Control*, Innsbruck, Austria, 2000, pp. 603-607.

[5] A. Jouan, B. Jarry, H. Michalska. Tracking Closely Maneuvering Targets in Clutter with an IMM-JVC Algorithm. *FUSION 2000*, in press.

```
#include "AssignmentMatrix.h"
#include "AssociationType.h"
#include "BBPtrArray.h"
#include "ContactPosition.h"
#include "IoBBFunctions.h"
#include "IoBlackBoard.h"
#include "IoExternParameters.h"
#include "IoIncludeBaseAgent.h"
#include "IoListBBPtr.h"
#include "matrix.h"
#include "PairID.h"
#include "PairTask.h"
#include "PairType.h"
#include "Prototypes.h"
#include "StateType.h"
#include "StateUpdate.h"
#include "TrackingType.h"
// IMM Includes
#include "IMMModeCombinationType.h"
#include "IMMStateUpdate.h"

void CreatePairs(IoBlackBoard *contact_buffer)
{
  AssignmentMatrix *assignment_matrix;
  AssociationType  *association_type;
  BBPtrArray       *track_list;
  Cardinal          contact_indx, track_indx, counter, contact_index,
  Cardinal         *expected_number, *list_size;
  ContactPosition  *contact_position;
  Io_Boolean       *delete_contact;
  Io_Boolean       *track_updated;
  IoBlackBoard     *pair, *assign;
  IoListBBPtr      *contact_list;
  IoListCardinal   *selected_track_list;
  IoListCardinal   *selected_track_type;
  PairID       *pair_id;
  PairTask     *pair_task;
  PairType     *pair_type;
  StateType    *contact_type;
  StateUpdate  *state_update;
  TrackingType *tracking_type;
  IMMStateUpdate          *IMM_state_update;
  IMMModeCombinationType  selectedModeCombo;

#ifdef DEBUG
  cerr<< "CreatePairs" << endl;
#endif

  /*************************************************************************
  *******/
  /**** This agents creates the contact/track pairs used for gating.  If no track
   ****/
  /**** is selected, the contacts will create new tracks and no gating (obvious
  y) ****/
  /**** will be performed.
   ****/
  /*************************************************************************
  *******/

  /*********************************/
  /**** Retrieve the Contact List.****/
  /*********************************/
  GetValueByData(contact_buffer, CONTACT_LIST, &contact_list),

  /*********************************/
  /**** Retrieve the Track List.  ****/
```



```
  /*********************************/
  GetValueByDataFirstInstance(TRACK_LIST, TRACK_LIST, &track_list);

  /*************************************************/
  /**** Allocate memory for selected track list.  ****/
  /*************************************************/
  FATAL_0_VALID(!(selected_track_list = new IoListCardinal())) != NULL,
                "Agent CreatePairs: No more space to allocate memory!!");
  FATAL_0_VALID(!(selected_track_type = new IoListCardinal())) != NULL,
                "Agent CreatePairs: No more space to allocate memory!!");

  /*************************************************************/
  /**** Select a sample of tracks (if track_list not empty). ****/
  /*************************************************************/
#ifdef DEBUG
  fprintf(stderr,"number of elements in track list: %d", track_list->NumberOfEle
ments());
#endif

  if((track_list->NumberOfElements()) != 0){
    DF_SelectTrackSample(selected_track_list, selected_track_type);
  }


  /*************************************************/
  /**** Check if the selected track list is empty.  ****/
  /*************************************************/

  if (selected_track_list->NumberOfElements() == 0){

    /*************************************************************/
    /**** Create contact/track pairs with track number = 0. ****/
    /*************************************************************/

#ifdef DEBUG
    fprintf(stderr,"CREATE_PAIRS : contact not assigned \n");
    fprintf(stderr,"number of elements in contact list= %d", contact_list->Numbe
rOfElements());
#endif

    for (contact_indx = 0 ; contact_indx < contact_list->NumberOfElements() ; co
ntact_indx++){

      /* Allocate Memory for a new CONTACT_TRACK_PAIR data type . */
      FATAL_0_VALID(!(track_updated = new Io_Boolean) != NULL,
                    "Agent CreatePairs: No more space to allocate memory!!");
      FATAL_0_VALID(!(pair_id = new PairID())) != NULL,
                    "Agent CreatePairs: No more space to allocate memory!!");
      FATAL_0_VALID(!(pair_type = new PairType) != NULL,
                    "Agent CreatePairs: No more space to allocate memory!!");
      FATAL_0_VALID(!(pair_task = new PairTask) != NULL,
                    "Agent CreatePairs: No more space to allocate memory!!");
      FATAL_0_VALID(!(tracking_type = new TrackingType) != NULL,
                    "Agent CreatePairs: No more space to allocate memory!!");

      /* Set values for all the members and attributes of a CONTACT_TRACK_PAIR
         (except member TRACK_STATE_UPDATE). */
      *pair_task = CREATE_NEW_TRACK;
      *track_updated = TRUE;
      *tracking_type      = (TrackingType) G_tracking_type;
      selectedModeCombo   = (IMMModeCombinationType) G_selectedModeCombo;

      /* Update pair id->track_number */
      pair_id->track_number = 0,
      pair_id->track_index = contact_indx;  //// **** ???????? check this one
!!
```

```
124
125         /* Update pair_id->contact_number */
126         GetValueByData(contact_list->Element(contact_indx), CONTACT_POSITION, &con
    tact_position);
127         pair_id->contact_number = contact_position->contact_number;
128
129         /* Set attribute PAIR_TYPE. */
130         GetValueByData(contact_list->Element(contact_indx), CONTACT_TYPE, &contact
    _type);
131         GetValueByData(contact_list->Element(contact_indx), DELETE_CONTACT, &delet
    e_contact);
132
133
134         // ************************************************
135         // Switch on contact type and Track Coordinate System
136         // ************************************************
137
138         switch (*contact_type){
139         case XY:
140           *pair_type = XYXY;
141 #ifdef DEBUG
142           cerr << "CreatePairs: pair_type = XYXY; no track in track_list" << endl;
143 #endif
144           FATAL_0_VALID((state_update = new StateUpdate[XY_STATE_DIM * 2])) != NU
    LL,                          \
145                         "Agent CreatePairs: No more space to allocate memory!!");
146           break;
147         case RB:
148           *pair_type = RBRB;
149           FATAL_0_VALID((state_update = new StateUpdate[RB_STATE_DIM * 2])) != NU
    LL,                          \
150                         "Agent CreatePairs: No more space to allocate memory!!");
151           break;
152         case BO:
153           *pair_type = BOBO;
154           FATAL_0_VALID((state_update = new StateUpdate[BO_STATE_DIM * 2])) != NU
    LL,                          \
155                         "Agent CreatePairs: No more space to allocate memory!!");
156           break;
157         default:
158           /**** Set the DELETE_CONTACT attribute to TRUE and activate the contact
     ****/
159           cerr << "CREATE_PAIRS: Invalid contact type.  Deleting contact list " <<
     endl;
160           for (counter = 0 ; counter < contact_list->NumberOfElements() ; counter+
    +){
161             GetValueByData(contact_list->Element(counter), DELETE_CONTACT, &delete
    _contact);
162             *delete_contact = TRUE;
163             ActivateBBData(contact_list->Element(counter));
164           }
165           DeActivateData(CONTACT_BUFFER);
166           // Exit from the routine
167           return;
168         }
169
170         // *** Instantiate the new pair on the BB. ***
171         InstantiateData(pair, CONTACT_TRACK_PAIR);
172         if ( *tracking_type == IMM_FILTER ) {
173           FATAL_0_VALID((IMM_state_update = new IMMStateUpdate[selectedModeCombo]
    != NULL, \
174                         "Agent CreatePairs: No more space to allocate memory!!");
175           SetValueByData(pair, IMM_STATE_UPDATE, IMM_state_update);
176         }
177         SetValueByData(pair, TRACK_UPDATED, track_updated);
178         SetValueByData(pair, PAIR_TYPE, pair_type);
```



```
179         SetValueByData(pair, PAIR_ID, pair_id);
180         SetValueByData(pair, TASK, pair_task);
181         SetValueByData(pair, TRACKING_TYPE, tracking_type);
182         SetValueByData(pair, TRACK_STATE_UPDATE, state_update);
183         pair->PutActOnBB();
184       }
185       delete selected_track_list;
186     }
187
188     else{      // NumberOfElements() != 0
189
190
191       //*****************************************
192       //**** Create the contact / track pairs.  ****
193       //*****************************************
194
195       for (contact_indx = 0 ; contact_indx < contact_list->NumberOfElements() ; co
    ntact_indx++){
196         for(track_indx = 0 ; track_indx < selected_track_list->NumberOfElements()
    ; track_indx++){
197
198           // Allocate Memory for a new CONTACT_TRACK_PAIR data type .
199           FATAL_0_VALID((track_updated = new Lo_Boolean) != NULL,
    \
200                         "Agent CreatePairs: No more space to allocate memory!!");
201           FATAL_0_VALID((pair_id = new PairID()) != NULL,                    \
202                         "Agent CreatePairs: No more space to allocate memory!!");
203           FATAL_0_VALID((pair_type = new PairType) != NULL,                  \
204                         "Agent CreatePairs: No more space to allocate memory!!");
205           FATAL_0_VALID((pair_task = new PairTask) != NULL,                  \
206                         "Agent CreatePairs: No more space to allocate memory!!");
207           FATAL_0_VALID((tracking_type = new TrackingType) != NULL,
      \
208                         "Agent CreatePairs: No more space to allocate memory!!");
209
210           // Set values for all the members and attributes of a
211           // CONTACT_TRACK_PAIR (except member TRACK_STATE_UPDATE).
212           *pair_task = GATTIN;
213           *track_updated = FALSE;
214           *tracking_type     = (TrackingType) G_tracking_type;
215           selectedModeCombo  = (IMMModeCombinationType) G_selectedModeCombo;
216
217           /* Update pair_id->track_number */
218           pair_id->track_number = selected_track_list->Element(track_indx);
219           pair_id->track_index = track_indx;
220
221           /* Update pair_id->contact_number */
222           GetValueByData(contact_list->Element(contact_indx), CONTACT_POSITION, &c
    ontact_position);
223           pair_id->contact_number = contact_position->contact_number;
224
225           /* Set attribute PAIR_TYPE  */
226           GetValueByData(contact_list->Element(contact_indx), CONTACT_TYPE, &conta
    ct_type);
227           GetValueByData(contact_list->Element(contact_indx), DELETE_CONTACT, &del
    ete_contact);
228
229           if(*contact_type == BO){
230             fprintf(stderr,"\n Number of Elements in Contact list =%d",  contact_l
    ist->NumberOfElements());
231             fprintf(stderr,"\n Number of Elements in Track list =%d",  selected_tr
    ack_list->NumberOfElements());
232             fprintf(stderr,"\n %dth contact element of type %d", contact_indx, *co
    ntact_type);
233             fprintf(stderr,"\n %dth track element of type %d corresponding to trac
    k number %d", track_indx, selected_track_type->Element(track_indx),
```

```
234                     selected_track_list->Element(track_indx));
235           }
236
237           switch (*contact_type){
238           case XY:
239
240             switch(selected_track_type->Element(track_indx)){
241             case XY:
242               *pair_type = XYXY;
243               FATAL_0_VALID((state_update = new StateUpdate(XY_STATE_DIM * 2))) !
    = NULL, \
244                             "Agent CreatePairs: No more space to allocate memory!!
    ");
245
246   #ifdef DEBUG
247               cerr << "CreatePair:: pair_type = XYXY" << endl;
248   #endif
249
250               break;
251             case BO:
252               *pair_type = XYBO;
253               FATAL_0_VALID((state_update = new StateUpdate(BO_STATE_DIM * 2))) !
    = NULL, \
254                             "Agent CreatePairs: No more space to allocate memory!!
    ");
255               break;
256             default:
257               cerr << "In CreatePairs:: track type undefined " << endl;
258               exit(0);
259           }
260           break;
261         case RB:
262           switch(selected_track_type->Element(track_indx)){
263           case RB:
264             *pair_type = RBRB;
265             FATAL_0_VALID((state_update = new StateUpdate(RB_STATE_DIM * 2))) !
    = NULL, \
266                           "Agent CreatePairs: No more space to allocate memory!!
    ");
267             break;
268           case BO:
269             *pair_type = RBBO;
270             FATAL_0_VALID((state_update = new StateUpdate(BO_STATE_DIM * 2))) !
    = NULL, \
271                           "Agent CreatePairs: No more space to allocate memory!!
    ");
272             break;
273           default:
274             cerr << "In CreatePairs:: track type undefined " << endl;
275             exit(0);
276           }
277           break;
278         case BO:
279           switch(selected_track_type->Element(track_indx)){
280           case XY:
281             *pair_type = BOXY;
282             FATAL_0_VALID((state_update = new StateUpdate(XY_STATE_DIM * 2))) !
    = NULL, \
283                           "Agent CreatePairs: No more space to allocate memory!!
    ");
284             break;
285           case RB:
286             *pair_type = BORB;
287             FATAL_0_VALID((state_update = new StateUpdate(RB_STATE_DIM * 2))) !
    = NULL, \
288                           "Agent CreatePairs: No more space to allocate memory!!
```



```
    ");
289             break;
290           case BO:
291             *pair_type = BOBO;
292             FATAL_0_VALID((state_update = new StateUpdate(BO_STATE_DIM * 2))) !
    = NULL, \
293                           "Agent CreatePairs: No more space to allocate memory!!
    ");
294             break;
295           default:
296             cerr << "In CreatePairs:: track type undefined." << endl;
297             exit(0);
298           }
299           break;
300         default:
301           /**** Set the DELETE_CONTACT attribute to TRUE and activate the contac
    t. ****/
302           cerr << "CREATE_PAIRS: Invalid contact type;  Deleting contact list "
    << endl;
303           for (counter = 0 ; counter < contact_list->NumberOfElements() ; counte
    r++){
304             GetValueByData(contact_list->Element(counter), DELETE_CONTACT, &dele
    te_contact);
305             *delete_contact = TRUE;
306             ActivateBBData(contact_list->Element(counter));
307           }
308           DeActivateData(CONTACT_BUFFER);
309           return;
310         }
311
312         /* Instantiate the new pair on the BB. */
313         InstantiateData(pair, CONTACT_TRACK_PAIR);
314         if ( *tracking_type == IMM_FILTER ) {
315           FATAL_0_VALID((IMM_state_update = new IMMStateUpdate(selectedModeCombo
    )) != NULL, \
316                         "Agent CreatePairs: No more space to allocate memory!!")
    ;
317           SetValueByData(pair, IMM_STATE_UPDATE, IMM_state_update);
318         }
319         SetValueByData(pair, TRACK_UPDATED, track_updated);
320         SetValueByData(pair, PAIR_TYPE, pair_type);
321         SetValueByData(pair, PAIR_ID, pair_id);
322         SetValueByData(pair, TASK, pair_task);
323         SetValueByData(pair, TRACKING_TYPE, tracking_type);
324         SetValueByData(pair, TRACK_STATE_UPDATE, state_update);
325         pair->PutActBBit();
326   #ifdef DEBUG
327         fprintf(stderr, "CREATE_PAIRS: contact %d associated to track %d \n",
    pair_id->contact_number, pair_id->track_number);
328   #endif
329
330       }
331     }
332
333     /*******************************************/
334     /**** Instantiate the ASSIGNMENT data type. ****/
335     /*******************************************/
336
337     InstantiateData(assign, ASSIGNMENT);
338     FATAL_0_VALID((assignment_matrix =
                                \
339                    new AssignmentMatrix(selected_track_list->NumberOfElements()),
     contact_list->NumberOfElements())) != NULL,     \
340                   "Agent CreatePairs: No more space to allocate memory!!");
341     FATAL_0_VALID((expected_number = new Cardinal) != NULL,
                                \
```

```
342                     "Agent CreatePairs: No more space to allocate memory!!!",
343       FATAL_O_VALID((association_type = new AssociationType) != NULL,
344                     "Agent CreatePairs: No more space to allocate memory!!!",
345       FATAL_O_VALID((list_size = new Cardinal) != NULL,
346                     "Agent CreatePairs: No more space to allocate memory!!!");
347
348       /* Set values for all the members and attributes of the Data Type ASSIGNMENT
    */
349       *expected_number = contact_list->NumberOfElements() * selected_track_list->N
    umberOfElements();
350       *list_size = 0;
351       *association_type = (AssociationType) G_association_type;
352       assignment_matrix->track_number_list = selected_track_list;
353
354       SetValueByData(assign, ASSOCIATION_TYPE, association_type);
355       SetValueByData(assign, EXPECTED_NUMBER, expected_number);
356       SetValueByData(assign, LIST_SIZE, list_size);
357       SetValueByData(assign, ASSIGNMENT_MATRIX, assignment_matrix);
358       SetValueByData(assign, CONTACT_TYPE, contact_type);
359
360       /* Put the Data Type ASSIGNMENT on the BB */
361
362       assign->PutActOnBB();
363     }
364
365     delete selected_track_type;
366
367     DeActivateData(CONTACT_BUFFER);
368     return;
369   }
370
371   IncludeBaseAgent(CreatePairs, "CONTACT_BUFFER CONTACT_TRACK_PAIR",
372                    "Create the contact/track pairs for gating");
373
374
```

```
#include "ContactID.h"
#include "ContactPosition.h"
#include "LoUUFunctions.h"
#include "LoBlackBoard.h"
#include "LoExternParameters.h"
#include "LoIncludeBaseAgent.h"
#include "OwnshipData.h"
#include "PairID.h"
#include "PairType.h"
#include "PairTask.h"
#include "Prototypes.h"
#include "Proposition.h"
#include "StateType.h"
#include "TrackState.h"

#include "TrackingType.h"
#include "IMMModeCombinationType.h"
#include "IMMTrackState.h"
#include "IMM_CVCA.h"

void CreateXYTrack(LoBlackBoard *pair)
{
  Cardinal          contact_number, index;
  ContactID        *contact_id;
  ContactPosition  *contact_position;
  LoBlackBoard     *data_ptr, *contact_ptr;
  Lo_Boolean       *delete_contact;
  PairID           *pair_id;
  PairTask         *pair_task;
  Proposition      *new_prop;
  StateType        *contact_type;
  TrackState       *track_state;
  OwnshipData      *ownship_data;
  double           x_rel, y_rel;

  TrackingType            tracking_type;
  IMMModeCombinationType  selectedModeCombo;
  IMMTrackState          *IMM_track_state;

#ifdef DEBUG
  cerr << "CreateXYTrack"<<endl;
#endif

  /**** This agent creates a new TrackState and Proposition ****/
  /**** in order to create a new XY track.                  ****/

  /**** Retrieve the contact. ****/
  GetValueByData(pair, PAIR_ID, &pair_id);
  contact_number = pair_id->contact_number;
  if ((contact_ptr = GetContact(contact_number)) == NULL)
    cerr << " Contact specified in the pair does not exist " << endl;
    return;
  }

  GetValueByData(contact_ptr, CONTACT_POSITION, &contact_position);
  GetValueByData(contact_ptr, CONTACT_ID, &contact_id);
  GetValueByData(contact_ptr, CONTACT_TYPE, &contact_type);
  GetValueByData(contact_ptr, DELETE_CONTACT, &delete_contact);

#ifdef DEBUG
    cerr << "CREATE_XY_TRACK: contact number = " << contact_number << endl;
    cerr << "CREATE_XY_TRACK: contact type   = " << *contact_type << endl;
#endif

  /**** Instantiate and fill the TRACK_STATE data type. */
```



```
  FATAL_U_VALID((track_state = new TrackState(XY_STATE_DIM * 2)) != NULL,
                \
                "Agent CreateXYTrack: No more space to allocate memory!!");

  track_state->target_number = contact_position->target_number;
  track_state->track_number = ++G_current_nb_tracks;
  track_state->type = *contact_type;
  track_state->status = INITIATED;

  track_state->time = contact_position->time;
  track_state->state_vec->me[0][0] = contact_position->state_vec->me[0][0];
  track_state->state_vec->me[1][0] = contact_position->state_vec->me[1][0];

   if (G_target_pos_coord == XY_REL_OWN) {
     track_state->state_vec->me[2][0] = 0.0;
     track_state->state_vec->me[3][0] = 0.0;
   }
   else if (G_target_pos_coord == XY_REL_OWN) {
     // Get ownship data
     GetValueByDataFirstInstance(OWNSHIP_DATA, OWNSHIP_DATA, &ownship_data);
     track_state->state_vec->me[2][0] = - ownship_data->x_vel;
     track_state->state_vec->me[3][0] = - ownship_data->y_vel;
   }

   //  track_state->state_vec->me[2][0] = 0.0;
   //  track_state->state_vec->me[3][0] = 0.0;

  track_state->cov_mat->me[0][0] = contact_position->cov_mat->me[0][0];
  track_state->cov_mat->me[0][1] = contact_position->cov_mat->me[0][1];
  track_state->cov_mat->me[1][0] = contact_position->cov_mat->me[1][0];
  track_state->cov_mat->me[1][1] = contact_position->cov_mat->me[1][1];
  track_state->cov_mat->me[2][2] = G_default_sigma_vxyair * G_default_sigma_vxya
ir;
  track_state->cov_mat->me[3][3] = G_default_sigma_vxyair * G_default_sigma_vxya
ir;


  /* Boost track covariance when not firm */
  sm_mult(G_ass_boost_factor, track_state->cov_mat, track_state->cov_mat);

  /* Should also compute x, y, altitude, vx, vy, and vz. */
  track_state->x = track_state->state_vec->me[0][0];
  track_state->y = track_state->state_vec->me[1][0];
  track_state->vx = track_state->state_vec->me[2][0];
  track_state->vy = track_state->state_vec->me[3][0];

  track_state->altitude = contact_position->altitude;

  // range and bearing always computed relative to Ownship
  if (G_target_pos_coord == XY_REL_OWN)
    {
      // Get ownship data
      GetValueByDataFirstInstance(OWNSHIP_DATA, OWNSHIP_DATA, &ownship_data);

      x_rel = contact_position->state_vec->me[0][0] - ownship_data->x_pos;
      y_rel = contact_position->state_vec->me[1][0] - ownship_data->y_pos;
    } else {
      x_rel = contact_position->state_vec->me[0][0];
      y_rel = contact_position->state_vec->me[1][0];
    }
  track_state->range   = Slant_Range(x_rel, y_rel);
  track_state->bearing = Slant_Bearing(x_rel, y_rel);

  tracking_type = (TrackingType) G_tracking_type;
  switch(tracking_type) {
```

```
130
131     case KALMAN_FILTER:
132       /**** Set the value of the TRACK_STATE data type and put it active on the BB
      ****/
133       InstantiateData(data_ptr, TRACK_STATE);
134       SetValueByData(data_ptr, TRACK_STATE, track_state);
135       data_ptr->PutActOnBB();
136       break;
137
138     case IMM_FILTER:
139       selectedModeCombo = (IMMModeCombinationType) G_selectedModeCombo;
140       FATAL_0_VALID((IMM_track_state = new IMMTrackState(selectedModeCombo)) != IMM
      LL,  \
141                     "Agent CreateXYTrack: No more space to allocate memory!!!");
142
143       switch(selectedModeCombo) {
144       case CVCA:
145         IMM_CVCA_CreateXYTrack(track_state, IMM_track_state);
146         break;
147       default:
148         cerr << "A_CreateXYTrack: selectedModeCombo undefined \n";
149       }
150
151       /**** Set the value of the IMM_COMBINE_STATE data type and put it on the BB
      ****/
152       InstantiateData(data_ptr, IMM_COMBINE_STATE);
153       SetValueByData(data_ptr, TRACK_STATE, track_state);
154       SetValueByData(data_ptr, IMM_TRACK_STATE, IMM_track_state);
155       data_ptr->PutActOnBB();
156       break;
157
158     default:
159       cerr << "A_CreateXYTrack: tracking_type undefined \n";
160     }
161
162     /**** Instantiate and fill the PROPOSITION data type. */
163     InstantiateData(data_ptr, PROPOSITION);
164     FATAL_0_VALID((new_prop = new Proposition(track_state->track_number)) != NULL,
                    \
165                   "Agent CreateXYTrack: No more space to allocate memory!!!");
166     new_prop->number_of_propositions = contact_id->number_of_propositions;
167     new_prop->proposition_origin.source = contact_id->proposition_origin.source;
168     new_prop->proposition_origin.time = contact_id->proposition_origin.time;
169     new_prop->proposition_origin.attribute_type = contact_id->proposition_origin.a
      ttribute_type;
170     new_prop->proposition_origin.attribute_value = contact_id->proposition_origin.
      attribute_value;
171
172     for (index = 0 ; index < contact_id->number_of_propositions ; index++)
173       copy (((char *) &(contact_id->proposition[index * DF_II_MAX_PROP_BYTES])),
174             ((char *) &(new_prop->proposition[index * DF_II_MAX_PROP_BYTES])), DF_II
      _MAX_PROP_BYTES);
175       new_prop->mass[index] = contact_id->mass[index];
176     }
177
178
179     SetValueByData(data_ptr, PROPOSITION, new_prop);
180     data_ptr->PutActOnBB();
181
182
183     /**** Set the PAIR_TASK attribute to DELETE_PAIR  ****/
184     GetValueByData(pair, TASK, &pair_task);
185     *pair_task = DELETE_PAIR;
186
187     /**** Set the DELETE_CONTACT attribute to TRUE and activate the contact  ****/
```



```
188     *delete_contact = TRUE;
189     ActivateBBData(contact_ptr);
190     return;
191
192   }
193
194   IncludeBaseAgent(CreateXYTrack, "CONTACT_TRACK_PAIR NEW_XY_TRACK",
195                    "Prepare information in order to create a new XY track.");
196
197
```

```
#include "IoBBFunctions.h"
#include "IoBlackBoard.h"
#include "IoIncludeBaseAgent.h"

#include "PairID.h"
#include "PairTask.h"
#include "PairType.h"
#include "StateUpdate.h"
#include "TrackingType.h"

// IMM includes
#include "IMMStateUpdate.h"

void DeletePair(IoBlackBoard *pair)
{
  Cardinal          index;
  Io_Boolean        *track_updated;
  PairTask          *pair_task;
  PairID            *pair_id;
  PairType          *pair_type;
  StateUpdate       *state_update;
  TrackingType      *tracking_type;
  IMMStateUpdate    *IMM_state_update;

#ifdef DEBUG
  cerr<< "DeletePair" << endl;
#endif

  /* This agent deletes a CONTACT_TRACK_PAIR. */

  /* Extract data pointer from blackboard and get all values  */
  (void)ExtractBBData(pair);
  GetValueByData(pair, PAIR_ID, &pair_id);
  GetValueByData(pair, TASK, &pair_task);
  GetValueByData(pair, PAIR_TYPE, &pair_type);
  GetValueByData(pair, TRACK_UPDATED, &track_updated);
  GetValueByData(pair, TRACK_STATE_UPDATE, &state_update);
  GetValueByData(pair, TRACKING_TYPE, &tracking_type);

  if ( *tracking_type == IMM_FILTER ) {
    GetValueByData(pair, IMM_STATE_UPDATE, &IMM_state_update);
    delete IMM_state_update;
  }

  /* Free all memory. */
  delete pair_id;
  delete pair_task;
  delete pair_type;
  delete track_updated;
  delete state_update;
  delete tracking_type;

  delete pair;

  return;
}

IncludeBaseAgent(DeletePair, "CONTACT_TRACK_PAIR IMM.",
                 "Delete a contact/track pair.");
```

```
1     #include <math.h>
2     #include "ContactPosition.h"
3     #include "LoBlackBoard.h"
4     #include "LoExternParameters.h"
5     #include "LoIncludeBaseAgent.h"
6     #include "LoListBBPtr.h"
7     #include "matrix.h"
8     #include "OwnshipData.h"
9     #include "PairID.h"
10    #include "PairTask.h"
11    #include "PositionType.h"
12    #include "Prototypes.h"
13    #include "StateType.h"
14    #include "TrackStatu.h"
15
16    #include "IMMModeCombinationType.h"
17    #include "IMMTrackState.h"
18    #include "IMMStateUpdate.h"
19    #include "IMM_CVCA.h"
20
21    void IMMFilterXY_XY(LoBlackBoard *pair)
22    {
23    /*-----------------------------------------*/
24    /*   IMM update of an XY track with an XY contact    */
25    /*-----------------------------------------*/
26
27      Cardinal          track_state_dim;
28      ContactPosition   *contact_position;
29      LoListBBPtr       *contact_list;
30      LoBlackBoard      *data_ptr, *track, *contact_ptr;
31      PairID            *pair_id;
32      PairTask          *pair_task;
33      PositionType      *track_position;
34      TrackState        *track_state, *state_upd;
35      OwnshipData       *ownship_data;
36      double             delta_time;
37      double             x_rel, y_rel;
38      double             covered_dist_x, covered_dist_y;
39
40      IMMModeCombinationType  modeCombo;
41      IMMTrackState           *IMM_track_state, *IMM_state_upd,
42      IMMStateUpdate          *IMM_state_update;
43
44      track_state_dim = 2 * XY_STATE_DIM;
45
46      /* Retrieve the pair. */
47      GetValueByData(pair, PAIR_ID, &pair_id);
48
49      /* Retrieve the track_state. */
50      track = GetTrack(pair_id->track_number);
51      GetValueByData(track, POSITION, &track_position);
52      track_state = track_position->state_history[0];
53
54      /* Retrieve the contact list. */
55      GetValueByDataFirstInstance(CONTACT_BUFFER, CONTACT_LIST, &contact_list);
56
57      /* Retrieve the contact. */
58      if ((contact_ptr = GetContact(pair_id->contact_number)) == NULL){
59        cerr << " Contact specified in the pair does not exist " << endl;
60        return;
61      }
62      GetValueByData(contact_ptr, CONTACT_POSITION, &contact_position);
63
64      delta_time = (contact_position->time - track_state->time);
65      if (!delta_time) delta_time = 0.001;
66
```



```
67      // Get ownship data
68      GetValueByDataFirstInstance(OWNSHIP_DATA, OWNSHIP_DATA, &ownship_data);
69
70      covered_dist_x = ownship_data->x_vel* delta_time;
71      covered_dist_y = ownship_data->y_vel* delta_time;
72
73      // Retrieve the IMM_track_state
74      GetValueByData(track, IMM_TRACK_STATE, &IMM_track_state);
75      modeCombo = IMM_track_state->modeCombo;
76
77      if (track_state->status == INITIATED){
78        // An XY Contact updates an initiated XY Track. No IMM
79        // filter is applied but the positional data and the
80        // covariance matrix must be updated.
81
82        // Start with the TRACK_STATE
83        FATAL_O_VALID((state_upd = new TrackState(track_state_dim)) != NULL,
                \
84                      "A_IMMFilterXY_XY: No more space to allocate memory!!");
85
86        state_upd->target_number = contact_position->target_number;
87        state_upd->track_number = track_state->track_number;
88        state_upd->type = track_state->type;
89        state_upd->status = TENTATIVE;
90        state_upd->time = contact_position->time;
91
92        /* Update the Track. */
93        if (((G_target_pos_coord == XY_REL_DIRP) && (G_target_vel_coord == VXVY_ABS)
    )
94            || ((G_target_pos_coord == XY_REL_OWNS) && (G_target_vel_coord == VXVY_R
    EL_OWNS)))
95          {
96            // Standard case, no velocity correction required :
97            // Either everything absolute or everything relative
98            state_upd->state_vec->me[0][0] = contact_position->state_vec->me[0][0];
99            state_upd->state_vec->me[1][0] = contact_position->state_vec->me[1][0];
100           state_upd->state_vec->me[2][0] = (contact_position->state_vec->me[0][0]
                                                 track_state->state_vec->me[0][0])/delt
    a_time;
102           state_upd->state_vec->me[3][0] = (contact_position->state_vec->me[1][0]
                                                 track_state->state_vec->me[1][0])/delt
    a_time;
104   #ifdef DEBUG
105           cerr << " ABS/ABS or REL/REL XY State Vector Updated " << endl;
106   #endif
107         }
108       else if ((G_target_pos_coord == XY_REL_OWNS) && (G_target_vel_coord == VXVY_
    ABS))
109         {
110           state_upd->state_vec->me[0][0] = contact_position->state_vec->me[0][0];
111           state_upd->state_vec->me[1][0] = contact_position->state_vec->me[1][0];
112           state_upd->state_vec->me[2][0] = (contact_position->state_vec->me[0][0]
                                                 track_state->state_vec->me[0][0] + cov
    ered_dist_x)/delta_time;
114           state_upd->state_vec->me[3][0] = (contact_position->state_vec->me[1][0]
                                                 track_state->state_vec->me[1][0] + cov
    ered_dist_y)/delta_time;
116   #ifdef DEBUG
117           cerr << "REL/ABS XY State Vector Updated " << endl;
118   #endif
119         } else {
120           cerr << "A_IMMFilterXY_XY :: Target position or velocity Coordinate not
```

```
    supported * << endl;
131         exit(0);
132       }
133
134     // For initializing, the current and prior contacts are assumed uncorrelated
    ?
135     state_upd->cov_mat->me[0][0] = contact_position->cov_mat->me[0][0];
136     state_upd->cov_mat->me[0][1] = contact_position->cov_mat->me[0][1];
137     state_upd->cov_mat->me[0][2] = contact_position->cov_mat->me[0][0]/delta_tim
    e;
138     state_upd->cov_mat->me[0][3] = contact_position->cov_mat->me[0][1]/delta_tim
    e;
139     state_upd->cov_mat->me[1][0] = contact_position->cov_mat->me[1][0];
140     state_upd->cov_mat->me[1][1] = contact_position->cov_mat->me[1][1];
141     state_upd->cov_mat->me[1][2] = contact_position->cov_mat->me[1][0]/delta_tim
    e;
142     state_upd->cov_mat->me[1][3] = contact_position->cov_mat->me[1][1]/delta_tim
    e;
143     state_upd->cov_mat->me[2][0] = contact_position->cov_mat->me[0][0]/delta_tim
    e;
144     state_upd->cov_mat->me[2][1] = contact_position->cov_mat->me[0][1]/delta_tim
    e;
145     state_upd->cov_mat->me[2][2] = (contact_position->cov_mat->me[0][0]+track_st
    ate->cov_mat->me[0][0])/(delta_time*delta_time);
146     state_upd->cov_mat->me[2][3] = (contact_position->cov_mat->me[0][1]+track_st
    ate->cov_mat->me[0][1])/(delta_time*delta_time);
147     state_upd->cov_mat->me[3][0] = contact_position->cov_mat->me[1][0]/delta_tim
    e;
148     state_upd->cov_mat->me[3][1] = contact_position->cov_mat->me[1][1]/delta_tim
    e;
149     state_upd->cov_mat->me[3][2] = (contact_position->cov_mat->me[1][0]+track_st
    ate->cov_mat->me[1][0])/(delta_time*delta_time);
150     state_upd->cov_mat->me[3][3] = (contact_position->cov_mat->me[1][1]+track_st
    ate->cov_mat->me[1][1])/(delta_time*delta_time);
151
152     // Should also compute x, y, altitude, vx, vy, and vz.
153     // x and y stored in same coordinates as the state vector
154     state_upd->x = state_upd->state_vec->me[0][0];
155     state_upd->y = state_upd->state_vec->me[1][0];
156     state_upd->altitude = contact_position->altitude;
157     state_upd->vx = state_upd->state_vec->me[2][0];
158     state_upd->vy = state_upd->state_vec->me[3][0];
159
160     // range and bearing always computed relative to Ownship
161     if (G_target_pos_coord == XY_REL_OWN)
162       {
163         x_rel = state_upd->state_vec->me[0][0] - ownship_data->x_pos;
164         y_rel = state_upd->state_vec->me[1][0] - ownship_data->y_pos;
165       } else {
166         x_rel = state_upd->state_vec->me[0][0];
167         y_rel = state_upd->state_vec->me[1][0];
168       }
169     state_upd->range   = Slant_Range(x_rel, y_rel);
170     state_upd->bearing = Slant_Bearing(x_rel, y_rel);
171
172     // Set up the IMM_TRACK_STATE
173     FATAL_0_VALID((IMM_state_upd = new IMMTrackState(modeCombo)) != NULL,
174                   "A_IMMFilterXY_XY: No more space to allocate memory!!");
175     m_copy(IMM_track_state->modemu, IMM_state_upd->modemu);
176
177     switch(modeCombo) {
178     case CVCA:
179       IMM_CVCA_InitFiltTrack(state_upd, IMM_state_upd);
180       break;
181     default:
182       cerr << "A_IMMFilterXY_XY: selectedModeCombo undefined \n";
```



```
183     }
184
185     /**** Set the value of the IMM_COMBINE_STATE data type and put it active on
    the BB. ****/
186     InstantiateData(data_ptr, IMM_COMBINE_STATE);
187     SetValueByData(data_ptr, TRACK_STATE, state_upd);
188     SetValueByData(data_ptr, IMM_TRACK_STATE, IMM_state_upd);
189     data_ptr->PutActOnBB();
190
191     /**** Set the PAIR_TASK attribute to ID_UPDATE. ****/
192     GetValueByData(pair, TASK, &pair_task);
193     *pair_task = ID_UPDATE;
194   #ifdef DEBUG
195     fprintf(stdout, "IMMFilterXY XY: fuse target %d and track %d \n", state_up
    d->target_number, pair_id->track_number);
196   #endif
197   }
198   else if ((track_state->status == TENTATIVE ) ||
199            (track_state->status == FIRM )){
200
201     /* Get the Track_State_Update and IMM_State_Update from the pair*/
202     GetValueByData(pair, IMM_STATE_UPDATE, &IMM_state_update);
203
204     /*****************************************************************/
205     /*          The input of the IMM filter for a XY Contact is prepared.
                  */
206     /*          We need the measurement "Z" and the covariance matrix "R".
              */
207     /*****************************************************************/
208
209     /* Update the Track's parameters */
210     FATAL_0_VALID((state_upd = new TrackState(track_state_dim)) != NULL,
                  \
211                   "Agent CreateXYTrack: No more space to allocate memory!!");
212
213     state_upd->target_number = contact_position->target_number;
214     state_upd->track_number = track_state->track_number;
215     state_upd->type = track_state->type;
216     state_upd->status = FIRM;
217     state_upd->time = contact_position->time;
218
219     // Here is where it happens
220     FATAL_0_VALID((IMM_state_upd = new IMMTrackState(modeCombo)) != NULL,  \
221                   "A_IMMFilterXY_XY: No more space to allocate memory!!");
222
223     switch(modeCombo) {
224     case CVCA:
225       IMM_CVCA_FiltTrack(IMM_state_update, contact_position, IMM_state_upd, stat
    e_upd);
226       break;
227     default:
228       cerr << "A_IMMFilterXY_XY: selectedModeCombo undefined \n";
229     }
230
231     /* Should also compute x, y, altitude, vx, vy, and vz  */
232     state_upd->x = state_upd->state_vec->me[0][0];
233     state_upd->y = state_upd->state_vec->me[1][0];
234     state_upd->altitude = contact_position->altitude;
235     state_upd->vx = state_upd->state_vec->me[2][0];
236     state_upd->vy = state_upd->state_vec->me[3][0];
237
238     // range and bearing always computed relative to Ownship
239     if (G_target_pos_coord == XY_REL_OWN)
240       {
241         x_rel = state_upd->state_vec->me[0][0] - ownship_data->x_pos;
242         y_rel = state_upd->state_vec->me[1][0] - ownship_data->y_pos;
```

```
233         } else {
234           x_rel = state_upd->state_vec->me[0][0];
235           y_rel = state_upd->state_vec->me[1][0];
236         }
237       state_upd->range   = Slant_Range(x_rel, y_rel);
238       state_upd->bearing = Slant_Bearing(x_rel, y_rel);
239
240       /**** Set the value of the IMM_COMBINE_STATE data type and put it active on
      the BB. ****/
241       InstantiateData(data_ptr, IMM_COMBINE_STATE);
242       SetValueByData(data_ptr, TRACK_STATE, state_upd);
243       SetValueByData(data_ptr, IMM_TRACK_STATE, IMM_state_upd);
244       data_ptr->PutActOnBB();
245
246       /**** Set the PAIR_TASK attribute to ID_UPDATE. ****/
247       GetValueByData(pair, TASK, &pair_task);
248       *pair_task = ID_UPDATE;
249
250   #ifdef DEBUG
251       fprintf(stderr, "A_IMMFilterXY_XY: fuse target %ld and track %ld \n", state
      upd->target_number, pair_id->track_number);
252   #endif
253     }
254
255     return;
256   }
257
258   IncludeBaseAgent(IMMFilterXY_XY, "CONTACT_TRACK_PAIR IMM_XY_XY", "State Update o
      f an XY Track with an XY Contact using an IMM Filter.");
```

```
1     #include "BBPtrArray.h"
2     #include "Identity.h"
3     #include "LoBBFunctions.h"
4     #include "LoBlackBoard.h"
5     #include "LoError.h"
6     #include "LoExternParameters.h"
7     #include "LoIncludeBaseAgent.h"
8     #include "PositionType.h"
9     #include "Prototypes.h"
10    #include "TrackState.h"
11
12    #include "IMMModeCombinationType.h"
13    #include "IMMTrackState.h"
14    #include "IMMStateUpdate.h"
15
16    #define Track_Output "HSDFTrack#"
17
18    void IMMGenerateTrack(LoBlackBoard *data_ptr)
19    {
20      Lo_Boolean     new_track = FALSE;
21      BBPtrArray    *track_list;
22      Identity      *identity, *track_identity;
23      LoBlackBoard  *track;
24      PositionType  *position;
25      TrackState    *track_state;
26      IMMTrackState *IMM_track_state;
27      char temp_name[20], track_number[20];
28      FILE *fd;
29
30      GetValueByData(data_ptr, TRACK_STATE, &track_state);
31      GetValueByData(data_ptr, IMM_TRACK_STATE, &IMM_track_state);
32
33    /* Remove contact from blackboard and delete data. */
34      (void)ExtractActBBData(data_ptr);
35
36      delete data_ptr;
37
38    /* If track has not been found on the blackboard, allocate memory for a new trac
      k. */
39    /* Reuse the same LoBlackBoard pointer.          */
40      if((track = GetTrack(track_state->track_number)) == NULL){
41        new_track = TRUE;
42        FATAL_O_VALID((identity = new Identity(track_state->track_number)) != NULL,
                        \
43                        "Agent GenerateTrack: No more space to allocate memory!!");
44        FATAL_O_VALID((position = new PositionType()) != NULL,
                        \
45                        "Agent GenerateTrack: No more space to allocate memory!!");
46      }
47      else{
48        GetValueByData(track, POSITION, &position);
49        SetValueByData(track, IMM_TRACK_STATE, IMM_track_state);
50      }
51
52    /* Add new contact into track. */
53      position->AddTrackState(track_state);
54
55    /* If new track, instantiate it (active) on the blackboard  */
56      if(new_track){
57        InstantiateData(track, TRACK);
58        SetValueByData(track, IDENTITY, identity);
59        SetValueByData(track, POSITION, position);
60        SetValueByData(track, IMM_TRACK_STATE, IMM_track_state);
61
62    /*  Put the track on the track list. */
63        GetValueByDataFirstInstance(TRACK_LIST, TRACK_LIST, &track_list);
```



```
64        track_list->IncludeElement(track);
65
66        track->PutActOnBB();
67      }
68      else{
69        /* Activate only the track to be processed  */
70        ActivateBBData(track);
71      }
72
73    #ifdef DEBUG
74      // Create as many text file as tracks and write their
75      // state vectors and covariance matrices in it.
76      fprintf(stderr,"\nWrite IMM Tracks");
77      sprintf(track_number, "_%d", track_state->track_number);
78      strcpy(temp_name,Track_Output);
79      strcat(temp_name,track_number);
80      fprintf(stderr,"\nSave data in file %s", temp_name);
81      fd = (FILE *) fopen(temp_name, "a");
82      /* print in file with the format: X, Y , Vx, Vy, covx, covy */
83      fprintf(fd, "%f %f %f %f %f %f %f\n",
84              position->state_history[0]->time,
85              position->state_history[0]->state_vec->mu[0][0],
86              position->state_history[0]->state_vec->mu[1][0],
87              position->state_history[0]->state_vec->mu[2][0],
88              position->state_history[0]->state_vec->mu[3][0],
89              position->state_history[0]->cov_mat->mu[0][0],
90              position->state_history[0]->cov_mat->mu[1][1]);
91      fclose(fd);
92    #endif
93
94      // Send Track data to HCI
95      OutputTrack(track);
96      return;
97    }
98
99    IncludeBaseAgent(IMMGenerateTrack, "IMM_COMBINE_STATE TRACK", "Generate or updat
      e track with IMM values ");
100
101
102
103
104
105
106
```

```
#include <math.h>
#include "BBPtrArray.h"
#include "ContactPosition.h"
#include "LoBlackBoard.h"
#include "LoListBBPtr.h"
#include "LoIncludeBaseAgent.h"
#include "LoExternParameters.h"
#include "matrix.h"
#include "OwnshipData.h"
#include "PairID.h"
#include "PositionType.h"
#include "Prototypes.h"
#include "StateUpdate.h"
#include "TrackState.h"

// IMM includes
#include "IMMModeCombinationType.h"
#include "IMMStateUpdate.h"
#include "IMMTrackState.h"
#include "IMM_CVCA.h"
#include "IMMRoutines.h"

void IMMTimeUpdateXYTrack(LoBlackBoard *pair)
{
/*-------------------------------*/
/*     IMMTimeUpdate XY track    */
/*-------------------------------*/

  Cardinal                 contact_number;
  Lo_Boolean               *track_updated;
  LoBlackBoard             *track, *contact_ptr;
  LoListBBPtr              *contact_list;
  PairID                   *pair_id;
  ContactPosition          *contact_position;
  OwnshipData              *ownship_data;
  PositionType             *track_position;
  StateUpdate              *state_update;
  TrackState               *track_state;
  IMMStateUpdate           *IMM_state_update;
  IMMTrackState            *IMM_track_state;
  double                   delta_time, q;
  IMMModeCombinationType   modeCombo;
  Cardinal                 i;
  VEC                      *x;
  MAT                      *P;

  GetValueByData(pair, PAIR_ID, &pair_id);

  // Retrieve the track_state.
  track = GetTrack(pair_id->track_number);
  GetValueByData(track, POSITION, &track_position);
  track_state = track_position->state_history[0];

  // Retrieve the contact list
  GetValueByDataFirstInstance(CONTACT_BUFFER, CONTACT_LIST, &contact_list);

  // Retrieve the contact_number, contact_position
  contact_number = pair_id->contact_number;
  if ((contact_ptr = GetContact(contact_number)) == NULL){
    cerr << " Contact specified in the pair does not exist " << endl;
    return;
  }
  GetValueByData(contact_ptr, CONTACT_POSITION, &contact_position);

  // Compute time since last track update:
  delta_time = (contact_position->time - track_state->time);
```



```
  if (delta_time == 0) delta_time = 0.001;

  /* Set the value of attribute for pair. */
  GetValueByData(pair, TRACK_UPDATED, &track_updated);
  *track_updated = TRUE;

  /* Set the value of the member TRACK_STATE_UPDATE for the pair. */
  GetValueByData(pair, TRACK_STATE_UPDATE, &state_update);

  // Retrieve IMM values
  GetValueByData(track, IMM_TRACK_STATE, &IMM_track_state);
  GetValueByData(pair, IMM_STATE_UPDATE, &IMM_state_update);
  modeCombo = (IMMModeCombinationType) IMM_track_state->modeCombo;

  // Copy in state update the track state prior to time update
  m_copy(track_state->state_vec, state_update->state_vec);
  m_copy(track_state->cov_mat,   state_update->cov_mat);
  m_copy(IMM_track_state->modex, IMM_state_update->modex);
  m_copy(IMM_track_state->modeP, IMM_state_update->modeP);

  if ((track_state->status == TENTATIVE ) ||
      (track_state->status == FIRM )){

    /*************************************************/
    /* An XY Contact updates a tentative or firm XY Track.    */
    /*************************************************/

    // If relative position AND absolute velocity,
    // have to compensate for ownship motion between the two points
    if ((G_target_pos_coord == XY_REL_OWNS) && (G_target_vel_coord == VXVY_ABS))
  {
      GetValueByDataFirstInstance(OWNSHIP_DATA, OWNSHIP_DATA, &ownship_data);
    }

    // Get the current process noise scalar
    q = DF_ProcessNoiseScalar(track_position, contact_position, delta_time);

    switch(modeCombo){

    case CVCA:
      IMM_CVCA_TimeUpdate(state_update, IMM_track_state, IMM_state_update, delta_ti
  me, q, ownship_data);
      break;

    default:
      cerr << "A IMMTimeUpdateXYTrack: IMM mode combination not selected.\n";
    }
  }

  else if(track_state->status == INITIATED){

    /*************************************************/
    /* An XY Contact updates an initiated XY Track.          */
    /* (No time update is performed)                 */
    /*************************************************/

    // If working in RELATIVE coordinates, make sure target is initially
    // FIXED w/r to the absolute referential, by removing the ownship motion
    // Only position is corrected, no velocity for initiated track

    if (G_target_pos_coord == XY_REL_OWNS) {

      GetValueByDataFirstInstance(OWNSHIP_DATA, OWNSHIP_DATA, &ownship_data);

      // Subtract Ownship motion from the position update
      state_update->state_vec->me[0][0] -= ownship_data->x_vel * delta_time;
```

```
131         state_update->state_vec->me[1][0] -= ownship_data->y_vel * delta_time;
132
133         // Do the same for the IMM modes
134         for ( i = 0; i < IMM_state_update->modex->m; i++ ) {
135           IMM_state_update->modex->me[0][i] -= ownship_data->x_vel * delta_time;
136           IMM_state_update->modex->me[1][i] -= ownship_data->y_vel * delta_time;
137         }
138       }
139     }
140     else {
141       cerr << "IMM Time Update XY ::  Unrecognized track status " << endl;
142     }
143
144     return;
145   }
146
147   IncludeBaseAgent(IMMTimeUpdateXYTrack, "CONTACT_TRACK_PAIR IMM TIME UPD_XY", "IM
      M Time Update of an XY Track.");
148
```

```
1     #include <math.h>
2     #include "AssignmentMatrix.h"
3     #include "AssociationType.h"
4     #include "BBPtrArray.h"
5     #include "ContactPosition.h"
6     #include "ContactID.h"
7     #include "LoBBFunctions.h"
8     #include "LoBlackBoard.h"
9     #include "LoListBBPtr.h"
10    #include "LoIncludeBaseAgent.h"
11    #include "LoExternParameters.h"
12    #include "matrix.h"
13    #include "PairID.h"
14    #include "PairTask.h"
15    #include "PairType.h"
16    #include "PositionType.h"
17    #include "Prototypes.h"
18    #include "StateType.h"
19    #include "StateUpdate.h"
20    #include "TrackingType.h"
21
22    // IMM Includes
23    #include "IMMModeCombinationType.h"
24    #include "IMMStateUpdate.h"
25
26    void JVCAssignment(LoBlackBoard *assign)
27    {
28      /*************************************************************/
29      /* Search for Optimal Assignement according to JVC Algorithm */
30      /*************************************************************/
31
32
33      int  ico, itrk;
34      int  i, m, n;
35      AssignmentMatrix *assignment_matrix;
36      AssociationType  *association_type;
37      BBPtrArray       *track_list;
38      Cardinal         *expected_number, *list_size;
39      Cardinal          track_number;
40      ContactPosition  *contact_position;
41      ContactID        *contact_id;
42      LoBlackBoard     *pair, *track_ptr;
43      Lo_Boolean       *track_updated;
44      Lo_Boolean       *delete_contact;
45      LoListBBPtr      *contact_list;
46      PairID        *pair_id;
47      PairTask      *pair_task;
48      PairType      *pair_type;
49      PositionType *track_position;
50      StateType     *contact_type;
51      StateUpdate   *state_update;
52      TrackingType *tracking_type;
53      float         *cost_matrix, *U, *V;
54      int           *first, *obj, *X, *Y;
55      int            oldui, arc;
56      float          large, totalcost;
57      MAT           *matrix;
58      IMMStateUpdate          *IMM_state_update;
59      IMMModeCombinationType  selectedModeCombo;
60
61
62    #ifdef DEBUG
63      printf("\nJVC_Assignment starts\n");
64    #endif
65
66      // Retrieve the Assignment Matrix.
```



```
67      GetValueByData(assign, ASSIGNMENT_MATRIX, &assignment_matrix);
68      // Retrieve the contact_list (to get the NumberOfElements)
69      GetValueByData(FirstInstance(CONTACT_BUFFER, CONTACT_LIST, &contact_list);
70      // Retrieve the track_list (to get the NumberOfElements)
71      GetValueByData(FirstInstance(TRACK_LIST, TRACK_LIST, &track_list);
72
73      // Reserve the memory for the Pairs array.
74
75      m = assignment_matrix->track_number_list->NumberOfElements();
76      n = contact_list->NumberOfElements();
77
78      // Reserve the memory for the cost matrix
79      FATAL_O_VALID((cost_matrix = (float *) calloc(40000, sizeof(float)))!= NULL,
80                    "Agent JVC : No more space to allocate memory!!");
81
82      // Reserve the memory for the array which will contain pointers to the start o
    f the data
83      //    for each rows in the array cost_matrix
84      FATAL_O_VALID((first = (int *) calloc(8001, sizeof(int)))!= NULL,
85                    "Agent JVC : No more space to allocate memory!!");
86
87      // Reserve the memory for the array which will contain the column index for ea
    ch value
88      //    in the array cost_matrix
89      FATAL_O_VALID((obj = (int *) calloc(40000, sizeof(int)))!= NULL,
90                    "Agent JVC: No more space to allocate memory!!");
91
92      // Reserve the memory for the array X.
93      //    X[i] is the index of the column (track) assigned to row (contact) i
94      FATAL_O_VALID((X = (int *) calloc(8000, sizeof(int)))!= NULL,
95                    "Agent JVC: No more space to allocate memory!!");
96
97      // Reserve the memory for the array Y.
98      //    Y[j] is the index of the row (contact) assigned to column (track) j
99      FATAL_O_VALID((Y = (int *) calloc(8000, sizeof(int)))!= NULL,
100                   "Agent JVC: No more space to allocate memory!!");
101
102     // Reserve the memory for the array U.
103     //    U[i] is a dual variable, the dual price of row i
104     FATAL_O_VALID((U = (float *) calloc(8000, sizeof(float)))!= NULL,
105                   "Agent JVC: No more space to allocate memory!!");
106
107     // Reserve the memory for the array V.
108     //    V[j] is a dual variable, the dual price of column j
109     FATAL_O_VALID((V = (float *) calloc(8000, sizeof(float)))!= NULL,
110                   "Agent JVC: No more space to allocate memory!!");
111
112     matrix = m_get(m+1,m+1);
113
114     oldui=0,
115     arc=0,
116
117     large = 0.9999 prob_max;
118
119     for(ico=0;ico<n;ico++)
120     {
121       for(itrk=0;itrk<m;itrk++)
122       {
123         matrix->me[ico+1][itrk+1]=assignment_matrix->probability->me[ico][itrk];
124       }
125     }
126
127   #ifdef DEBUG
128     for(ico=1;ico<=n;ico++)
129     {
130       for(itrk=1;itrk<=m;itrk++)
```

```
131       {
132         printf("\nassignment_matrix[%d][%d]=%f", ico, itrk,
133                matrix->me[ico][itrk]);
134       }
135     }
136   #endif
137
138     for(ico=1;ico<=n;ico++)
139     {
140       for(itrk=1;itrk<=m;itrk++)
141       {
142         if(ico!=oldm1)
143           do{
144             arc++;
145             cost_matrix[arc]=large;
146             oldm1++;
147             obj[arc]=m+oldm1;
148             first[oldm1]=arc;
149           }while(ico>oldm1);
150         arc++;
151         obj[arc]=itrk;
152         cost_matrix[arc]=matrix->me[ico][itrk];
153   #ifdef DEBUG
154         printf("\ncost[%d]=%f\n", arc, cost_matrix[arc]);
155   #endif
156       }
157     }
158     while(oldm1<n)
159     {
160       arc++;
161       cost_matrix[arc]=large;
162       oldm1++;
163       obj[arc]=m+oldm1;
164       first[oldm1]=arc;
165     }
166
167     first[n+1]=arc+1;
168
169     m=m+n;
170
171     DF_JVC( n, m, cost_matrix, obj, first, X, Y, U, V, &totalcost);
172
173     m = m-n;
174
175   #ifdef DEBUG
176     printf("\nm = %d n = %d \n", m, n);
177
178     for(ico=1; ico<=n;ico++)
179       printf("\nRow %d Column %d Cost %f\n", ico, X[ico], U[ico]);
180
181     for(ico=1; ico<=m;ico++)
182       printf("\nColumn %d Row %d Cost %f\n", ico, Y[ico], V[ico]);
183   #endif
184
185     //*****************************************************************
186     // Find the pairs.
187     //   Two ways are actually possible according to which result is used:
188
189     //     1) X[I] gives  the index of the column (the track)
190     // associated to the row I (contact index), the dual price of this
191     // association is contained in the matrix U[I];
192     // 2) Y[J] is the index of the row (the contact)
193     // associated to the column J (track index), the dual price of this
194     // association is contained in the matrix V[J].
195     //
196     //     Theoretically, the algorithm assumes that each row has at least one
```



```
197     // possible association.
198     // Practically this can be overcomed by creating pairs for X[I] or Y[J] if
199     // their U[I] or V[J] is different from 9999 or 0.
200     //
201     // *****************************************************************
202
203     // In the following we select the pairs using the Y[J] and V[J] outputs
204
205
206     for(ico=1; ico<=n;ico++)
207     {
208       if ((matrix->me[ico][X[ico]] != G_nass_prob_max) && (X[ico] <=m))
209       {
210
211   #ifdef DEBUG
212         printf("\nmatrix[%d][%d]=%f, Assigned -> new PAIR\n",ico, X[ico], matrix->
      me[ico][X[ico]]);
213   #endif
214
215         // This is a new pair.
216         // Allocate Memory for a new pair.
217         FATAL_O_VALID((track_updated = new Lo_Boolean) != NULL,                    \
218                       "Agent JVC: No more space to allocate memory!!");
219         FATAL_O_VALID((pair_id = new PairID()) != NULL,                            \
220                       "Agent JVC: No more space to allocate memory!!");
221         FATAL_O_VALID((pair_type = new PairType) != NULL,                          \
222                       "Agent JVC: No more space to allocate memory!!");
223         FATAL_O_VALID((pair_task = new PairTask) != NULL,                          \
224                       "Agent JVC: No more space to allocate memory!!");
225         FATAL_O_VALID((tracking_type = new TrackingType) != NULL,
      \
226                       "Agent JVC: No more space to allocate memory!!");
227
228
229
230         // The co_num represents the contact index (and not the contact number)
231         // so the following line is OK.
232
233         GetValueByData(contact_list->Element(ico-1), CONTACT_POSITION, &contact_po
      sition);
234         GetValueByData(contact_list->Element(ico-1), CONTACT_TYPE, &contact_type);
235         GetValueByData(contact_list->Element(ico-1), CONTACT_ID, &contact_id);
236
237         pair_id->contact_number = contact_position->contact_number;
238         //      printf("\ncontact #%d", pair_id->contact_number);
239         //      fprintf(stderr, "\nX[ico]-1=%d", X[ico]-1);
240
241         track_number = assignment_matrix->track_number_list->Element(X[ico]-1);
242         //      printf("\ntrack #%d", track_number);
243
244         if((track_ptr = GetTrack(track_number)) == NULL){
245           cout << " Track specified in the pair does not exist " << endl;
246           return;
247         }
248
249         GetValueByData(track_ptr, POSITION, &track_position);
250
251         *track_updated = FALSE;
252         *pair_task = POS_UPDATE;
253         *tracking_type = (TrackingType) G_tracking_type;
254         selectedNodeCombo  = (IHHNodeCombinationType) G_selectedNodeCombo;
255
256         pair_id->track_number = track_number;
257
258   #ifdef DEBUG
259     printf("\nNumber of elements in the track list=%d", assignment_matrix->track_n
```

```
    umber_list->NumberOfElements());
260   for (i=0;i<assignment_matrix->track_number_list->NumberOfElements();i++)
261     printf("\n Element %d = track number=%d", i,assignment_matrix->track_number
    _list->Element(i));
262   printf("\nassigned pair %d to track %d", pair_id->contact_number, pair_id->t
    rack_number);
263   printf("\ntrack type %d", track_position->state_history[0]->type);
264 #endif
265
266       switch (track_position->state_history[0]->type){
267
268       case XY:
269         switch (*contact_type){
270         case XY:
271           *pair_type = XYXY;
272           break;
273         case BO:
274           *pair_type = BOXY;
275           break;
276         default:
277           fprintf(stderr, "In JVC: invalid contact type \n");
278           exit(0);
279         }
280         FATAL_O_VALID((state_update = new StateUpdate(XY_STATE_DIM * 2)) != NU
    LL,                    \
281                       "Agent JVC: No more space to allocate memory!!");
282         break;
283       case RB:
284         switch (*contact_type){
285         case RB:
286           *pair_type = RBRB;
287           break;
288         case BO:
289           *pair_type = BORB;
290           break;
291         default:
292           fprintf(stderr, "In JVC: invalid contact type \n");
293           exit(0);
294         }
295         FATAL_O_VALID((state_update = new StateUpdate(RB_STATE_DIM * 2)) != NU
    LL,                    \
296                       "Agent JVC: No more space to allocate memory!!");
297         break;
298       case BO:
299         switch (*contact_type){
300         case XY:
301           *pair_type = XYBO;
302           break;
303         case RB:
304           *pair_type = RBBO;
305           break;
306         case BO:
307           *pair_type = BOBO;
308           break;
309         default:
310           fprintf(stderr, "In JVC: invalid contact type \n");
311           exit(0);
312         }
313
314         FATAL_O_VALID((state_update = new StateUpdate(BO_STATE_DIM * 2)) != NU
    LL,                    \
315                       "Agent JVC: No more space to allocate memory!!");
316         break;
317       default:
318         fprintf (stderr, "In JVC: invalid track type \n");
319         exit(0);
```



```
320
321       }
322
323       // Instantiate the new pair on the BB.
324       InstantiateData(pair, CONTACT_TRACK_PAIR);
325       if ( *tracking_type == IMM_FILTER ) {
326         FATAL_O_VALID((IMMStateUpdate = new IMMStateUpdate(selectedModeCombo))
    != NULL, \
327                       "Agent JVC: No more space to allocate memory!!");
328         SetValueByData(pair, IMM_STATE_UPDATE, IMM_state_update);
329       }
330       SetValueByData(pair, TRACK_UPDATED, track_updated);
331       SetValueByData(pair, PAIR_TYPE, pair_type);
332       SetValueByData(pair, PAIR_ID, pair_id);
333       SetValueByData(pair, TASK, pair_task);
334       SetValueByData(pair, TRACKING_TYPE, tracking_type);
335       SetValueByData(pair, TRACK_STATE_UPDATE, state_update);
336       pair->PutActOnBB();
337     }
338     else{
339       // Unassigned Contacts are given a track_number of "MAX_TRACK".
340       // There will become new Tracks.
341
342       // This is an unassigned Contact that will become a new Track.
343       // This is a new pair.
344       // Allocate Memory for a new pair.
345
346 #ifdef DEBUG
347   printf("\nmatrix[%d][%d]=%f, unassigned => new TRACK\n",ico, X[ico], matrix->m
    v[ico][X[ico]]);
348 #endif
349
350       GetValueByData(contact_list->Element(ico-1), CONTACT_POSITION, &contact_
    position);
351       GetValueByData(contact_list->Element(ico-1), CONTACT_TYPE, &contact_type
    );
352       GetValueByData(contact_list->Element(ico-1), CONTACT_ID, &contact_id);
353
354       FATAL_O_VALID((track_updated = new Lo_Boolean) != NULL,
    \
355                       "Agent JVC: No more space to allocate memory!!");
356       FATAL_O_VALID((pair_id = new PairID()) != NULL,                    \
357                       "Agent JVC: No more space to allocate memory!!");
358       FATAL_O_VALID((pair_type = new PairType) != NULL,                  \
359                       "Agent JVC: No more space to allocate memory!!");
360       FATAL_O_VALID((pair_task = new PairTask) != NULL,                  \
361                       "Agent JVC: No more space to allocate memory!!");
362       FATAL_O_VALID((tracking_type = new TrackingType) != NULL,
    \
363                       "Agent JVC: No more space to allocate memory!!");
364
365       *track_updated = TRUE;
366       *tracking_type = (TrackingType) G_tracking_type;
367       selectedModeCombo = (IMMModeCombinationType) G_selectedModeCombo;
368
369       pair_id->contact_number = contact_position->contact_number;
370       pair_id->track_number = 0;
371
372       *pair_task = CREATE_NEW_TRACK;
373
374       switch (*contact_type){
375       case RB:
376         *pair_type = RBRB;
377         FATAL_O_VALID((state_update = new StateUpdate(RB_STATE_DIM * 2)) !=
    NULL,                  \
378                       "Agent JVC: No more space to allocate memory!!");
```

```
379             break;
380           case BO:
381             *pair_type = BOBO;
382             FATAL_O_VALID((state_update = new StateUpdate((BO_STATE_DIM * 2)) !=
    NULL),                          \
383                           "Agent JVC: No more space to allocate memory!!!");
384           case XY:
385             *pair_type = XYXY;
386             FATAL_O_VALID((state_update = new StateUpdate((XY_STATE_DIM * 2)) !=
    NULL),                          \
387                           "Agent JVC: No more space to allocate memory!!!");
388             break;
389           default:
390             fprintf (stderr, "In JVC: Invalid track type. \n");
391             exit(0);
392           }
393
394           // Instantiate the new CONTACT_TRACK_PAIR on the BB.
395           InstantiateData(pair, CONTACT_TRACK_PAIR);
396           if ( *tracking_type == IMM_FILTER ) {
397             FATAL_O_VALID((IMM_state_update = new IMMStateUpdate(&selectedModel[index
    ])) != NULL, \
398                           "Agent JVC: No more space to allocate memory!!!");
399             SetValueByData(pair, IMM_STATE_UPDATE, IMM_state_update);
400           }
401           SetValueByData(pair, TRACK_UPDATED, track_updated);
402           SetValueByData(pair, PAIR_TYPE, pair_type);
403           SetValueByData(pair, PAIR_ID, pair_id);
404           SetValueByData(pair, TASK, pair_task);
405           SetValueByData(pair, TRACKING_TYPE, tracking_type);
406           SetValueByData(pair, TRACK_STATE_UPDATE, state_update);
407           pair->PutActOnBB();
408       }
409     } // for(ico=.....
410
411     // Extract assignment pointer from blackboard and get all values.
412
413     (void)ExtractBBData(assign);
414     GetValueByData(assign, ASSOCIATION_TYPE, &association_type);
415     GetValueByData(assign, EXPECTED_NUMBER, &expected_number);
416     GetValueByData(assign, LIST_SIZE, &list_size);
417     GetValueByData(assign, ASSIGNMENT_MATRIX, &assignment_matrix);
418
419     /* Free all memory. */
420     delete association_type;
421     delete expected_number;
422     delete list_size;
423     delete assignment_matrix;
424
425     delete assign;
426
427     /* Free the allocated memory */
428
429   #ifdef DEBUG
430     printf("\nfree allocated memory\n");
431   #endif
432
433     (void) free((float *) cost_matrix);
434     (void) free((int *) first);
435     (void) free((int *) obj);
436     (void) free((int *) X);
437     (void) free((int *) Y);
438     (void) free((float *) U);
439     (void) free((float *) V);
440     m_free(matrix);
441
```



```
442   #ifdef DEBUG
443     printf("\nDeactivate JVCAssignment\n");
444   #endif
445
446     DeActivateAgent(JVCAssignment);
447
448     return;
449
450   }
451
452   IncludeBaseAgent(JVCAssignment, "ASSIGNMENT ASSOCIATION_JVC", "Contact-Track Ass
      ociation with the JVC Algorithm ");
453
454
455
456
457
458
459
460
461
462
463
```

```
1     #include "Lo_Types.h"
2     #include "Lo_IncludeContextFunction.h"
3     #include "PairTask.h"
4     #include "PairType.h"
5     #include "TrackingType.h"
6
7     #define ARGUMENT_TYPE PairUpdate_arguments
8     #define RETURN_TYPE PairUpdate_return_type
9
10    Cardinal PairUpdate(const Cardinal * argument_array, void** values)
11    {
12      Lo_Boolean      track_updated = *((Lo_Boolean *)values[argument_array[0]]);
13      PairTask        pair_task = *((PairTask *)values[argument_array[1]]);
14      PairType        pair_type = *((PairType *)values[argument_array[2]]);
15      TrackingType    tracking_type = *((TrackingType *)values[argument_array[3]]);
16
17      if (!track_updated){
18
19        switch (tracking_type){
20
21        case KALMAN_FILTER:
22          switch(pair_type){
23          case BOBO:
24          case RBBO:
25          case XYBO:
26            return 0;      // EAKF Time update BO track
27          case BORB:
28          case RBRB:
29            return 1;      // EAKF Time update RB track
30          case BOXY:
31          case XYXY:
32            return 2;      // EAKF Time update XY track
33          }
34
35        case IMM_FILTER:
36          switch(pair_type){
37          case BOBO:
38          case RBBO:
39          case XYBO:
40            return 29;     // IMM Time update BO track
41          case BORB:
42          case RBRB:
43            return 30;     // IMM Time update RB track
44          case BOXY:
45          case XYXY:
46            return 31;     // IMM Time update XY track
47          }
48        }
49      }
50
51      switch (pair_task){
52
53      case GATING:
54        switch(pair_type){
55        case BOBO:
56          return 3;      // Gate BO track with BO contact
57        case BORB:
58          return 4;      // Gate RB track with BO contact
59        case BOXY:
60          return 5;      // Gate XY track with BO contact
61        case RBBO:
62          return 6;      // Gate BO track with RB contact
63        case RBRB:
64          return 7;      // Gate RB track with RB contact
65        case XYBO:
66          return 8;      // Gate BO track with XY contact
```



```
67        case XYXY:
68          return 9;      // Gate XY track with XY contact
69        }
70
71      case POS_UPDATE:
72        switch(tracking_type){
73        case KALMAN_FILTER:
74          switch(pair_type){
75          case BOBO:
76            return 10;   // Update BO track state with BO contact using EAKF
77          case BORB:
78            return 11;   // Update RB track state with BO contact using EAKF
79          case BOXY:
80            return 12;   // Update XY track state with BO contact using EAKF
81          case RBBO:
82            return 13;   // Update BO track state with RB contact using EAKF
83          case RBRB:
84            return 14;   // Update RB track state with RB contact using EAKF
85          case XYBO:
86            return 15;   // Update BO track state with XY contact using EAKF
87          case XYXY:
88            return 16;   // Update XY track state with XY contact using EAKF
89          }
90        case IMM_FILTER:
91          switch(pair_type){
92          case BOBO:
93            //return 17;     // Update BO track state with BO contact;
94            return 10;       // IMM NOT SUPPORTED, USES EAKF
95          case BORB:
96            //return 18;     // Update RB track state with BO contact;
97            return 11;       // IMM NOT SUPPORTED, USES EAKF
98          case BOXY:
99            //return 19;     // Update XY track state with BO contact;
100           return 12;       // IMM NOT SUPPORTED, USES EAKF
101         case RBBO:
102           //return 20;     // Update BO track state with RB contact;
103           return 13;       // IMM NOT SUPPORTED, USES EAKF
104         case RBRB:
105           //return 21;     // Update RB track state with RB contact;
106           return 14;       // IMM NOT SUPPORTED, USES EAKF
107         case XYBO:
108           //return 22;     // Update BO track state with XY contact;
109           return 15;       // IMM NOT SUPPORTED, USES EAKF
110         case XYXY:
111           return 23;     // Update XY track state with XY contact using IMMCVCA
112         }
113       }
114
115     case ID_UPDATE:
116       return 24;         // ALL THE CASES;  Update track identity with contact iden
    tity
117
118     case CREATE_NEW_TRACK:
119       switch(pair_type){
120       case BOBO:
121         return 25;       // Create new BO track
122       case RBRB:
123         return 26;       // Create new RB track
124       case XYXY:
125         return 27;       // Create new XY track
126
127       }
128
129     case DELETE_PAIR:
130       return 28;         // Delete pair
131     }
```

```
112
113     return Io_ERROR;                        // Return Io_ERROR which means "do not
    hing".
114   }
115
116   Character *ARGUMENT_TYPE = "TRACK_UPDATED TASK PAIR_TYPE TRACKING_TYPE";
117
118   Character *RETURN_TYPE[] = {"TIME_UPD_BO",          // = 0 ; Time update BO track
119                               "TIME_UPD_RB",          // = 1 ; Time update RB track
120                               "TIME_UPD_XY",          // = 2 ; Time update XY track
121                               "ASSIGN_BO_BO",         // = 3 ; Gate BO track with B
    O contact
122                               "ASSIGN_BO_RB",         // = 4 , Gate RB track with B
    O contact
123                               "ASSIGN_BO_XY",         // = 5 , Gate XY track with B
    O contact
124                               "ASSIGN_RB_BO",         // = 6 ; Gate BO track with R
    B contact
125                               "ASSIGN_RB_RB",         // = 7 ; Gate RB track with R
    B contact
126                               "ASSIGN_XY_BO",         // = 8 ; Gate BO track with X
    Y contact
127                               "ASSIGN_XY_XY",         // = 9 ; Gate XY track with X
    Y contact
128                               "EAKF_BO_BO",           // = 10; Update BO track stat
    e with BO contact using EAKF
129                               "EAKF_BO_RB",           // = 11; Update RB track stat
    e with BO contact using EAKF
130                               "EAKF_BO_XY",           // = 12; Update XY track stat
    e with BO contact using EAKF
131                               "EAKF_RB_BO",           // = 13; Update BO track stat
    e with RB contact using EAKF
132                               "EAKF_RB_RB",           // = 14, Update RB track stat
    e with RB contact using EAKF
133                               "EAKF_XY_BO",           // = 15; Update BO track stat
    e with XY contact using EAKF
134                               "EAKF_XY_XY",           // = 16, Update XY track stat
    e with XY contact using EAKF
135                               "IMMCVCA_BO_BO",        // = 17; Update BO track stat
    e with BO contact using IMMCVCA
136                               "IMMCVCA_BO_RB",        // = 18; Update RB track stat
    e with BO contact using IMMCVCA
137                               "IMMCVCA_BO_XY",        // = 19; Update XY track stat
    e with BO contact using IMMCVCA
138                               "IMMCVCA_RB_BO",        // = 20, Update BO track stat
    e with RB contact using IMMCVCA
139                               "IMMCVCA_RB_RB",        // = 21; Update RB track stat
    e with RB contact using IMMCVCA
140                               "IMMCVCA_XY_BO",        // = 22; Update BO track stat
    e with XY contact using IMMCVCA
141                               "IMM_XY_XY",            // = 23, Update XY track stat
    e with XY contact using IMM
142                               "PROPOSITION",          // = 24; Update track identit
    y with contact identity
143                               "NEW_BO_TRACK",         // = 25; Create new BO track
144                               "NEW_RB_TRACK",         // = 26; Create new RB track
145                               "NEW_XY_TRACK",         // = 27; Create new XY track
146                               "NULL",                 // = 28; Delete pair
147                               "IMM_TIME_UPD_BO",      // = 29, Time update BO track
     using IMM
148                               "IMM_TIME_UPD_RB",      // = 30, Time update RB track
     using IMM
149                               "IMM_TIME_UPD_XY"};     // = 31, Time update XY track
     using IMM
150
151   IncludeContextFunction(PairUpdate, ARGUMENT_TYPE, RETURN_TYPE,
```



```
172       "This context function selects the agent to be activated based on the track_
     updated status, the task to be performed and the pair type of the contact/track
     pair" ),
173
174
175
```

```
1     //---{{{  Begin IMMModeCombinationType.h
2
3     #ifndef          __IMMModeCombinationType__                    // __IMMModeComb
      inationType__
4     #define          __IMMModeCombinationType__
5
6     #include "IoTypes.h"
7
8     // To allow a range of models for the modes
9
10    enum IMMModeCombinationType
11    {
12      CVCA                  // = 0;  One constant velocity model, one constant accele
      ration model
13    };
14
15    #endif                                                          // ? __IMMModeCo
      mbinationType__
16
17    //---}}}  End IMMModeCombinationType.h
18
```

```
1     //---{{{  Begin IMMRoutines.h
2
3     #ifndef          __IMMRoutines__                                  // _ _IMMRoutines
      __
4     #define          __IMMRoutines__
5
6     #include "IoTypes.h"
7     #include "matrix.h"
8     #include "matrix2.h"
9
10    MAT    *mixmu_calc(u_int nm,MAT *cbar,MAT *modemu,MAT *TransPr,MAT *mixmu),
11    MAT    *modeP_calc(u_int nx,u_int nm,MAT *modex,MAT *modex0,MAT *mixmu,MAT *modeP
      );
12    MAT    *col_to_m(MAT *mat1,u_int col,MAT *out);
13    MAT    *m_to_col(MAT *mat1,MAT *out,u_int col);
14    MAT    *quadtr(MAT *F,MAT *P,MAT *FPFtr);
15    VEC    *x_calc(u_int nx,MAT *modex,MAT *modemu,VEC *x);
16    MAT    *P_calc(u_int nx,u_int nm,MAT *modex,MAT *modeP,MAT *modemu,VEC *x,MAT *P)
      ;
17    MAT    *likeli_calc(u_int nz,u_int i,double cbari,MAT *S,MAT *Sinv,VEC *nu,MAT *c
      i);
18    MAT    *modemu_calc(u_int nm,MAT *c,MAT *modumu);
19
20    #endif                                                           // ? _ _IMMRoutin
      es__
21
22    //---}}}  End IMMRoutines.h
```

```
#include <math.h>
#include "IMMRoutines.h"

/* MATRIX MANIPULATION ROUTINES */

/* compute the determinant of a (square) matrix */
double det(MAT *A)
{
  MAT  *LU;
  PERM *pivot;
  u_int j;
  u_int n = A->m;
  double d;

  if ( A==(MAT *)NULL )
    error(E_NULL,"det");
  if ( A->m != A->n )
    error(E_SIZES,"det");
  pivot = px_get(n);

  LU = m_get(n,n);
  LU = m_copy(A,LU);
  LUfactor(LU,pivot);
  d = (double)px_sign(pivot);
  for ( j=0; j<n; j++ )
    d *= LU->me[j][j];

  m_free(LU);
  px_free(pivot);
  return(d);
}

/* computes the element-wise product of two matrices of same size
   note: the inclusion of the output as a parameter is to put
   allocation and freeing outside the function */
MAT *m_eltmlt(MAT *mat1,MAT *mat2,MAT *out)
{
  u_int m,n,i,j;

  if ( mat1==(MAT *)NULL || mat2==(MAT *)NULL )
    error(E_NULL,"m_eltmlt");
  if ( mat1->m != mat2->m || mat1->n != mat2->n )
    error(E_SIZES,"m_eltmlt");
  if ( out->m != mat1->m || out->n != mat1->n )
    error(E_SIZES,"m_eltmlt");
  m = mat1->m;  n = mat1->n;

  for ( i=0; i<m; i++ )
  {
    for ( j=0; j<n; j++ )
      out->me[i][j] = mat1->me[i][j]*mat2->me[i][j];
  }

  return (out);
}

/* computes the reciprocal of each element in a matrix */
MAT *m_eltinv(MAT *mat1,MAT *out)
{
  u_int m,n,i,j;

  if ( mat1==(MAT *)NULL )
    error(E_NULL,"m_eltinv");
  if ( out->m != mat1->m || out->n != mat1->n )
    error(E_SIZES,"m_eltinv");
  m = mat1->m;  n = mat1->n;
```



```

  for ( i=0; i<m; i++ )
  {
    for ( j=0; j<n; j++ )
      if ( mat1->me[i][j] != 0 )
        out->me[i][j] = 1/(mat1->me[i][j]);
  }

  return (out);
}

/* computes the v1*v2tr product of two vectors */
MAT *vvtr_mlt(VEC *v1,VEC *v2,MAT *out)
{
  MAT *v1mat, *v2mat;

  if ( v1==(VEC *)NULL || v2==(VEC *)NULL )
    error(E_NULL,"vvtr_mlt");
  if ( out->m != v1->dim || out->n != v2->dim )
    error(E_SIZES,"vvtr_mlt");
  v1mat = m_get(v1->dim,1);
  v2mat = m_get(1,v2->dim);

  set_col(v1mat,0,v1,0);
  set_row(v2mat,0,v2,0);
  m_mlt(v1mat,v2mat,out);

  m_free(v1mat); m_free(v2mat);
  return(out);
}

/* puts a matrix into the jth column of another; column 0 is
   above column 1, which is followed by column 2, etc */
MAT *m_to_col(MAT *mat1,MAT *out,u_int col)
{
  u_int m,n,i,j;
  VEC *dummy;

  if ( mat1==(MAT *)NULL || out==(MAT *)NULL )
    error(E_NULL,"m_to_col");
  m = mat1->m;  n = mat1->n;
  if ( (m*n) > out->m )
    error(E_SIZES,"m_to_col");
  if ( col >= out->n )
    error(E_SIZES,"m_to_col");

  dummy = v_get(out->m);
  v_zero(dummy);
  get_col(out,col,dummy,0);

  for ( j=0, i=0; j<n; j++ )
  {
    for ( i=0; i<m; i++ )
      dummy->ve[m*j+i] = mat1->me[i][j];
  }

  v_free(dummy);
  return(out);
}

/* puts the jth column of a matrix into another matrix containing
   a lesser number of rows; the size of out is pre-determined */
MAT *col_to_m(MAT *mat1,u_int col,MAT *out)
{
  u_int m,n,i,j;

```

```
    if ( mat1==(MAT *)NULL || out==(MAT *)NULL )
      error(E_NULL,"col_to_m");
    m = out->m;   n = out->n;
    if ( (m*n) > mat1->m )
      error(E_SIZES,"col_to_m");
    if ( col >= mat1->n )
      error(E_SIZES,"col_to_m");
    for ( j=0; j<n; j++ )
    {
      for ( i=0; i<m; i++ )
        out->me[i][j] = mat1->me[m*j+i][col];
    }

    return(out);
}

/* returns the sum of the elements in a row or col of a matrix,
   rows is dim 0, cols is dim 1, j is the particular row or col */
double m_eltsum(MAT *A,u_int dim,u_int j)
{
  u_int i;
  double sum = 0.0;

  if ( dim==0 )
    for ( i=0; i<(A->n); i++ ) sum += A->me[j][i];
  else if ( dim==1 )
    for ( i=0; i<(A->m); i++ ) sum += A->me[i][j];
  else
    error(E_SIZES,"m_eltsum");

  return(sum);
}

MAT *quadtr(MAT *F,MAT *P,MAT *FPFtr)
{
  MAT *dummy;
  dummy = m_get(P->m,F->m);

  mmtr_mlt(P,F,dummy);
  m_mlt(F,dummy,FPFtr);

  m_free(dummy);
  return(FPFtr);
}

/* END OF MATRIX MANIPULATION ROUTINES */

/* IMM ROUTINES */

MAT   *mixmu_calc(u_int nm,MAT *cbar,MAT *modemu,MAT *TransPr,MAT *mixmu)
{
  MAT  cbarinv, *dummy;
  cbarinv = m_get(nm,1);
  dummy   = m_get(nm,nm);

  m_eltinv(cbar,cbarinv);
  mmtr_mlt(modemu,cbarinv,dummy);
  m_eltmlt(TransPr,dummy,mixmu);          /* mixing probs          */

  m_free(cbarinv); m_free(dummy);
  return(mixmu);
}

/* calculates the prior filter covariances */
MAT   *modeP_calc(u_int nx,u_int nm,MAT *modex,MAT *modex0,MAT *mixmu,MAT *modeP
)
```



```
{
  u_int nx2 = nx*nx;   /* square of state dim    */
  u_int i;
  MAT *PP, *modePP, *dummy;
  VEC *xk1, *modex1, *modex01;

  PP      = m_get(nx,nx);       xk1 = v_get(nx);
  modex1 = v_get(nx);           modex01 = v_get(nx);
  modePP = m_get(nx2,nm);       dummy  = m_get(nx2,nm);

  for ( i=0; i<nm; i++ )
  {
    get_col(modex,i,modex1);  get_col(modex0,i,modex01);
    v_sub(modex1,modex01,xk1);
    vvtr_mlt(xk1,xk1,PP);
    m_to_col(PP,modePP,i);
  }

  m_add(modeP,modePP,dummy);
  m_mlt(dummy,mixmu,modeP);

  m_free(PP);  v_free(modex1);  v_free(modex01);
  v_free(xk1); m_free(modePP);  m_free(dummy);
  return(modeP);
}

/* likelihood calculations */
MAT *likeli_calc(u_int nz,u_int i,double cbari,MAT *S,MAT *Sinv,VEC *nu,MAT *c)
{
  VEC *dummyvec;
  MAT *dummy;
  dummy    = m_get(nz,nz);
  dummyvec = v_get(nz);

  c->me[0][i] = cbari/sqrt(fabs(det(sm_mlt(2*M_PI,S,dummy))));
  c->me[1][i] = in_prod(nu,mv_mlt(Sinv,nu,dummyvec));

  m_free(dummy);
  v_free(dummyvec);
  return(c);
}

/* calculates the updated mode probabilities */
MAT *modemu_calc(u_int nm,MAT *c,MAT *modemu)
{
  u_int  i,j;
  double the_sum;
  MAT    *dummy;

  for ( i=0; i<nm; i++ )
  {
    the_sum = 0.0;
    for ( j=0; j<nm; j++ )
      if ( j != i )
        the_sum += (c->me[0][j])*exp(-0.5*(c->me[1][j]-c->me[1][i]));
    modemu->me[i][0] = 1/(1+the_sum/(c->me[0][i]));
  }

  dummy = m_get(modemu->m,modemu->n);
  the_sum = m_eltsum(modemu,1,0);
  sm_mlt(1/the_sum,modemu,dummy);
  m_copy(dummy,modemu);

  m_free(dummy);
  return(modemu);
}
```

```
264
265   /* calculates the combined state as a vector */
266   VEC   *x_calc(u_int nx,MAT *modex,MAT *modemu,VEC *x)
267   {
268     MAT *xmat;
269     xmat = m_get(nx,1);
270
271     m_mlt(modex,modemu,xmat);
272     get_col(xmat,0,x);  /* change to vector */
273
274     m_free(xmat);
275     return(x);
276   }
277
278   /* calculates the combined covariance */
279   MAT   *P_calc(u_int nx,u_int nm,MAT *modex,MAT *modeP,MAT *modemu,VEC *x,MAT *P)
280   {
281     u_int nx2 = nx*nx;   /* square of state dim */
282     u_int i;
283     MAT *PP, *modePP, *dummy, *vecP;
284     VEC *xk1, *modexi;
285
286     PP     = m_get(nx,nx);      xk1    = v_get(nx);
287     modexi = v_get(nx);
288     modePP = m_get(nx2,nm);     dummy = m_get(nx2,nm);
289     vecP   = m_get(nx2,1);
290
291     for ( i=0; i<nm; i++ )
292     {    ;
293       get_col(modex,i,modexi);
294       v_sub(modexi,x,xk1);
295       vvtr_mlt(xk1,xk1,PP);
296       m_to_col(PP,modePP,i);
297     }
298
299     m_add(modeP,modePP,dummy);
300     m_mlt(dummy,modemu,vecP);
301     col_to_m(vecP,0,P);
302
303     m_free(PP);   v_free(modexi);  m_free(vecP);
304     v_free(xk1);  m_free(modePP);  m_free(dummy);
305
306     return(P);
307   }
308
309   /* END OF IMM ROUTINES */
```

```
1     //---{{{  Begin IMMStateUpdate.h
2
3     #ifndef          __IMMStateUpdate__                                    // _ IMM
      StateUpdate__
4     #define          __IMMStateUpdate__
5
6     #include "LoTypes.h"
7     #include "matrix.h"
8     #include "matrix2.h"
9     #include "IMMModeCombinationType.h"
10
11    // The mode matrices are as follows:
12    //
13    // In the matrix modex the first column is the the first mode state vector, the
      second
14    // column is the second mode state vector, etc.
15    //
16    // The mode covariance matrices are first converted into tall vectors such that
      the
17    // first column is above the second column, which is above the third, etc.  Then
       these
18    // tall vectors are combined into a single matrix modeP where the first colum i
      s the
19    // first covariance matrix, the second column is the second covariance matrix, e
      tc.
20
21    class  IMMStateUpdate
22    {
23      public;
24        IMMStateUpdate(IMMModeCombinationType selectedModeCombo);
25        ~IMMStateUpdate();
26
27        MAT        *modex;
28        MAT        *modeP;
29        MAT        *cbar;
30    };
31
32    #endif                                                                // ? __IMMStateU
      pdate__
33
34    //---}}}  End IMMStateUpdate.h
```

```
1     #include "LoError.h"
2     #include "LoExternParameters.h"
3     #include "LoTypes.h"
4     #include "IMMStateUpdate.h"
5
6     IMMStateUpdate::IMMStateUpdate(IMMModeCombinationType selectedModeCombo)
7     {
8       Cardinal nx;  // dimension of state vector
9       Cardinal nm;  // number of modes
10
11      switch(selectedModeCombo) {
12
13      case CVCA:
14        nx     = 6;
15        nm     = 2;
16        modex = m_get(nx, nm);
17        modeP = m_get(nx*nx, nm);
18        cbar  = m_get(nm, 1);
19        break;
20
21      default:
22        cerr << "IMMStateUpdate: CVCA is currently the only mode combination type \n
    ";
23      }
24
25      return;
26    }
27
28    IMMStateUpdate::~IMMStateUpdate()
29    {
30      m_free(modex);
31      m_free(modeP);
32      m_free(cbar);
33
34      return;
35    }
36
```

```
1   //---{{{  Begin IMMTrackState.h
2
3   #ifndef          __IMMTrackState__                          // _IMMTrackSta
    te__
4   #define          __IMMTrackState__
5
6   #include "LoTypes.h"
7   #include "matrix.h"
8   #include "matrix2.h"
9   #include "IMMModeCombinationType.h"
10
11  // The mode matrices are as follows:
12  //
13  // In the matrix modex the first column is the the first mode state vector, the
    second
14  // column is the second mode state vector, etc.
15  //
16  // The mode covariance matrices are first converted into tall vectors such that
    the
17  // first column is above the second column, which is above the third, etc.  Then
     these
18  // tall vectors are combined into a single matrix modeP where the first column i
    s the
19  // first covariance matrix, the second column is the second covariance matrix, e
    tc.
20  //
21  // The mode probability matrix modemu is of vector form with the first mode prob
    ability
22  // above the second, the second above the third, etc
23  //       /
24  // The transition probability flow matrix is found, like the selectedModeCombo,
    in the
25  // Parameter_File (ie. it is set by the user)
26
27  class IMMTrackState
28  {
29    public:
30      IMMTrackState(IMMModeCombinationType selectedModeCombo);
31      ~IMMTrackState();
32
33      IMMModeCombinationType  modeCombo;
34      MAT                     *modex;          // Mode state vectors
35      MAT                     *modeP;          // Mode covariance matrices
36      MAT                     *modemu;         // Mode probability matrix
37  };
38
39  #endif                                                    // ? __IMMTrack
    tate__
40
41  //---}}}  End IMMTrackState.h
```

```
1     #include "LoError.h"
2     #include "LoExternParameters.h"
3     #include "LoTypes.h"
4     #include "IMMTrackState.h"
5
6     IMMTrackState::IMMTrackState(IMMModeCombinationType selectedModeCombo)
7     {
8       u_int nx;  // dimension of state vector
9       u_int nm;  // number of modes
10
11      switch(selectedModeCombo) {
12
13      case CVCA:
14        nx         = 6;
15        nm         = 2;
16        modex      = m_get(nx, nm);
17        modeP      = m_get(nx*nx, nm);
18        modemu     = m_get(nm, 1);
19        modeCombo = selectedModeCombo;
20        break;
21
22      default:
23        cerr << "IMMTrackState: CVCA is currently the only mode combination type.\n"
    ;
24      }
25
26      return;
27    }
28
29    IMMTrackState::~IMMTrackState()
30    {
31      m_free(modex);
32      m_free(modeP);
33      m_free(modemu);
34
35      return;
36    }
37
```

```
1     //---{{{  Begin IMM_CVCA.h
2
3     #ifndef          __IMM_CVCA__                                    // . IMM_CVCA _
4     #define          __IMM_CVCA__
5
6     #include "LoTypes.h"
7     #include "matrix.h"
8     #include "matrix2.h"
9     #include "StateUpdate.h"
10    #include "TrackState.h"
11    #include "ContactPosition.h"
12    #include "OwnshipData.h"
13    #include "IMMModeCombinationType.h"
14    #include "IMMTrackState.h"
15    #include "IMMStateUpdate.h"
16
17    void IMM_CVCA_TimeUpdate(StateUpdate *state_update, IMMTrackState *IMM_track_sta
      te, IMMStateUpdate *IMM_state_update, double dt, double q, OwnshipData *ownship_
      ate);
18    void IMM_CVCA_CreateXYTrack(TrackState *track_state, IMMTrackState *IMM_track_st
      ate);
19    void IMM_CVCA_InitFiltTrack(TrackState *state_upd, IMMTrackState *IMM_state_upd)
      ;
20    void IMM_CVCA_FiltTrack(IMMStateUpdate *IMM_state_update, ContactPosition *conta
      ct_position, IMMTrackState *IMM_state_upd, TrackState *state_upd);
21
22    #endif                                                          // ? _. IMM_CVCA_
23
24    //---}}}  End IMM_CVCA.h
```

```
1     #include <stdio.h>
2     #include <stdlib.h>
3     #include "LoError.h"
4     #include "LoExternParameters.h"
5     #include "LoTypes.h"
6     #include "IMM_CVCA.h"
7     #include "IMMRoutines.h"
8     #include "IMMModeCombinationType.h"
9
10    /**** Time Update for a CONTACT_TRACK_PAIR using IMMCVCA ****/
11    void IMM_CVCA_TimeUpdate(StateUpdate *state_update, IMMTrackState *IMM_track_sta
      te, IMMStateUpdate *IMM_state_update, double dt, double q, OwnshipData *ownship_
      ata)
12    {
13      Cardinal      nx = 6;  // State dimension, since CVCA
14      Cardinal      nm = 2;  // Number of Modes, since CVCA
15      Cardinal      i;
16      double        dt2 = dt*dt;
17      MAT           *TransPr, *mixmu;
18      MAT           *modemu = IMM_track_state->modemu;  // mode probabilities
19      MAT           *cbar   = IMM_state_update->cbar;   // probability normalizing co
      nstants
20      MAT           *modex  = IMM_track_state->modex;   // before mixing track state
21      MAT           *modex0 = IMM_state_update->modex;  // after mixing track state
22      MAT           *modeP  = IMM_state_update->modeP;  // before mixing covariance
23      MAT           *Q[2], *F[2], *P, *FPFtr;
24      VEC           *x, *xp;
25      double        q_02, q_22, q_04, q_24, q_44, q_CV, q_CA;
26
27      // The mode probability flow matrix.
28      double a11 = G_mode_prob_flow_a11;  double a12 = G_mode_prob_flow_a12;
29      double a21 = G_mode_prob_flow_a21;  double a22 = G_mode_prob_flow_a22;
30
31      // The mode transition matrix
32      TransPr = m_get(nm,nm);
33      TransPr->me[0][0] = 1 + a11*dt;  TransPr->me[0][1] = a12*dt;
34      TransPr->me[1][0] = a21*dt;      TransPr->me[1][1] = 1 + a22*dt;
35
36      if ( TransPr->me[0][0] < 0 ) {
37        TransPr->me[0][0] = 0;         TransPr->me[0][1] = 1;
38      }
39      if ( TransPr->me[1][1] < 0 ) {
40        TransPr->me[1][0] = 1;         TransPr->me[1][1] = 0;
41      }
42
43      // Calculation of mixing probabilities
44      mixmu = m_get(nm,nm);
45
46      mtrm_mlt(TransPr,modemu,cbar);                  // normalizing constants
47      mixmu_calc(nm,cbar,modemu,TransPr,mixmu);       // mixing probabilities
48      m_free(TransPr);
49
50      // Mixing
51      m_mlt(modex,mixmu,modex0);                      // modex0 becomes the after mixin
      g track state
52      modeP_calc(nx,nm,modex,modex0,mixmu,modeP);     // modeP becomes the after mixing
       coverlance
53      m_free(mixmu);
54
55      // Set the CV process noise coverlance matrix
56      q_02 = 2.0*q/dt;   q_22 = 2.0*q_02/dt;
57
58      Q[0] = m_get(nx,nx);       q_CV = G_process_noise_cov_CV;
59      Q[0]->me[0][0] = q_CV*q;                Q[0]->me[0][2] = q_CV*q_02;
60      Q[0]->me[1][1] = q_CV*q;                Q[0]->me[1][3] = q_CV*q_02;
61      Q[0]->me[2][0] = q_CV*q_02;             Q[0]->me[2][2] = q_CV*q_22;
```



```
62      Q[0]->me[3][1] = q_CV*q_02;             Q[0]->me[3][3] = q_CV*q_22;
63
64      // Set the CA process noise coverlance matrix
65      q_04  = q_02/dt;    q_24 = q_22/dt;
66      q_44  = q_24/dt;
67
68      Q[1] = m_get(nx,nx);       q_CA = G_process_noise_cov_CA;
69      Q[1]->me[0][0] = q_CA*q;     Q[1]->me[0][2] = q_CA*q_02;  Q[1]->me[0][4] = q_C
      A*q_04;
70      Q[1]->me[1][1] = q_CA*q;     Q[1]->me[1][3] = q_CA*q_02;  Q[1]->me[1][5] = q_C
      A*q_04;
71      Q[1]->me[2][0] = q_CA*q_02;  Q[1]->me[2][2] = q_CA*q_22;  Q[1]->me[2][4] = q_C
      A*q_24;
72      Q[1]->me[3][1] = q_CA*q_02;  Q[1]->me[3][3] = q_CA*q_22;  Q[1]->me[3][5] = q_C
      A*q_24;
73      Q[1]->me[4][0] = q_CA*q_04;  Q[1]->me[4][2] = q_CA*q_24;  Q[1]->me[4][4] = q_C
      A*q_44;
74      Q[1]->me[5][1] = q_CA*q_04;  Q[1]->me[5][3] = q_CA*q_24;  Q[1]->me[5][5] = q_C
      A*q_44;
75
76      // Set the state transition matrices
77      F[0] = m_get(nx,nx);
78      F[0]->me[0][0] = 1;  F[0]->me[0][2] = dt;
79      F[0]->me[1][1] = 1;  F[0]->me[1][3] = dt;
80                           F[0]->me[2][2] = 1;
81                           F[0]->me[3][3] = 1;
82
83      F[1] = m_get(nx,nx);
84      F[1]->me[0][0] = 1;  F[1]->me[0][2] = dt;  F[1]->me[0][4] = dt2/2;
85      F[1]->me[1][1] = 1;  F[1]->me[1][3] = dt;  F[1]->me[1][5] = dt2/2;
86                           F[1]->me[2][2] = 1;   F[1]->me[2][4] = dt;
87                           F[1]->me[3][3] = 1;   F[1]->me[3][5] = dt;
88                                                 F[1]->me[4][4] = 1;
89                                                 F[1]->me[5][5] = 1;
90
91      // Mode-matched prediction
92      x = v_get(nx);          xp    = v_get(nx);
93      P = m_get(nx,nx);       FPFtr = m_get(nx,nx);
94
95      for ( i=0; i<nm; i++ ) {
96        get_col(modex0,i,x);
97        col_to_m(modeP,i,P);
98
99        // State prediction
100       mv_mlt(F[i],x,xp);
101       if ((G_target_pos_coord == XY_REL_OWNS) && (G_target_vel_coord == VXVY_ABS))
      {
102         // Subtract Ownship motion from the position update
103         xp->ve[0] -= ownship_data->x_vel * dt;
104         xp->ve[1] -= ownship_data->y_vel * dt;
105       }
106       set_col(modex0,i,xp,0);
107
108       // Covariance prediction
109       m_add(quadtr(F[i],P,FPFtr),Q[i],P);
110       m_free(F[i]);   m_free(Q[i]);
111       m_to_col(P,modeP,i);
112     }
113
114     v_free(xp);   m_free(FPFtr);
115
116     // Combination for prediction
117     x_calc(nx,modex0,modemu,x);
118     P_calc(nx,nm,modex0,modeP,modemu,x,P);
119
120     // Put into state_update
```

```
121     state_update->state_vec->me[0][0] = x->ve[0];
122     state_update->state_vec->me[1][0] = x->ve[1];
123     state_update->state_vec->me[2][0] = x->ve[2];
124     state_update->state_vec->me[3][0] = x->ve[3];
125
126     state_update->cov_mat->me[0][0] = P->me[0][0];
127     state_update->cov_mat->me[0][1] = P->me[0][1];
128     state_update->cov_mat->me[0][2] = P->me[0][2];
129     state_update->cov_mat->me[0][3] = P->me[0][3];
130
131     state_update->cov_mat->me[1][0] = P->me[1][0];
132     state_update->cov_mat->me[1][1] = P->me[1][1];
133     state_update->cov_mat->me[1][2] = P->me[1][2];
134     state_update->cov_mat->me[1][3] = P->me[1][3];
135
136     state_update->cov_mat->me[2][0] = P->me[2][0];
137     state_update->cov_mat->me[2][1] = P->me[2][1];
138     state_update->cov_mat->me[2][2] = P->me[2][2];
139     state_update->cov_mat->me[2][3] = P->me[2][3];
140
141     state_update->cov_mat->me[3][0] = P->me[3][0];
142     state_update->cov_mat->me[3][1] = P->me[3][1];
143     state_update->cov_mat->me[3][2] = P->me[3][2];
144     state_update->cov_mat->me[3][3] = P->me[3][3];
145
146     v_free(x);   m_free(P);
147
148     return;
149   }
150
151   /**** IMM_TRACK_STATE setup for TRACK creation. ****/
153   void IMM_CVCA_CreateXYTrack(TrackState *track_state, IMMTrackState *IMM_track_st
      ate)
153   {
154     u_int i;
155
156     for ( i=0; i<2; i++ ) {
157       // Set the state
158       IMM_track_state->modex->me[0][i] = track_state->state_vec->me[0][0];
159       IMM_track_state->modex->me[1][i] = track_state->state_vec->me[1][0];
160       IMM_track_state->modex->me[2][i] = track_state->state_vec->me[2][0];
161       IMM_track_state->modex->me[3][i] = track_state->state_vec->me[3][0];
162
163       // Set the covariance
164       IMM_track_state->modeP->me[0][i]  = track_state->cov_mat->me[0][0];
165       IMM_track_state->modeP->me[1][i]  = track_state->cov_mat->me[1][0];
166       IMM_track_state->modeP->me[6][i]  = track_state->cov_mat->me[0][1];
167       IMM_track_state->modeP->me[7][i]  = track_state->cov_mat->me[1][1];
168       IMM_track_state->modeP->me[14][i] = track_state->cov_mat->me[2][2];
169       IMM_track_state->modeP->me[21][i] = track_state->cov_mat->me[3][3];
170     }
171     // Set the acceleration part of the CA covariance
172     IMM_track_state->modeP->me[28][i] = G_default_sigma_axyair * G_default_sigma_a
      xyair;
173     IMM_track_state->modeP->me[35][i] = G_default_sigma_axyair * G_default_sigma_a
      xyair;
174
175     IMM_track_state->modemu->me[0][0] = G_default_CV_mode_prob;
176     IMM_track_state->modemu->me[1][0] = G_default_CA_mode_prob;
177
178     return;
179   }
180
181   /**** IMM_TRACK_STATE setup for an initialized TRACK  ****/
182   void IMM_CVCA_InitFiltTrack(TrackState *state_upd, IMMTrackState *IMM_state_upd)
183   {
```



```
184     u_int i;
185
186   #ifdef DEBUG
187     FILE *f;
188   #endif
189
190     for ( i=0; i<2; i++ ) {
191       // Set the state
192       IMM_state_upd->modex->me[0][i] = state_upd->state_vec->me[0][0];
193       IMM_state_upd->modex->me[1][i] = state_upd->state_vec->me[1][0];
194       IMM_state_upd->modex->me[2][i] = state_upd->state_vec->me[2][0];
195       IMM_state_upd->modex->me[3][i] = state_upd->state_vec->me[3][0];
196
197       // Set the covariance
198       IMM_state_upd->modeP->me[0][i]  = state_upd->cov_mat->me[0][0];
199       IMM_state_upd->modeP->me[1][i]  = state_upd->cov_mat->me[1][0];
200       IMM_state_upd->modeP->me[2][i]  = state_upd->cov_mat->me[2][0];
201       IMM_state_upd->modeP->me[3][i]  = state_upd->cov_mat->me[3][0];
202
203       IMM_state_upd->modeP->me[6][i]  = state_upd->cov_mat->me[0][1];
204       IMM_state_upd->modeP->me[7][i]  = state_upd->cov_mat->me[1][1];
205       IMM_state_upd->modeP->me[8][i]  = state_upd->cov_mat->me[2][1];
206       IMM_state_upd->modeP->me[9][i]  = state_upd->cov_mat->me[3][1];
207
208       IMM_state_upd->modeP->me[12][i] = state_upd->cov_mat->me[0][2];
209       IMM_state_upd->modeP->me[13][i] = state_upd->cov_mat->me[1][2];
210       IMM_state_upd->modeP->me[14][i] = state_upd->cov_mat->me[2][2];
211       IMM_state_upd->modeP->me[15][i] = state_upd->cov_mat->me[3][2];
212
213       IMM_state_upd->modeP->me[18][i] = state_upd->cov_mat->me[0][3];
214       IMM_state_upd->modeP->me[19][i] = state_upd->cov_mat->me[1][3];
215       IMM_state_upd->modeP->me[20][i] = state_upd->cov_mat->me[2][3];
216       IMM_state_upd->modeP->me[21][i] = state_upd->cov_mat->me[3][3];
217     }
218     // Set the acceleration part of the CA covariance
219     IMM_state_upd->modeP->me[28][i] = G_default_sigma_axyair * G_default_sigma_axy
      air;
220     IMM_state_upd->modeP->me[35][i] = G_default_sigma_axyair * G_default_sigma_axy
      air;
221
222     IMM_state_upd->modemu->me[0][0] = G_default_CV_mode_prob;
223     IMM_state_upd->modemu->me[1][0] = G_default_CA_mode_prob;
224
225   #ifdef DEBUG
226     if (( f = fopen("/home/bjarry/AIRBORNE/test/tf.out","a")) == NULL)
227     {
228       fprintf(stderr,"Cannot open output file.\n");
229       exit(1);
230     }
231
232     fprintf(f,"\nIMM_CVCA_InitFiltTrack: track %d\n", state_upd->track_number);
233     fprintf(f,"IMM state upd->modex\n");  m_foutput(f, IMM_state_upd->modex);
234     fprintf(f,"IMM state upd->modemu\n"); m_foutput(f, IMM_state_upd->modemu);
235
236     fclose(f);
237   #endif
238
239     return;
240   }
241
242   void IMM_CVCA_FiltTrack(IMMStateUpdate *IMM_state_update, ContactPosition *conta
      ct_position, IMMTrackState *IMM_state_upd, TrackState *state_upd)
243   {
244     Cardinal  i;
245     Cardinal  nx = 6,                                  // State dimension, since CV
      CA
```

```
246     Cardinal  nm = 2;                                    // Number of modes, since CV
    CA
247     Cardinal  nz = 2;                                    // Measurement dimension
248     MAT       *cbar     = IMM_state_update->cbar;        // probability normalizing c
    onstants
249     MAT       *modex0   = IMM_state_update->modex;       // k+1|k track states
250     MAT       *modeP0   = IMM_state_update->modeP;       // k+1|k covariances
251     MAT       *modex    = IMM_state_upd->modex;          // k+1|k+1 track states
252     MAT       *modeP    = IMM_state_upd->modeP;          // k+1|k+1 covariances
253     MAT       *modemu   = IMM_state_upd->modemu;         // k+1 mode probabilities
254     MAT       *z        = contact_position->state_vec;   // measurement
255     MAT       *R        = contact_position->cov_mat;     // measurement noise covaria
    nce
256
257     MAT       *P, *S, *Sinv, *W, *Pu, *I_WH, *KPKtr, *WRWtr, *t,
258     VEC       *z, *x, *xu, *nu;
259
260   #ifdef DEBUG
261     FILE      *f;
262   #endif
263
264     // Set up the measurement vector
265     z = v_get(nz);
266     get_col(Z,0,z);
267
268     // Mode-matched filtering
269     x     = v_get(nx);      xu    = v_get(nx);       nu    = v_get(nz),
270     P     = m_get(nx,nx);   S     = m_get(nz,nz);    Sinv  = m_get(nz,nz);
271     W     = m_get(nx,nz);   Pu    = m_get(nx,nx);
272     I_WH = m_get(nx,nx);   KPKtr = m_get(nx,nx);    WRWtr = m_get(nx,nx);
273     c     = m_get(2,nm);
274
275     for ( i=0; i<nm; i++ ) {
276       get_col(modex0,i,x);
277       col_to_m(modeP0,i,P);
278
279       // Form the innovation and its covariance
280       nu->ve[0] = z->ve[0] - x->ve[0];
281       nu->ve[1] = z->ve[1] - x->ve[1];
282
283       S->me[0][0] = P->me[0][0] + R->me[0][0];     S->me[0][1] = P->me[0][1] + R->m
    e[0][1];
284       S->me[1][0] = P->me[1][0] + R->me[1][0];     S->me[1][1] = P->me[1][1] + R->m
    e[1][1];
285
286       // Form the Kalman gain matrix
287       m_inverse(S,Sinv);
288
289       W->me[0][0] = P->me[0][0]*Sinv->me[0][0] + P->me[0][1]*Sinv->me[1][0],
290       W->me[1][0] = P->me[1][0]*Sinv->me[0][0] + P->me[1][1]*Sinv->me[1][0],
291       W->me[2][0] = P->me[2][0]*Sinv->me[0][0] + P->me[2][1]*Sinv->me[1][0],
292       W->me[3][0] = P->me[3][0]*Sinv->me[0][0] + P->me[3][1]*Sinv->me[1][0];
293
294       W->me[0][1] = P->me[0][0]*Sinv->me[0][1] + P->me[0][1]*Sinv->me[1][1],
295       W->me[1][1] = P->me[1][0]*Sinv->me[0][1] + P->me[1][1]*Sinv->me[1][1],
296       W->me[2][1] = P->me[2][0]*Sinv->me[0][1] + P->me[2][1]*Sinv->me[1][1],
297       W->me[3][1] = P->me[3][0]*Sinv->me[0][1] + P->me[3][1]*Sinv->me[1][1],
298
299       if ( i==1 ) {
300         W->me[4][0] = P->me[4][0]*Sinv->me[0][0] + P->me[4][1]*Sinv->me[1][0],
301         W->me[5][0] = P->me[5][0]*Sinv->me[0][0] + P->me[5][1]*Sinv->me[1][0];
302
303         W->me[4][1] = P->me[4][0]*Sinv->me[0][1] + P->me[4][1]*Sinv->me[1][1];
304         W->me[5][1] = P->me[5][0]*Sinv->me[0][1] + P->me[5][1]*Sinv->me[1][1],
305       }
306
```



```
307       // State update and store
308       mv_mltadd(x,nu,W,1.0,xu),
309        set_col(modex,i,xu,0);
310
311       // Covariance update and store
312       I_WH->me[0][0] = 1 - W->me[0][0],    I_WH->me[0][1] = -W->me[0][1];
313       I_WH->me[1][0] =  -W->me[1][0];       I_WH->me[1][1] = 1 - W->me[1][1];
314       I_WH->me[2][0] =  -W->me[2][0];       I_WH->me[2][1] = -W->me[2][1];       I_
    WH->me[2][2] = 1;
315       I_WH->me[3][0] = -W->me[3][0];        I_WH->me[3][1] = -W->me[3][1];       I_
    WH->me[3][3] = 1;
316
317       if ( i==1 ) {
318         I_WH->me[4][0] =  -W->me[4][0];       I_WH->me[4][1] = -W->me[4][1];
    I_WH->me[4][4] = 1;
319         I_WH->me[5][0] = -W->me[5][0];        I_WH->me[5][1] = -W->me[5][1];
    I_WH->me[5][5] = 1;
320       }
321
322       m_add(quadtr(I_WH,P,KPKtr),quadtr(W,R,WRWtr),Pu);
323       m_to_col(Pu,modeP,i),
324
325   #ifdef DEBUGALLTHOME
326       if ( i==0 ) {
327         if (( f = fopen("/home/bjarry/AIRBORNE/test/pf.out","a")) == NULL)
328           {
329             fprintf(stderr,"Cannot open output file.\n");
330             exit(1);
331           }
332
333         fprintf(f,"\n\nIMM CVCA_FiltTrack: track %d time %f\n", state_upd->track_num
    ber, state_upd->time);
334         fprintf(f,"xCV\n");  v_foutput(f, x);
335         fprintf(f,"PCV\n");  m_foutput(f, P);
336         fprintf(f,"zCV\n");  v_foutput(f, z);
337         fprintf(f,"SCV\n");  m_foutput(f, S);
338         fprintf(f,"SinvCV\n");  m_foutput(f, Sinv);
339         fprintf(f,"WCV\n");  m_foutput(f, W);
340         fprintf(f,"I_WHCV\n");  m_foutput(f, I_WH);
341         fprintf(f,"PuCV\n");  m_foutput(f, Pu);
342
343         fclose(f);
344       }
345   #endif
346
347       // Likelihood calculations
348       likelv_calc(nz,i,cbar->me[i][0],S,Sinv,nu,c);
349     }
350
351     v_free(z);    v_zero(x);       v_free(xu);       v_free(nu);
352     m_zero(P);    m_free(S);       m_free(Sinv);     m_free(W);
353     m_free(Pu);   m_free(I_WH);    m_free(KPKtr);    m_free(WRWtr);
354
355     // Mode probability update
356     modemu_calc(nm,c,modemu);
357     m_free(c);
358
359     // Combination for output
360     x_calc(nx,modex,modemu,x),
361     P_calc(nx,nm,modex,modeP,modemu,x,P);
362
363     // Put into state_upd
364     state_upd->state_vec->me[0][0] = x->ve[0];
365     state_upd->state_vec->me[1][0] = x->ve[1];
366     state_upd->state_vec->me[2][0] = x->ve[2];
367     state_upd->state_vec->me[3][0] = x->ve[3];
```

```
168
169     state_upd->cov_mat->me[0][0] = P->me[0][0];
170     state_upd->cov_mat->me[0][1] = P->me[0][1];
171     state_upd->cov_mat->me[0][2] = P->me[0][2];
172     state_upd->cov_mat->me[0][3] = P->me[0][3];
173
174     state_upd->cov_mat->me[1][0] = P->me[1][0];
175     state_upd->cov_mat->me[1][1] = P->me[1][1];
176     state_upd->cov_mat->me[1][2] = P->me[1][2];
177     state_upd->cov_mat->me[1][3] = P->me[1][3];
178
179     state_upd->cov_mat->me[2][0] = P->me[2][0];
180     state_upd->cov_mat->me[2][1] = P->me[2][1];
181     state_upd->cov_mat->me[2][2] = P->me[2][2];
182     state_upd->cov_mat->me[2][3] = P->me[2][3];
183
184     state_upd->cov_mat->me[3][0] = P->me[3][0];
185     state_upd->cov_mat->me[3][1] = P->me[3][1];
186     state_upd->cov_mat->me[3][2] = P->me[3][2];
187     state_upd->cov_mat->me[3][3] = P->me[3][3];
188
189   #ifdef DEBUG
190     if ((f = fopen("/home/bjarry/AIRBORNE/test/tf_out","a")) == NULL)
191     {
192       fprintf(stderr,"Cannot open output file.\n");
193       exit(1);
194     }
195
196     fprintf(f,"\nIMM_CVCA_FiltTrack: track %d time %f\n", state_upd->track_number,
      state_upd->time);
197     fprintf(f,"IMM_state_upd->modex\n");  m_foutput(f, IMM_state_upd->modex);
198   #ifdef DEBUGALOT
199     fprintf(f,"IMM_state_upd->modeP\n");  m_foutput(f, IMM_state_upd->modeP);
200   #endif
201     fprintf(f,"IMM_state_upd->modemu\n"); m_foutput(f, IMM_state_upd->modemu);
202
203     fclose(f);
204   #endif
205
206     v_free(x);   m_free(P);
207
208     return;
209   }
```

```
1
2     ######################################
3     # MSDF Parameters                    #
4     # Units are:                         #
5     #   Distance ::  METERS              #
6     #   Time     ::  SECONDS             #
7     #   Speed    ::  M/S                 #
8     #   Angles   ::  RADIANS.            #
9     ######################################
10    # Default altitude is in meters.     #
11    # state_history is in seconds.       #
12    ######################################
13
14    identity_history::        10
15    state_history::           120.
16    #default_altitude::        10000.
17    default_altitude::        0.
18    #kill_delta_time::         15.5
19    kill_delta_time::         100000.0
20
21    # Add noise only if the input data are unnoisy
22    add_noise::               0
23    noise_sigma_noise::       1.
24    start_seed::              1.
25
26    # Parameters related to data association.
27    # association type = 0 => Nearest Neighbour
28    # association type = 1 => JVC
29    association_type::        0
30    prob_threshold::          99.9
31    mass_prob_max::           9999.9
32    ass_gate_factor::         1000.
33    ass_boost_factor::        1.
34
35    ######################################
36    #  MSDF Internal Coordinate System
37    # Values MUST BE defined as in file
38    ########## CoordTypes.h; ############
39    ######################################
40
41    ownship_position_coordinates::   0     # XY_REL_OWNP
42    ownship_velocity_coordinates::   0     # VXVY_ABS
43
44    #target_position_coordinates::    2     # RB_REL_OWNS
45    #target_position_coordinates::    1     # XY_REL_OWNS
46    target_position_coordinates::    0     # XY_REL_OWNP
47    #target_velocity_coordinates::    1     # VXVY_REL_OWNS
48    target_velocity_coordinates::    0     # VXVY_ABS
49
50
51    ################################################
52    #   Output Parameters (OFF = 0   ON = 1)       #
53    ################################################
54
55    print_agents_timing::     0
56    plot_results::            1
57    print_results::           1
58
59
60    ########################################################
61    # Parameters related to data positional fusion
62    # tracking type = 0 => Extended Adaptive Kalman Filter
63    # tracking type = 1 => IMM-CV-CA
64    ########################################################
65    tracking_type::           0
66    nmin_for_q::              5
```



```
67    nmax_for_q::              5
68    q_min::                   0.000001
69
70
71    ########################################################################
72    # Parameters related to the processing of out-of-sequence data.
73    # out_of_sequence_proc = 0 => No processing
74    # out_of_sequence_proc = 1 => Processing
75    ########################################################################
76    out_of_sequence_proc::          0
77    noise_free_retrodiction::       0
78    prediction_with_lag::           0
79
80
81    # Parameters related to Truncated Dempster-Shafer.
82    id_fusion_opt::           1
83    ident_max_mass::          0.95
84    ident_min_mass::          0.01
85    ident_updtime1::          0.5
86    ident_updtime2::          1.5
87    ident_updtime3::          0.5
88    ignorance_threshold::     0.01
89    delta_vel_threshold::     0.2
90    vel_igno_threshold::      0.2
91    length_igno_threshold::   0.1
92
93    ###################################
94    # Parameters related to sensors.
95    ###################################
96    # Sensor position coordinates:
97    #   Centered on the ESM;
98    #   X is Starboard (right)
99    #   Y is Forward
100   # Sigma RB in meters and radians
101   ###################################
102   sg150_xposition::         0
103   sg150_yposition::         0
104   sg150_zposition::         0
105   sg150_sigma_range::       30.
106   sg150_sigma_bearing::     0.006
107
108   sps49_xposition::         0
109   sps49_yposition::         0
110   sps49_zposition::         0
111   sps49_sigma_range::       500.
112   sps49_sigma_bearing::     0.0035
113
114   lff49_xposition::         0
115   lff49_yposition::         0
116   lff49_zposition::         0
117   lff49_sigma_range::       500
118   lff49_sigma_bearing::     0.0175
119
120   lff150_xposition::        0
121   lff150_yposition::        0
122   lff150_zposition::        0
123   lff150_sigma_range::      500
124   lff150_sigma_bearing::    0.0175
125
126   esm_sigma_bearing::       0.025
127   esm_sigma_vbearing::      0.1
128
129   aps506_xposition::        0
130   aps506_yposition::        0
131   aps506_zposition::        0
132   aps506_sigma_range::      45.0
```

```
apx506_sigma_bearing::  0.0042


apx502_xposition::      0.
apx502_yposition::      0
apx502_zposition::      0.
apx502_sigma_range::    45.0
apx502_sigma_bearing::  0.0042

lamps_radar_xposition:: 0.
lamps_radar_yposition:: 0.
lamps_radar_zposition:: 0.
lamps_radar_sigma_range::       92.65       /* lamps radar uncertainties for LAMP
S scenario 5 */
lamps_radar_sigma_bearing::     0.0087

prss_xposition::        0.
prss_yposition::        0.
prss_zposition::        0.
prss_sigma_range::      9265.0      /* prss uncertainties for LAMPS scenario 5 *
/
prss_sigma_bearing::    0.0174

link11m1_sigma_pos::    0.001
link11m1_sigmabearing:: 0.0175
link11m2_sigmarange::   100
link11m2_sigmabearing:: 0.0175

ownship_sigma_pos::     0.001
ship_sigma_pos::        0.001

default_sigmarb::       0.0
default_sigma_vxyair::  300.
default_sigma_vxysurf:: 3.
default_vbearing::      0.1


#######################################
# Caseatti Parameters (seconds)    #
#######################################
time_start::            0.0
step_time::             0.1
refresh_time::          2.0
time_update::           0.

#########################
# SARsim Parameters    #
#########################
SARsim_ships_dir::      /home/ajouan/NEWTESTBED/TESTBED/src/SARSIM/ships/

################################################
# STA Parameters                               #
# Units are meters, meter/second, & radians.#
################################################
QR_range::              11000.
QR_cpa::                9150.
QR_speed::              225.
ASC_bearing_min::       4.66
ASC_bearing_max::       4.76

# Units are meters & meter/second.
cluster_min_radius::    8300.
cluster_deltav_max::    170.

################################################
# RM Parameters                                #
```



```
# Units are meters, seconds, & meter/second #
# Proximity fuse is in meters               #
#############################################
gun_inventory::         16
gun_inventory::         150
t_deploy_sam::          3 1
t_deploy_gun::          1 5
range_minimum_sam::     0 0
range_maximum_sam::     20000
range_minimum_gun::     0.0
range_maximum_gun::     5000.
range_maximum_str::     50000
weapon_speed_sam::      300.
weapon_speed_gun::      850
proximity_fuse::        500
sam_kill_probability::  1 0
gun_kill_probability::  1 0


################################################
# Tracking Parameters                          #
# Used by the IMM.                             #
################################################
selectedModeCombo::     0
mode_prob_flow_all::    0.05
mode_prob_flow_a12::    0 05
mode_prob_flow_a21::    0 05
mode_prob_flow_a22::    0.05
process_noise_cov_CV::  1 0
process_noise_cov_CA::  1 0
default_sigma_axyair::  10 0
default_CV_mode_prob::  0 9
default_CA_mode_prob::  0 1

-100-)
```