



NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

**A DISTRIBUTED SENSOR NETWORK ARCHITECTURE
FOR DEFENSE AGAINST THE SHIP AS A WEAPON IN
THE MARITIME DOMAIN**

by

Jackson Ng

June 2011

Thesis Co-Advisors:

Thomas V. Huynh
Kim Leng Poh

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			<i>Form Approved OMB No. 0704-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE June 2011	3. REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE A Distributed Sensor Network Architecture for Defense Against the Ship as a Weapon in the Maritime Domain			5. FUNDING NUMBERS	
6. AUTHOR(S) Jackson Ng				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government, and Ministry of Defence or the Singapore Government.. IRB Protocol number: N/A.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			12b. DISTRIBUTION CODE A	
13. ABSTRACT (maximum 200 words) A successful terrorist attack using a ship as a weapon (SAW) on shore infrastructure in the Malacca and Singapore Straits would cause chaos to global trade, as these Straits carry over one-quarter of the world's commerce and half the world's oil. This calamity must be prevented. Toward this goal, this thesis aims at developing and determining the best distributed sensor network (DSN) architecture and implementing a sensor fusion algorithm for tracking a SAW intended to run into the oil and chemical terminals on Jurong Island, Singapore. The work in this thesis involves the application of (1) an integrated systems engineering methodology for designing alternative DSN architectures, (2) Kalman and information filters for SAW tracking and sensor data fusion, (3) a track-to-track fusion algorithm, and (4) a Monte Carlo simulative study to assess the effectiveness of three distributed sensor fusion network architectures—centralized, de-centralized, and hybrid. Each distributed sensor fusion network architecture includes the various sensors that Singapore deploys in and along the Singapore Straits. The simulative study results indicate that, with and without communication bandwidth constraints, a ship with the intent to attack Jurong can be identified accurately at an earlier time with both the centralized and de-centralized sensor fusion network architectures than with the hybrid sensor fusion network architecture.				
14. SUBJECT TERMS Maritime Domain Awareness, Distributed Sensor Network, Sensor Fusion, Data Fusion, Track Fusion, Kalman Filter, Information Filter			15. NUMBER OF PAGES 425	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UU	

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

**A DISTRIBUTED SENSOR NETWORK ARCHITECTURE FOR DEFENSE
AGAINST THE SHIP AS A WEAPON IN THE MARITIME DOMAIN**

Jackson Ng
Military Expert Grade 5, Republic of Singapore Navy
B.Eng., National University of Singapore, 2002

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN SYSTEMS ENGINEERING

from the

**NAVAL POSTGRADUATE SCHOOL
June 2011**

Author: Jackson Ng

Approved by: Thomas V. Huynh
Thesis Co-Advisor

Kim Leng Poh
Thesis Co-Advisor

Clifford Whitcomb
Chair, Department of Systems Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

A successful terrorist attack using a ship as a weapon (SAW) on shore infrastructure in the Malacca and Singapore Straits would cause chaos to global trade, as these Straits carry over one-quarter of the world's commerce and half the world's oil. This calamity must be prevented. Toward this goal, this thesis aims at developing and determining the best distributed sensor network (DSN) architecture and implementing a sensor fusion algorithm for tracking a SAW intended to run into the oil and chemical terminals on Jurong Island, Singapore.

The work in this thesis involves the application of (1) an integrated systems engineering methodology for designing alternative DSN architectures, (2) Kalman and information filters for SAW tracking and sensor data fusion, (3) a track-to-track fusion algorithm, and (4) a Monte Carlo simulative study to assess the effectiveness of three distributed sensor fusion network architectures—centralized, de-centralized, and hybrid. Each distributed sensor fusion network architecture includes the various sensors that Singapore deploys in and along the Singapore Straits.

The simulative study results indicate that, with and without communication bandwidth constraints, a ship with the intent to attack Jurong can be identified accurately at an earlier time with both the centralized and de-centralized sensor fusion network architectures than with the hybrid sensor fusion network architecture.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	INTRODUCTION.....	1
A.	BACKGROUND	1
B.	PURPOSE.....	2
C.	RESEARCH QUESTIONS	3
D.	SCOPE	3
E.	BENEFITS OF THE RESEARCH.....	3
F.	APPROACH.....	3
G.	THESIS ORGANIZATION.....	5
II.	THE MARITIME DOMAIN SECURITY PROBLEM	7
A.	INTRODUCTION.....	7
	1. Maritime Domain Security (MDS)	7
	2. Maritime Domain Awareness (MDA)	7
	3. Network-Centric Operations (NCO).....	8
	4. Data Fusion.....	9
B.	PROBLEM STATEMENT	10
C.	MISSION ANALYSIS	11
	1. Operating Environment	11
	a. Physical Characteristics.....	11
	b. Shipping Traffic	11
	c. Maritime Conditions	13
	d. Climate Conditions.....	13
	2. Potential Targets	13
	3. Threat Characteristics.....	16
	a. Characteristics of Containers	17
	b. Characteristics of Tankers.....	17
	4. Attack Approaches.....	17
	5. Sensors, Sensor Platforms, and Fusion Centers.....	20
	a. Maritime Port Authority of Singapore	20
	b. Fearless Class Patrol Vessels	21
	c. Coastal Patrol Craft	22
	d. Maritime Patrol Aircraft.....	22
	e. Fusion Centers	23
	f. Open-Source Databases.....	23
	g. Future Sensors	24
	6. Sensor Coverage.....	24
D.	NEEDS ANALYSIS	26
	1. Sensor-Network Needs.....	26
	2. Measures of Merit	26
E.	REQUIREMENTS ANALYSIS	28
	1. Operational Requirements Analysis	28
	2. Functional Analysis.....	28
	a. Functional Requirements	28

	b. <i>Functional Decomposition</i>	29
	3. Measures of Performance.....	31
F.	SUMMARY	31
III.	DISTRIBUTED SENSOR NETWORK ARCHITECTURES.....	33
A.	INTRODUCTION.....	33
B.	FUSION ARCHITECTURES	33
	1. Centralized Data Fusion Architecture	33
	2. Decentralized Data Fusion Architecture	34
	3. Hybrid Data Fusion Architecture.....	34
C.	ARCHITECTURE ALTERNATIVES	36
	1. Centralized Architecture.....	37
	2. Decentralized Architecture	38
	3. Hybrid Architecture	39
D.	SUMMARY	40
IV.	SENSOR FUSION ALGORITHMS	41
A.	INTRODUCTION.....	41
B.	DATA FUSION MODELS.....	41
C.	STATE MODEL	42
D.	MEASUREMENT MODEL	43
E.	ESTIMATION FOR DATA FUSION.....	43
	1. Kalman Filter Algorithm	44
	2. Information Filter Algorithm	46
	3. Comparing KF and IF	47
	4. Track-to-Track Fusion	47
	a. Track Association Test.....	48
	b. Fusion of Estimates	48
F.	SUMMARY	48
V.	SIMULATIVE STUDY	49
A.	INTRODUCTION.....	49
B.	MODELING AND SIMULATION	49
	1. Model Development	49
	a. The Ship as a Weapon	49
	b. Sensor Topology.....	50
	c. Target Destination.....	50
	d. Sensor Network Architecture and Fusion Algorithm.....	50
	2. Simulation Program Description.....	51
	a. Centralized Sensor Network Architecture With Kalman Filter	52
	b. Centralized Sensor Network Architecture With Information Filter.....	53
	c. Decentralized Sensor Network Architecture with Information Filter.....	53
	d. Hybrid Sensor Network Architecture With Information Filter and Track-to-Track Fusion Algorithm.....	54

3.	Evaluation Parameters	54
a.	Calculate the Mean Impact Point.....	55
b.	Calculate the Probability of Impact.....	56
4.	Test Cases	57
C.	DISCUSSION OF RESULTS	59
1.	Comparison Between Kalman and Information Filters.....	59
2.	Comparison Between Centralized and Decentralized DSN Architectures	61
3.	Comparison Between Different Number of Sensors for Centralized DSN.....	63
4.	Comparison Between Centralized and Hybrid DSN Architectures Using Five Sensors.....	65
5.	Comparison Between Centralized and Hybrid DSN Architectures Using Nine Sensors	67
6.	Comparison Between Different Number of Sensors for Hybrid DSN.....	69
7.	Effects of Data Loss for Centralized and Decentralized DSN Architectures	71
a.	Using Five Sensors.....	71
b.	Using Eight Sensors.....	72
8.	Comparison Between Different DSN Architectures and Different Number of Sensors	73
D.	PERFORMANCE EVALUATION.....	75
E.	SUMMARY	76
VI.	CONCLUSION AND FUTURE RESEARCH.....	77
A.	INTRODUCTION.....	77
B.	RESEARCH SUMMARY	77
C.	KEY FINDINGS	77
D.	AREAS FOR FUTURE RESEARCH.....	78
E.	CONCLUSION	79
	APPENDIX A. SENSOR COVERAGE ANALYSIS.....	81
A.	MPA SHORE STATION SENSOR COVERAGE	81
B.	COASTAL PATROL CRAFT SENSOR COVERAGE.....	82
C.	PATROL VESSEL SENSOR COVERAGE	83
D.	MARITIME PATROL AIRCRAFT SENSOR COVERAGE.....	84
E.	COMPLETE SENSOR COVERAGE IN THE SINGAPORE STRAIT ..	85
1.	Case Without Sensor Failures.....	85
2.	Case With Sensor Failures	86
a.	Sensor Coverage Simulation Source Code	87
	APPENDIX B. SIMULATION PROGRAM SOURCE CODE.....	89
A.	BLOCK 1: CENTRALIZED DATA FUSION ARCHITECTURE USING KALMAN FILTER.....	89
B.	BLOCK 2. CENTRALIZED DATA FUSION ARCHITECTURE USING INFORMATION FILTER	168

C.	BLOCK 3. DECENTRALIZED DATA FUSION ARCHITECTURE USING INFORMATION FILTER	242
D.	BLOCK 4. HYBRID DATA & TRACK FUSION ARCHITECTURE USING INFORMATION FILTER & TRACK-TO-TRACK ALGORITHM.....	324
	LIST OF REFERENCES.....	401
	INITIAL DISTRIBUTION LIST	405

LIST OF FIGURES

Figure 1.	System-of-Systems Architecture Development Process (From: [11]).....	4
Figure 2.	Traffic Separation Scheme in the Singapore Strait (From: [20]).....	12
Figure 3.	Jurong Island, Located Southwest of Singapore Mainland (From: [24])	14
Figure 4.	Identified Target Areas of Interest in Jurong Island (From: [25-26]).....	15
Figure 5.	The East-to-West Routes Likely to Be Taken by a SAW to Attack Jurong Island (After: [24]).....	19
Figure 6.	The West-to-East Routes Likely to Be Taken by a SAW to Attack Jurong Island (After: [20]).....	19
Figure 7.	Photograph of a Patrol Vessel (From: [31]).....	21
Figure 8.	Photograph of a Coastal Patrol Craft (From: [34])	22
Figure 9.	Photograph of a Maritime Patrol Aircraft (From: [31]).....	23
Figure 10.	An Instance of the Sensor Platform's Disposition and Sensor Coverage Areas (After: [35])	25
Figure 11.	Average Number of Operational Sensors in Each Sector of the Singapore Strait in a Day	25
Figure 12.	Singapore MDA Sensor Network, Use Case Diagram	29
Figure 13.	Singapore MDA Sensor Network Requirements Diagram	30
Figure 14.	Three Sensor Data-fusion Architectures: (a) Centralized Processing at the Fusion Center, (b) Decentralized Processing at Each Node-processing Unit, and (c) Hierarchical Processing. (After: [40])	35
Figure 15.	Command and Control (C2) Structure within and between the Various Maritime Security Agencies	36
Figure 16.	Data Flow for Centralized Architecture.....	37
Figure 17.	Data Flow for Decentralized Architecture.....	38
Figure 18.	Data Flow for Hybrid Architecture.....	39
Figure 19.	Joint Directors of Laboratories Data Fusion Model (From: [42])	42
Figure 20.	Block diagram of the Kalman filter cycle (After: [44]).....	45
Figure 21.	Comparison of KF and IF for Centralized Five-sensor Network Architecture.....	60
Figure 22.	Comparison of Five-sensor Centralized and Decentralized DSN Architectures	62
Figure 23.	Comparison of Number of Sensors (Five and Nine) for a Centralized DSN...	64
Figure 24.	Comparison between Five-sensor Centralized and Hybrid DSN Architectures	66
Figure 25.	Comparison between Nine-sensor Centralized and Hybrid DSN Architectures	68
Figure 26.	Comparison of Number of Sensors (Five and Nine) for a Hybrid DSN.....	70
Figure 27.	Comparison of the Centralized DSN with No Data Loss, 10%, 30%, and 50% Data Loss, and the Hybrid DSN of Five Sensors	72
Figure 28.	Comparison of the Centralized DSN with No Data Loss, 10%, 30%, and 50% Data Loss, and the Hybrid DSN of Eight Sensors.....	73

Figure 29.	Comparison of the Centralized DSN with No Data Loss, 10%, 30%, and 50% Data Loss, and the Hybrid DSN of Five and Eight Sensors.....	74

LIST OF TABLES

Table 1.	Summary of Narrow Points in the Traffic Separation Scheme (After: [20])...	12
Table 2.	Numbers and Types of Vessel Arrivals, Over 75GRT, to Singapore (From:[27])	16
Table 3.	Four Categories of Measures of Merit (From: [36]).....	27
Table 4.	Applying Algorithms to Architectures.....	51
Table 5.	Various Cases Used in This Simulative Study	58
Table 6.	A Selection of Salient Data Points.....	75

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

AIS	Automatic Identification System
C2	Command and Control
C3	C2 and Communications
CPC	Coastal Patrol Craft
CVM	Constant Velocity Model
DF	Data Fusion
DWT	Deadweight Tonnage
DSN	Distributed Sensor Network
GRT	Gross Registered Tons
HARTS	Harbour Craft Transponder System
HQ	Headquarters
IF	Information Filter
IMB	International Maritime Bureau
IMO	International Maritime Organization
JDL	Joint Directors of Laboratories
KF	Kalman Filter
M&S	Modeling and Simulation
MDA	Maritime Domain Awareness
MDS	Maritime Domain Security
MOE	Measure of Effectiveness
MOP	Measure of Performance
MPA	Maritime Port Authority of Singapore
MPaA	Maritime Patrol Aircraft
NCO	Network-Centric Operation
PCG	Police Coast Guard
POCC	Port Operations Control Centre
PV	Patrol Vessel
RSAF	Republic of Singapore Airforce
RSN	Republic of Singapore Navy
SAW	Ship As a Weapon

SMSC	Singapore Maritime Security Centre
SoS	System of Systems
SoSADP	SoS Architecture Development Process
TEU	Twenty-foot Equivalent Unit
TSS	Traffic Separation Scheme
U.S. EIA	United States Energy Information Administration
VTIS	Vessel Traffic Information System
VTs	Vessel Traffic Service

ACKNOWLEDGMENTS

This thesis would not have been possible if not for the guidance and brilliance of my thesis advisor, Professor Thomas Huynh. I am indebted to him for his tireless efforts and amazing patience in helping me learn the algorithms required for this research. In the course of this research, I finally understood what it means and takes to be a sound and credible systems engineer.

To my family, no words can describe how grateful I am to have them by my side each day, showering me with their unconditional love and encouragement. To my wonderful wife, Saranthorn, “Thank you! You are the greatest.” To my two fantastic children, Aaron and Ava, “Let’s go Disneyland! Daddy’s ‘homework’ is finally complete!”

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

The lion's share of Singapore's trade is conducted by sea. ... 50% of global oil shipments, including 70% of Japan's oil imports and 80% of China's oil imports, pass through the Malacca Strait every day. As a littoral state Singapore will continue to work together with our neighbouring littoral states to ensure that the Malacca and Singapore Straits are safe to allow freedom of navigation for global trade [1].

Minister for Defence Teo Chee Hean
January 16, 2009

A. BACKGROUND

Singapore's economy is dependent on global trade, with the bulk of it transported in the maritime domain. For 2010, the Maritime Port Authority of Singapore (MPA) estimated that Singapore's container throughput was 28.4 million 20-foot-equivalent units (TEUs) and total shipping tonnage was 1.9 billion gross tons. Singapore was the world's top bunkering port, with bunker sales reaching a record high of 40.9 million tons [2]. While exact figures have yet to be released at the time of writing, the maritime performance for Singapore is expected to exceed 7% of Singapore's GDP. The straits of Malacca and Singapore are the lifelines of Singapore, and it is in her vital interest to ensure the undisrupted flow of global trade in these waterways.

As for global energy, the United States Energy Information Administration (U.S. EIA) reports that the straits of Malacca and Singapore carry 15 million barrels of oil per day, making them the world's second most strategic chokepoints. Chokepoints are defined by the U.S. EIA as narrow channels along widely used global sea routes, and they pose navigational hazards to their heavy shipping traffic, which can lead to disastrous oil spills and exposure to piracy and terrorist attacks. Even a temporary blockage of a chokepoint can lead to an unwelcome detour, adding hundreds of miles to the route and substantially increasing the costs of freight, time delays, and even insurance premiums [3].

Piracy has plagued the straits of Malacca and Singapore for many years. Though the number of pirate attacks has been reported as on the decline in recent years, with none

reported by the International Maritime Bureau (IMB) [4] in the first quarter of 2010 (due to coordinated military patrols among the littoral states and user states), the straits of Malacca and Singapore remain attractive targets. Complicating the maritime threat picture is the growing nexus between piracy and terrorism [5]. Studies [6-8] have hypothesized the possibility of terrorist attacks similar to those that devastated New York City on September 11, 2001, but with ships as a weapon against ports, critical shore installations, or other high-value ships. For example, in March 2003, the MV *Dewi Madrim*, a chemical tanker, was boarded by pirates while underway. Reports suggested that the pirates were in fact terrorists trying to gain experience in ship handling before conducting an attack using a similar ship against U.S. naval vessels in port [8]. For the period of 2002–2008, Chalk [5] reported a “modest yet highly discernable spike in high-profile terrorist attacks and plots at sea.”

The potential devastation to lives and the economy that terrorist attacks can produce gives Singapore a high incentive to enhance its maritime security. One way to enhance maritime security is to achieve maritime-domain awareness (MDA), which the U. S. government defines as “the effective understanding of anything associated with the maritime domain, all areas and things of, on, under, relating to, adjacent to, or bordering on a sea, ocean, or other navigable waterway” [9]. In other words, the key to an effective MDA is to make data and information available through a multitude of reliable sources and high-fidelity sensors and fuse them to provide clarity on a situation and enable faster and better-informed decisions in response to maritime threats. This requires systematic understanding and engineering of a complex system of systems (SoS) that includes sensors, communications links, decision-support systems, and engagement systems [10].

B. PURPOSE

The research reported in this thesis seeks to identify the best distributed sensor network (DSN) architecture for a chosen sensor fusion algorithm for Singapore’s protection against the ship as a weapon (SAW).

C. RESEARCH QUESTIONS

Two questions shape this research and direct the efforts towards achieving the purpose of this thesis: What sensor-network architecture is best for Singapore to use in defense against a SAW? What is the most suitable sensor fusion algorithm to implement in support of the chosen architecture?

D. SCOPE

The scope of this research is to develop a DSN architecture and implement a sensor fusion algorithm to realize accurate identification of SAWs and early warning of impending impact upon Jurong Island. The specific targets of interest in this research are the oil and chemical terminals on Jurong Island. This research takes into consideration Singapore's existing suite of sensors (e.g., surveillance and navigation radars) and sensor platforms (e.g., naval patrol vessels, police and coast guard boats, maritime patrol aircraft, and MPA radar towers) used in and along the Singapore Strait.

E. BENEFITS OF THE RESEARCH

Determining and applying an optimal DSN architecture will produce optimal performance outcomes in terms of accurately identifying a ship's intentions and raising the alarm in a timely manner. These outcomes will provide valuable insights in developing response capabilities against SAWs and enhancing maritime security in the Singapore Strait. The results and concepts produced from this research are intended to be applicable to other key installations within Singapore (e.g., Tuas and Changi naval bases and other commercial terminals) and beyond (e.g., the Malacca and Hormuz straits), and for any given mix of available sensors (sea-, land-, or aerial-based and fixed or mobile).

F. APPROACH

A three-stage approach is taken in this research. The first stage develops different sensor network architectures. The second stage identifies and implements a suitable sensor fusion algorithm to support these architectures. In the third stage, a comparative study is made using modelling and simulation to evaluate the performance of the various sensor network architectures. The overall process that guides these three stages follows

closely the System-of-Systems Architecture Development Process (SoSADP) (Figure 1) [11]. The SoSADP presents a “logical, systematic, layered process” that begins with a SoS problem and flows through mission, needs, and requirements analyses to develop alternative SoS architectures. It then conducts cost and risk analysis and concludes by assessing and ranking the alternative architectures using modelling and simulation. How the SoSADP guides the development of the various sensor-network architectures for MDA will be elaborated in later chapters.

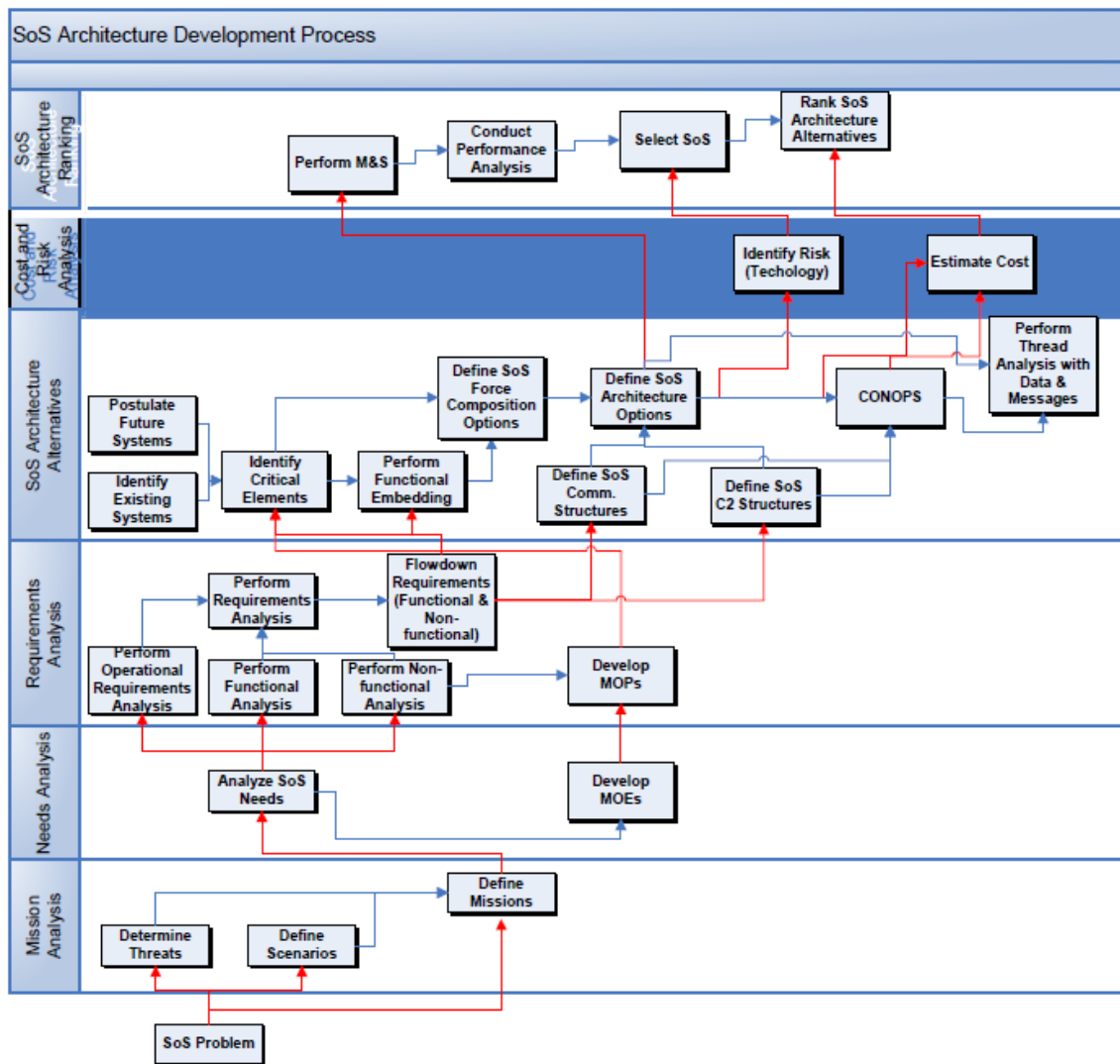


Figure 1. System-of-Systems Architecture Development Process (From: [11])

G. THESIS ORGANIZATION

This chapter provides the background, purpose, scope, and benefits of the research and the approach to achieving the stated goals. In Chapter II, the relationships among maritime domain security (MDS), maritime domain awareness (MDA), network-centric operations (NCO), and data fusion (DF) are discussed. Next is a description of the problem statement for this research and mission, needs, and requirements analyses, which are the first three layers of the SoSADP. In Chapter III, the fourth layer of SoSADP, the alternative sensor-network architectures, is developed. In Chapter IV, alternative sensor algorithms are discussed. In Chapter V, the fifth and last layer of SoSADP, a comparative study using simulation to select the best sensor-network architecture and its associated sensor-fusion algorithm is conducted. Finally, Chapter VI summarizes the research findings and recommends areas for future work.

THIS PAGE INTENTIONALLY LEFT BLANK

II. THE MARITIME DOMAIN SECURITY PROBLEM

A. INTRODUCTION

The first part of this chapter describes the security challenges posed in the maritime domain, rationalizes the need for maritime domain awareness, explains how network-centric operations (NCO) support the enhancement of MDA, and discusses the relationship between data fusion (DF) and NCO. The second part of this chapter describes the research problem and carries out the first three layers of the SoSADP: the analysis of mission, needs, and requirements.

1. Maritime Domain Security (MDS)

Acts of terrorism remain a clear and present danger today and for many years to come. Studies [5, 8] have predicted that terrorist acts in the maritime domain are likely to continue, if not grow. Five reasons for the shift in terrorist attack from predominately land- to sea-based are put forth by Chalk [5]. First, the same openness and lack of regulation in many port and coastal areas that encourages pirate attacks applies equally to acts of terror. Second, the accessibility and growth of maritime sports and activities have granted training and resource opportunities for operations at sea. The financial and psychological devastation, in the short and long terms, to local and global economies that can be achieved from a successful and sizeable maritime attack is very attractive account for the other two reasons. Finally, extensive and hard-to-inspect, global container shipping facilitates the covert movement of resources (arms and explosives, chemical and biological material, nuclear material, etc.). Therefore, in order to thwart a terrorist attack in the maritime domain, one would need to build barriers to harden security and not let these five factors become an option of choice.

2. Maritime Domain Awareness (MDA)

One way to enhance MDS is to achieve MDA. In Chapter I, MDA is defined as knowing everything and anything that happens in the maritime domain. Therefore, achieving MDA rests on the ability to continuously survey maritime and maritime-related

activities such that contacts of interest can be detected, their movements tracked, and their intention classified. Galdorisi and Goshorn [12] list seven MDA requirements that are imperative to attaining a comprehensive and accurate appreciation of maritime domain activities:

- Focused sensing and data acquisition
- Dynamic, interoperable connectivity
- Responsive information management
- Information assurance
- Consistent representation
- Distributed collaboration
- Dynamic decision support

While these seven requirements represent vitally important parts of the “MDA value chain,” the last two bear most directly on the challenge of comprehensive MDA. Focused sensing and data acquisition refers to the collection of specific data and information on the contact or area of interest, as opposed to gathering a barrage of ambiguous data. Dynamic decision support emphasizes the need to deploy sensing resources efficiently and sparingly, as more data does not mean better data. In this thesis, these two MDA requirements are considered in the evaluation of sensor-network architecture and data fusion algorithms. The remaining five requirements form the elements of NCO. Though not the primary focus of this thesis, they are indispensable in the attainment of comprehensive MDA and are briefly discussed next.

3. Network-Centric Operations (NCO)

The cornerstone of NCO lies with a networked force. Networks can increase operational effectiveness, because they enable distributed sensors to be synchronized in time, facilitate communication among dispersed forces, and integrate intelligence, surveillance, and reconnaissance to build and share a common operating picture [13]. The five NCO-related MDA requirements are now elaborated.

- Dynamic, interoperable connectivity. The expansiveness of the maritime domain demands that all users have reliable, secure, and flexible access to each other and other information sources. Connectivity must be dynamic

in order to address the changing real-time needs of users and changes to the environment as bandwidth demands change along with the operation scenario.

- Responsive information management. The user must have enough information to make informed decisions, but not so much as to cause information overload. Hence, the methods of accessing information — user information pull, producer information push, and planned information ordering — need to be well balanced.
- Information assurance. The provision of access controls, authentication mechanisms, confidentiality, and integrity features enables users to assert their identity and access resources in both peer–peer and client–server interactions.
- Consistent representation. Information must be consistent spatially, temporally, and as to content for users to act in sync. While every user at every level is not necessarily required to view the identical common operational picture at all times, every user must have access to the same accurate and timely information.
- Distributed collaboration. This imperative involves maintaining fully connected and transparent interactions among users and providing tools and connectivity for collaboration at the user level.

MDA cannot be achieved effectively without networks, and networks cannot be evaluated in isolation without considering limitations or disruptions to communications. In this thesis, communications bandwidth is used as a limiting factor to determine its interaction and influence on the performance of the sensor network architectures studied.

4. Data Fusion

Charniak *et al.* describes data fusion (DF) as “a multilevel, multifaceted process dealing with the detection, association, correlation, estimation and combination of data and information from multiple sources to achieve refined state and identity estimation, and complete and timely assessments of situation and threat” [14]. The output produced from fusing data captured from a multitude of differing sensors should be richer in information than any individual sensor can. Furthermore, the automation of the DF process is necessary today because the vast amount of data captured, along with progressively more complex and ambiguous data measurements, is exceeding the human ability to associate and classify manually. Thus, there is a push for sensor networks to

incorporate automation “of all or part of the individual processes of fusion, to provide timely, accurate, and easily understood information, rather than masses of raw data” [15]. Parenthetically, in this thesis, the terms DF process and DF will be used interchangeably.

Effective data fusion is dependent on the underlying sensor-network architecture, fusion algorithm, and communication links within the network [16]. In this thesis, the NCO provides the context on how data fusion should be employed to fuse data captured independently from different sensor systems from different agencies. Communications bandwidth limits are taken into account to determine whether DF should be centralized or distributed. This in turn determines if sensors or sensor nodes should transmit large quantities of raw data or processed track estimates. Transmitting raw data may ensure performance optimality by creating “super” measurements, but may not be practical under realistic operational settings. In addition, while transmitting processed track estimates can reduce bandwidth requirements, it adds more processing layers and may lose some richness of information.

B. PROBLEM STATEMENT

The distributed sensor network architecture problem statement is a description of the DSN that needs to be built, and it sets in motion the SoSADP. It provides missions, threats, multiagency undertaking, timeframe, geographical settings, needs in a sensor network, and constraints [11].

In this research, the problem to be addressed is captured in two questions: what should the DSN architecture be, given the different and dispersed sensors and sensor platforms used by Singapore, and what is the most suitable sensor fusion algorithm to support the architecture so as to enable accurate and early identification of hostile intent by a ship transiting the Singapore Strait? Singapore and the Singapore Strait serve as the setting for this research. The SAW is assumed to attack the chemical and oil terminals on Jurong Island.

C. MISSION ANALYSIS

Mission analysis consists of determining threats, refining missions, defining scenarios, and determining mission requirements [11]. A scenario includes the threats and their possible navigation routes, the targets and their locations, the sensors, and the physical environment they all operate in.

1. Operating Environment

a. Physical Characteristics

The Singapore Strait is located south of both the island of Singapore and the southeastern tip of peninsular Malaysia, north of the Indonesian Rian Islands. The narrowest breadth of the Singapore Strait is only about three miles, and throughout, it is less than 15 miles wide. At its eastern outlet into the South China Sea, where it is bounded solely by Malaysia and Indonesia, the seaway is approximately 11 miles wide [17]. The traffic separation scheme (TSS) adopted by the International Maritime Organization (IMO) in 1977 further constrains the allowable seaway for navigation. The TSS divides the straits of Malacca and Singapore into nine sectors with Sectors 1 to 6 lying in the Malacca Strait and Sectors 7 to 9 in the Singapore Strait (Figure 2). To provide an appreciation of the restrictedness of navigating in the Singapore Strait, Table 1 tabulates the narrow points within the Strait, with the narrowest just 0.33 nautical miles in Sector 8, south of St. John's Island.

b. Shipping Traffic

As a consequence of high shipping-trade volume in a narrow strait, ship traffic density in the Singapore Strait is generally high and especially so in the Philip Channel [18]. By regulation, very large crude carriers (i.e., a tanker of 150,000 DWT and above) and deep-draught vessels (45 feet and above) navigating in the Strait are prohibited from traveling at speeds exceeding 12 knots [19]. Otherwise, vessel speeds have been reported to range from 8 to 25 knots [18]. For navigational safety, the general traffic flow is about 15 knots.

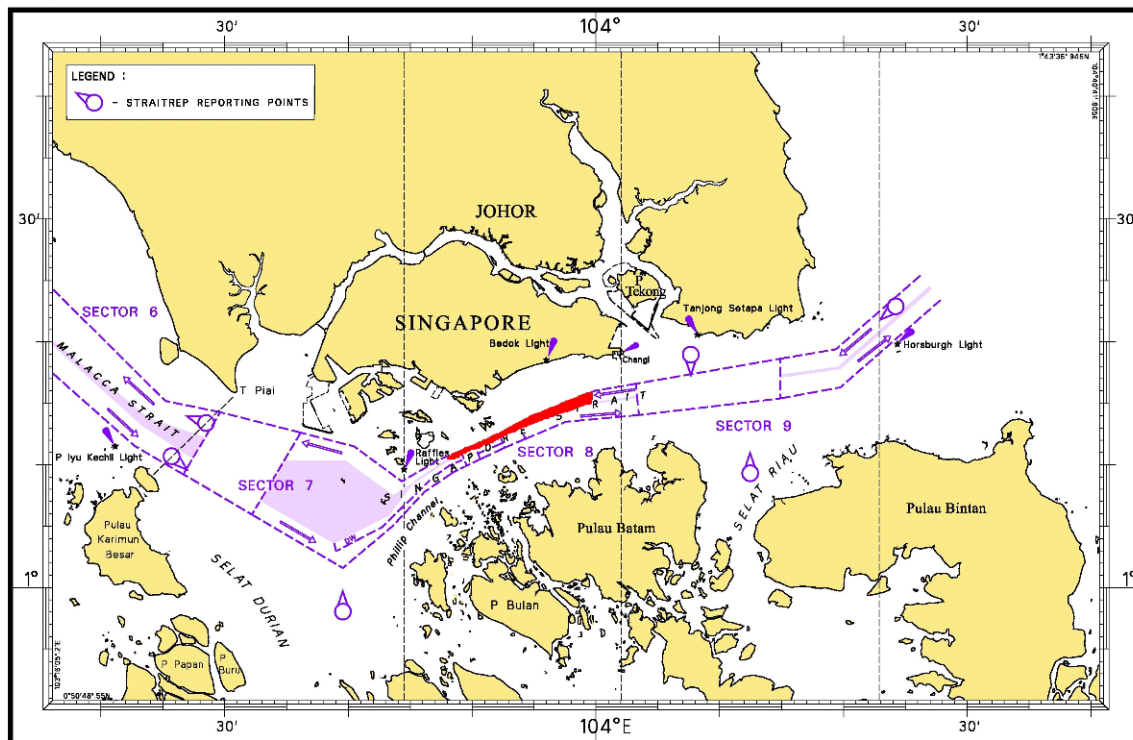


Figure 2. Traffic Separation Scheme in the Singapore Strait (From: [20])

Table 1. Summary of Narrow Points in the Traffic Separation Scheme (After: [20])

	Sectors 7-9 (Singapore Strait)
Overall	1.34 nautical miles (near St John's Island, Singapore, Sector 8)
Eastbound	1 nautical mile (near St John's Island, Singapore, Sector 8)
Westbound	0.33 nautical miles (near St John's Island, Singapore, Sector 8)

c. *Maritime Conditions*

The tidal range in the Singapore Strait varies within 5 feet and the tidal streams reach up to 6 knots [7]. Ships with deep draughts may be concerned with tidal heights, but the tidal stream would have little effect on ships with shallow draughts, such as a very large crude carrier traversing at high speed. In other words, the tidal conditions pose a minimal barrier to a SAW of large size travelling at high speeds towards its target. Furthermore, the sea is usually calm, with a sea-state condition of one or at most two. Visibility in general is good over the whole area, except during heavy rain showers [21].

d. *Climate Conditions*

The climate of the Straits of Singapore is equatorial, with uniform high temperature, high humidity, and copious rainfall. Prevailing winds are generally north to northeast during the Northeast Monsoon (December to April) and south to southeast during the Southwest Monsoon (June to October). Wind speeds average less than 10 knots throughout the year, although occasional strong gusts are likely when showers or thunderstorms are in the vicinity. In sum, the climate conditions in the Singapore Straits are fairly constant and would not pose a navigational deterrent to using a ship as a weapon.

2. *Potential Targets*

The southern coast of Singapore presents an attractive target for seaborne terrorist attacks because of its accessibility and proximity to the Straits of Singapore. There are several potential targets along the southern coast—seaports, industrial parks, Sentosa Island, and even naval bases—but Jurong Island stands out among them all. Jurong plays a significant role in Singapore's economy and the global business ecosystem. It hosts over 95 global companies, including heavyweights such as Shell, ExxonMobil, Chevron, DuPont, BASF, Sumitomo Chemicals, and Mitsui Chemicals, which collectively have invested more than S\$31 billion in fixed assets on the island. In addition, these companies are the largest contributors to Singapore's manufacturing output, at almost 40% in 2008 [22–23]. And because a successful terrorist attack would trigger undesired

large-scale economic, energy, and environmental destruction, Jurong Island was included in the list of restricted areas on 8 December 2006 [25] to guard against approach by unauthorized vessels. Hence, Jurong Island is the ideal target choice for this research.

Three locations, A1, A2, and A3 (Figure 3), have been identified as potential impact destinations for a SAW. They are selected primarily due to their proximity to the Singapore Strait and high concentration of refineries and infrastructure near the coast. Furthermore, any sort of attack against any of these three areas would be equally applicable to the adjacent areas.

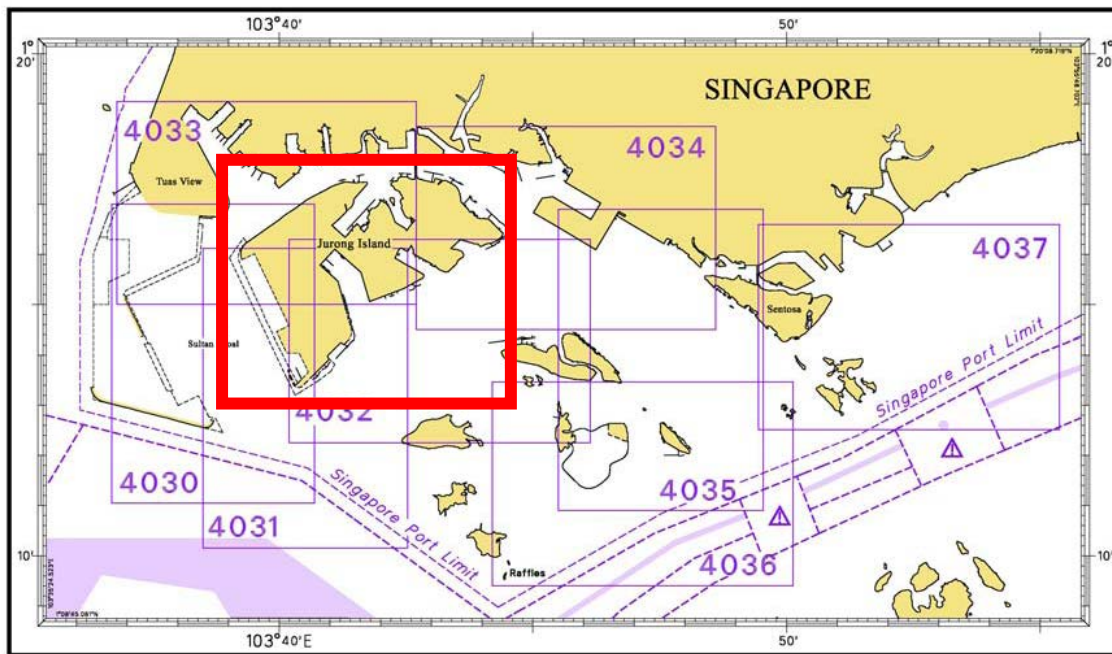


Figure 3. Jurong Island, Located Southwest of Singapore Mainland (From: [24])

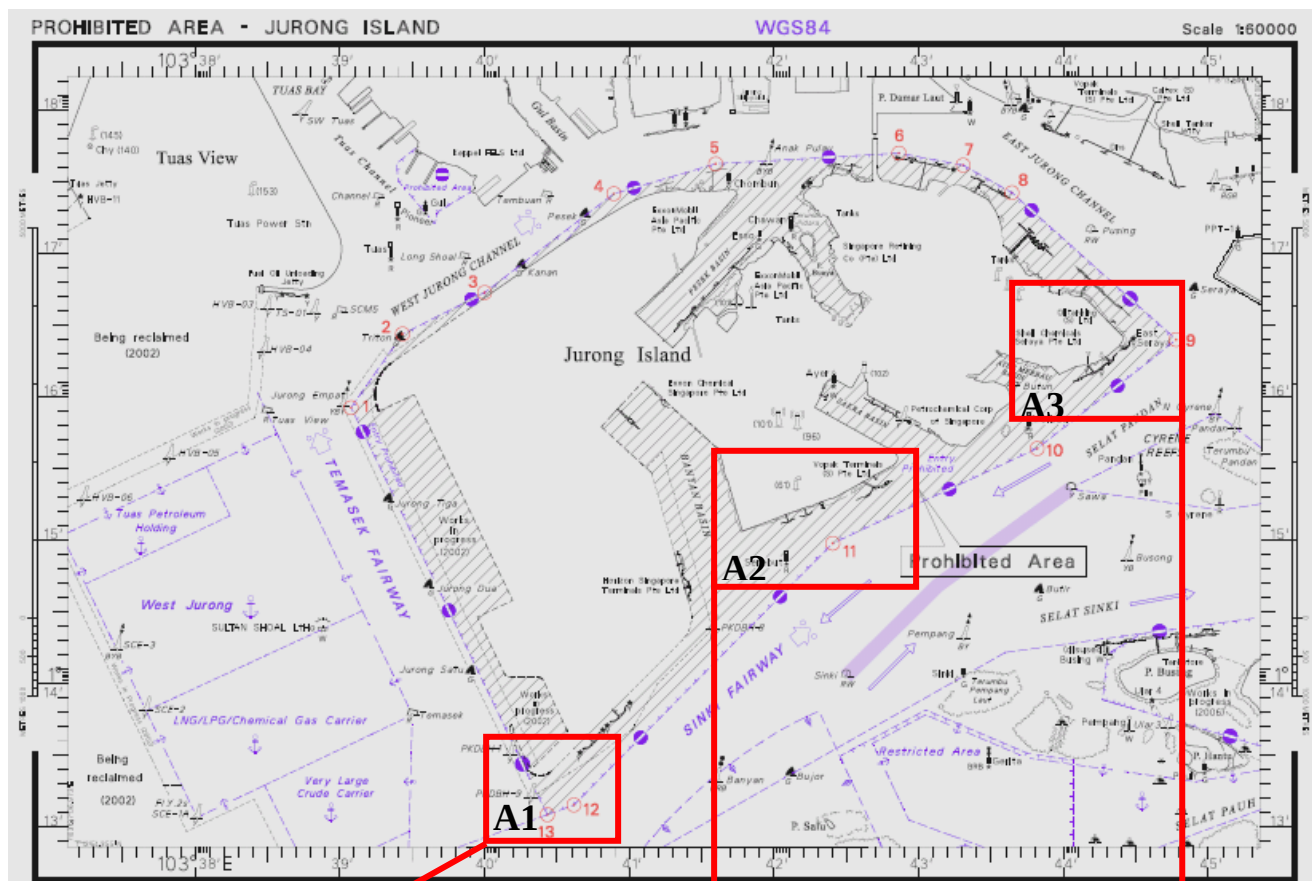


Figure 4. Identified Target Areas of Interest in Jurong Island (From: [25-26])

3. Threat Characteristics

Singapore is ranked as one of the world's busiest port, with an average annual arrival of about 130,000 vessels of 75 gross registered tons (GRT) and above [27]. While it has been reported that some types of ship (particularly smaller, general-cargo ships, product tankers, tugs, and barges) are more likely to be attacked than others [17], the report is based mostly on piracy and sea-robbery incidents. A survey of high-profile maritime terrorist incidents from 1961 to 2004 shows that the targets of terrorist attacks consisted of cruise ships, tankers, freighters, coasters, trawlers, small boats, ports, and offshore oil terminals [5]. In this research, it is deemed that a SAW must be able to overcome offshore barriers such as water barricades and wharves. On impact, the SAW must not only create significant direct damage but also cause extensive collateral damage to onshore structures such as oil refinery plants. Hence, vessels that are both large and fast (i.e., traversing at a speed of greater than 20 knots), such as containers and tankers, are potential threats of concern. In 2010, the count of 40,322 containers and tankers represented about 31% of total shipping arrivals in Singapore (Table 2). This is a sizable number and such vessels are a recognized threat.

Table 2. Numbers and Types of Vessel Arrivals, Over 75GRT, to Singapore (From:[27])

Year	Total	Containers	Freighters	Coasters	Bulk Carriers	Tankers	Regional Ferries	Barges	Tugs	Misc.
2002	142,745	16,418	4,867	5,869	5,097	16,919	53,209	15,995	13,704	9,902
2003	135,386	16,155	4,477	6,091	5,298	17,084	49,149	14,651	12,597	9,414
2004	133,185	17,333	4,386	5,897	5,327	17,576	46,733	13,947	12,274	9,314
2005	130,318	18,415	4,594	5,228	6,636	17,315	43,030	12,904	11,853	9,871
2006	128,922	19,161	4,610	4,909	7,912	18,195	37,986	12,789	12,561	10,009
2007	128,568	19,946	4,873	4,991	8,653	19,312	36,530	11,600	11,772	10,160
2008	131,695	20,589	5,083	4,619	9,280	19,460	32,643	14,047	13,736	11,215
2009	130,575	18,005	4,415	3,850	11,873	20,080	31,247	14,778	15,269	10,129
2010	127,299	18,967	4,619	4,123	12,234	21,355	31,094	11,972	12,551	9,739

a. *Characteristics of Containers*

Container vessels represent about 11.6% of the worldwide merchant-fleet tonnage and they make up 14.6% of vessel arrivals in Singapore. They range in size from small vessels of about 1,200 deadweight tonnage (DWT) to very large ships lifting up to about 12,000 TEUs. Speeds range from about 14 knots for the smaller vessels to 25 knots for the larger ships [18]. The possibility of containers carrying nuclear devices or weapons of mass destruction (WMD) is of growing concern to maritime-safety agencies, and more so after several spectacular accidents due to the combustion or explosion of undeclared or mis-declared cargo [28].

b. *Characteristics of Tankers*

Tankers make up 16.4% of vessel arrivals in Singapore. There are several variants, mainly oil, chemical, and liquefied gas. Crude-oil tankers are generally large vessels over 100,000 DWT with speeds of around 15 knots. A successful terrorist attack on a crude-oil tanker could cause massive economic and environmental damage. Chemical tankers, by contrast, are smaller, ranging from about 1,000 DWT to 50,000 DWT, with speeds of around 10 to 12 knots for the smaller vessels. They carry highly volatile cargoes that, in the event of a spill or marine accident, may pose severe hazards to human life, property, and the marine environment. The liquefied-gas tankers range widely in size, from small liquefied-petroleum-gas carriers of about 1,000 DWT to large liquefied-natural-gas carriers of around 90,000 DWT. The larger vessels are relatively fast, with service speeds of about 18 knots. The cargo carried by these ships can potentially be used as a floating bomb [29], and hence, are a potential target for terrorist attacks.

4. *Attack Approaches*

As a SAW, intending to attack Jurong Island, is traversing the Singapore Strait within the TSS, it makes an abrupt turn towards the island when it reaches the narrowest distance from the TSS to the island. Figures 5 and 6 illustrate the different attack approaches the SAW can take.

Examining the TSS and the distances from the TSS to Jurong Island, the least noticeable approach to Jurong would be to transit east to west then turn towards area A1 (see the red, bold lines in Figure 5). This approach offers the shortest distance from the point where the SAW leaves the TSS for area A1, giving authorities the least time to assess intent and to respond. It is very likely that this same route will be taken to reach areas A2 and A3, because it affords a shorter distance as compared to turning in at the south of Sentosa, where shipping traffic is probably higher due to the presence of Sentosa, Keppel Terminal, and Pasir Panjang Terminal (see the black, dashed lines in Figure 5).

If the SAW approaches Jurong Island while traversing west to east, it will attract considerable attention when it cuts across the TSS before heading towards Jurong Island (see the black, dashed lines in Figure 6). It is, therefore, deemed unlikely that the SAW will take a west-to-east approach.

In the worst-case scenario, the SAW would take the westbound route, navigate the TSS without attracting much attention and launch an attack on area A1. This attack approach offers the SAW the shortest travel from the TSS.

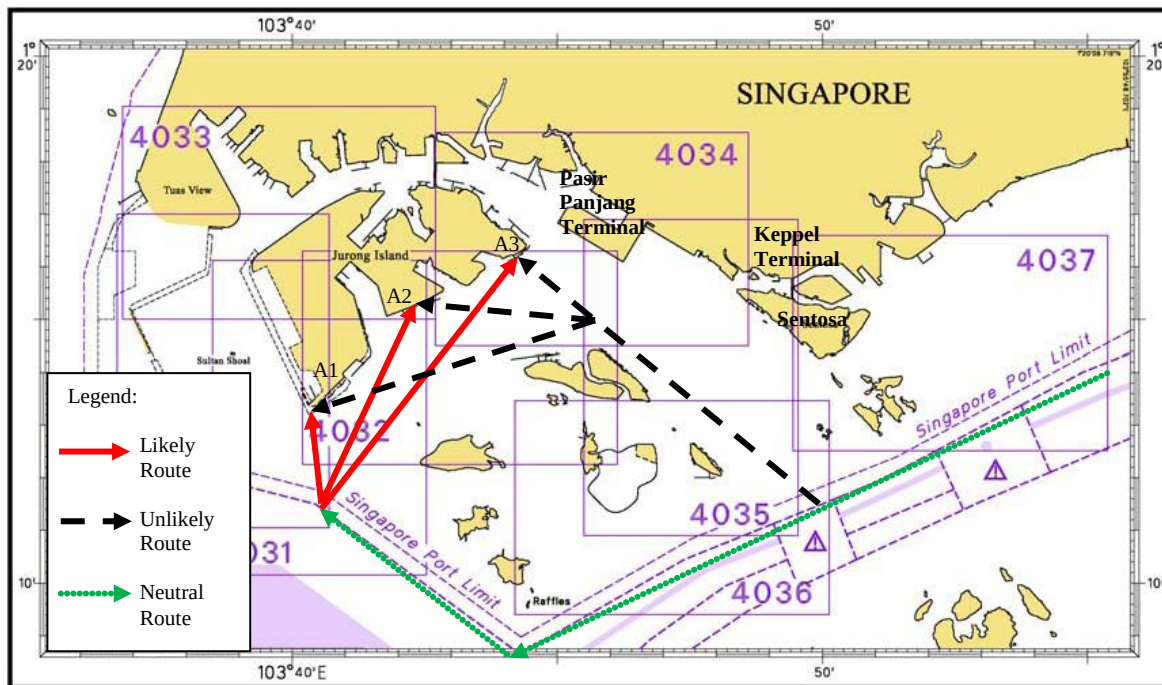


Figure 5. The East-to-West Routes Likely to Be Taken by a SAW to Attack Jurong Island (After: [24])

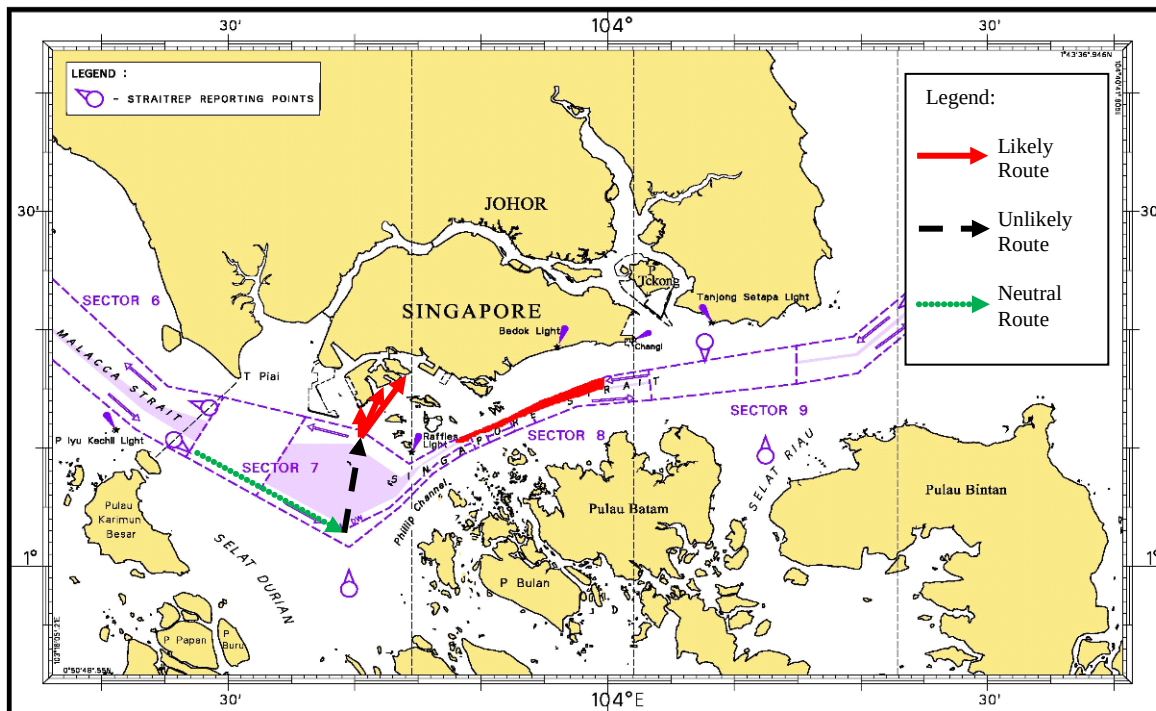


Figure 6. The West-to-East Routes Likely to Be Taken by a SAW to Attack Jurong Island (After: [20])

5. Sensors, Sensor Platforms, and Fusion Centers

This section describes the organic sensors, sensor platforms, and fusion centers used by Singapore to achieve MDA in the Singapore Strait. These sensing and data-processing resources are used in this thesis as elements of sensor network architectures. A brief description of the use of open-source shipping databases and some future sensor platforms is included here for completeness.

a. Maritime Port Authority of Singapore

The Maritime and Port Authority of Singapore (MPA) [30] is responsible for the safe navigation of ships within port waters and the Strait. It closely monitors vessel movements in these waters through two port operations control centers (POCCs), located at Tanjong Pagar Complex (POCC1) and PSA Vista (POCC2). POCC1 monitors vessel traffic in eastern port waters, while POCC2 is responsible for traffic in the western sector and Singapore Strait.

Both centers employ the multi-radar Vessel Traffic Information System (VTIS) to track and monitor shipping of up to 5,000 vessels in real-time in the Strait and port waters. Under an International Maritime Organization (IMO)-approved, mandatory ship-reporting system for the Malacca and Singapore straits (STRAITREP), ships larger than 300 gross tons and all passenger vessels must report to the coastal Vessel Traffic Service (VTS) authorities when approaching Horsburgh Lighthouse in the east or Tg Piai (southwest of Johor) in the west. Having received the vessels' reports, the VTIS starts continuous monitoring of their movements into port.

The MPA has also set up automatic identification systems (AIS), transponder base stations to enable its control centers to automatically receive ship identities and positions transmitted from transponders carried aboard. These base stations are located along the southern coast of Singapore to cover the Singapore Strait and port waters. For small vessels that are not required to carry AIS transponders, the MPA and enforcement agencies have developed and implemented a harbor-craft transponder system (HARTS) to give small vessels the ability to transmit craft identity, position, course, and speed.

b. Fearless Class Patrol Vessels

Republic of Singapore Navy (RSN) patrol vessels (PVs) (Figure 7) patrol the Singapore Strait daily to enhance the coastal security of Singapore.



Figure 7. Photograph of a Patrol Vessel (From: [31])

The sensors [32] a PV carries include:

- (1) Radars
 - Surface search and fire control: Elta EL/M-2228(X); I-band
 - Navigation: Kelvin Hughes 1007; I-band
- (2) Electro-Optical
 - Elbit MSIS optronic director
- (3) Sonar
 - Hull-mounted Thomson Sintra TSM 2362 Gudgeon; active attack; medium frequency
- (4) Electronic countermeasures
 - Electronic support measures: Elisra NS-9010C; intercept

c. Coastal Patrol Craft

The Police Coast Guard (PCG) (Figure 8) enforces the law, maintains order in Singaporean territorial waters, and acts to prevent and detect crime. Of interest is the coastal-patrol squadron, which operates the newly replaced PH-class coastal-patrol craft (CPC) to secure the sea passages in Singaporean waters and ensure safe passage for legitimate users. The new boats may be equipped with enhanced radar and electro-optical surveillance systems [33].



Figure 8. Photograph of a Coastal Patrol Craft (From: [34])

d. Maritime Patrol Aircraft

The F50 Maritime Patrol Aircraft (MPaA) (Figure 9) is jointly operated by the RSN and Republic of Singapore Air Force (RSAF). Its mission is to provide maritime air surveillance to protect the seaward defense and sea lines of communication. [31]. Together with counterparts from Malaysia, Indonesia, and Thailand, the F50 routinely conducts sorties over the Singapore Strait and is currently involved in “eyes in the sky” joint maritime air surveillance of the Malacca Straits. A maritime-patrol aircraft would be equipped with, presumably, surveillance radar and an electro-optical sensor.



Figure 9. Photograph of a Maritime Patrol Aircraft (From: [31])

e. Fusion Centers

Depending on the DSN architecture, there can be one or many data-fusion centers within a network, as discussed in the next chapter. At this juncture, it suffices to know that each maritime agency (i.e., MPA, RSN, RSAF, or PCG) has its own data-fusion center. For MPA, the fusion center is the POCCs, while for the RSN, RSAF, and PCG, fusion centers are assumed to reside in their respective headquarters (HQs). Over and above these fusion centers, the Singapore Maritime Security Center (SMSC) was set up in 2009 to advance interagency cooperation among Singapore's maritime agencies.

f. Open-Source Databases

Various databases, both commercial and open source, contain voluminous information that could aid in profiling, identifying, and preempting a SAW. Some of these databases include Lloyd's List Intelligence, the International Chamber of Commerce, AISLive, and MaritimeTraffic.

g. Future Sensors

Autonomous aerial, surface, and underwater vehicles, equipped with radar, sonar, and electro-optical sensors, can be deployed to patrol the Singapore Strait and augment current maritime-security resources. While having more sensors presumably helps increase MDA, data overload or excessive redundancy can lead to wastage. Furthermore, prudence refrains from adding more “vehicles” to the already congested and shallow waters of the Singapore Strait. Space surveillance via satellite is another area to be explored. While satellite offers a much bigger surveillance footprint, using it solely for the monitoring the Singapore Strait may not be a cost-effective option. These sensors are mentioned for awareness purposes and will not be discussed or used further in the remainder of this thesis.

6. Sensor Coverage

A sensor-coverage analysis is conducted to determine the level of sensor coverage effected by the sensors and sensor platforms used in this research. The level of sensor coverage is assessed in terms of time, space, and the number of sensors involved. The TSS in the Singapore Strait is subdivided into 10 sectors (Figure 10) and, for a typical day, each sensor platform is assumed to undertake a predetermined sector-patrol profile in the Strait. The detection ranges of the sensors used are adopted from open literature and are typical for the operating environment in the Singapore Strait. A Monte-Carlo simulation of 50 runs over 30 days is carried out. To add realism to the sensor and sensor-platform operating profiles, sensor failures (assumed to be one failure per day, with a turnaround of two hours) are factored into the simulation. Details of the sensor-coverage simulation are in Appendix A.

The results of the sensor coverage simulation are illustrated in Figure 11. Jurong Island lies in Sector 73. As can be seen, the minimum and the maximum numbers of available sensors in that sector are three and six, respectively. These figures will then be used in the simulative study in Chapter V for sensor network architecture performance evaluation.

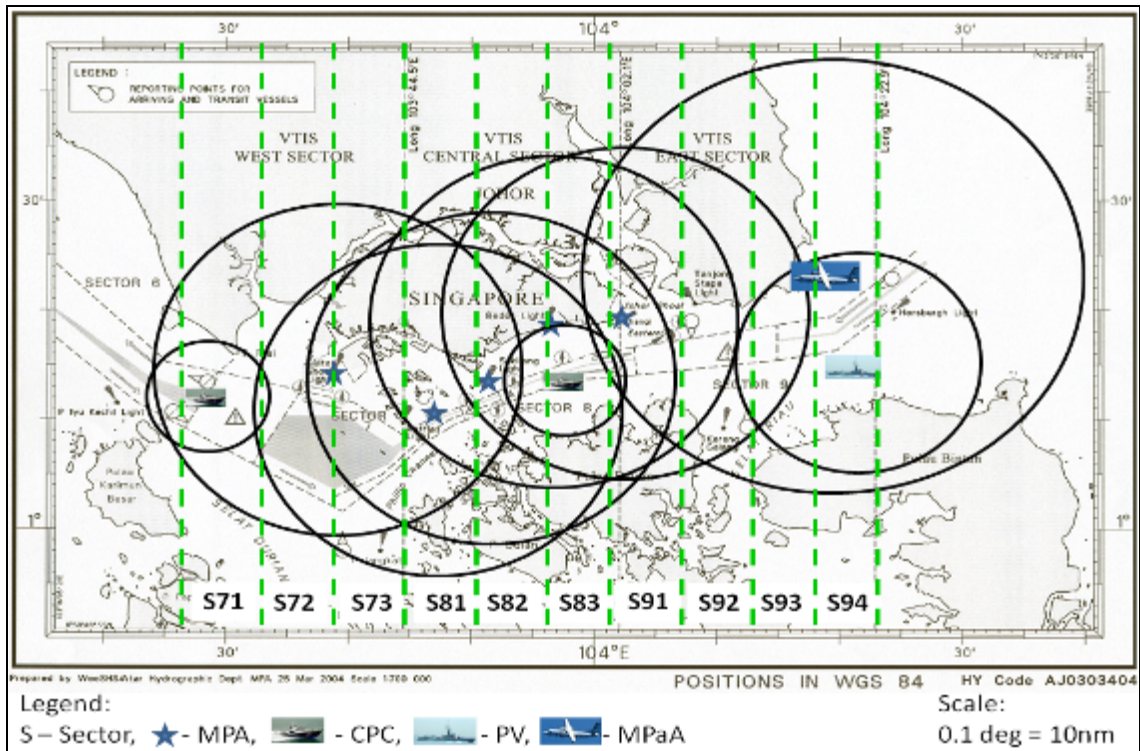


Figure 10. An Instance of the Sensor Platform's Disposition and Sensor Coverage Areas (After: [35])

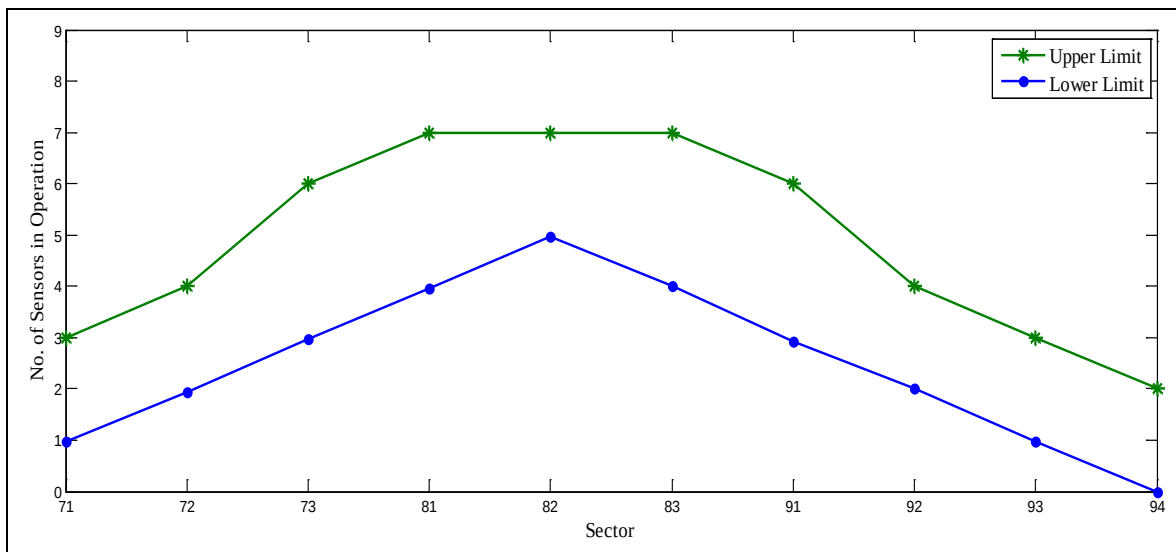


Figure 11. Average Number of Operational Sensors in Each Sector of the Singapore Strait in a Day

D. NEEDS ANALYSIS

The needs analysis layer is built upon the output of ‘Define Missions’ in the mission-analysis layer of the SoSADP (Figure 1). The purpose of the needs analysis is to ascertain which functions the sensor network must perform and to develop measures of effectiveness (MOEs) for the level of performance the sensor network must attain to support the mission.

1. Sensor-Network Needs

The mission of the sensor network in this research is to provide early and accurate identification of a SAW launching an attack on Jurong Island. The mission is considered achieved if the SAW is identified before reaching its intended destination (i.e., Jurong Island). The interdiction of a SAW is beyond the scope of this research, but the outcome of this research would provide valuable insights for the design of the response against the SAW.

2. Measures of Merit

The relationships between DF performance and military effectiveness must be properly understood in order to develop measures and models that relate them. The Military Operations Research Society [36] has recommended a hierarchy of measures that relate performance characteristics of command, control, and communications (C3) systems (including fusion) to military effectiveness (Table 3). Their recommendations provide a quantified measure of merit on how improved information accuracy, timeliness, and content, enabled by sensors and fusion processes, maps to military effectiveness. MOFEs, MOPs, and dimensional parameters are not used in this work, but their definitions are included for completeness.

Table 3. Four Categories of Measures of Merit (From: [36])

Measure	Definition	Typical Examples
Measures of force effectiveness (MOFEs)	Measure of how a C3 system and the force of which it is a part (sensors, weapons, C3 system) perform military missions	Outcome of battle; cost of system; survivability; attrition rate; exchange ratio; weapons on targets
Measures of effectiveness (MOEs)	Measure of how a C3 system functions within an operational environment	Target nomination rate; timeliness of information; accuracy of information; warning time; target leakage; countermeasure immunity; communications survivability
Measures of performance (MOPs)	Measures closely related to dimensional parameters (both physical and structural), but measure attributes of behaviour	Detection probability; false-alarm rate; location transmission; communication delay; sensor
Dimensional parameters	The properties or characteristics inherent in the physical entities whose values determine system behaviour and the structure under question, even when not operating	Signal-to-noise ratio; operations per second; aperture dimensions; bit-error rates; resolution; sample rates; anti-jamming margins; cost

From the mission analysis, it is clear that the MOEs for the DSN are the accurate identification of a SAW with the intention of crashing into Jurong Island and the earliest impact warning the DSN can provide. In the next section, the MOP will be developed after a requirement analysis is conducted to support this MOE.

E. REQUIREMENTS ANALYSIS

The requirements analysis is built on the needs analysis and comprises an operational-requirements analysis, a functional analysis, and a non-functional analysis. The goal of the requirements analysis is to generate MOP(s).

1. Operational Requirements Analysis

Operational requirements are derived directly from the problem statement. The top-level need is to provide accurate early warning of a SAW traveling east to west on the TSS, abruptly turning, and accelerating towards Area A1 of Jurong Island at full speed, which is deemed the worst-case scenario. Hence, the operational requirement for the sensor network mandates that a SAW be identified early and with high confidence before it reaches Area A1.

2. Functional Analysis

Functional analysis involves identifying the top-level functions of the sensor network and performing functional decomposition, as captured in a functional-requirements diagram [21].

a. Functional Requirements

The use case diagrams in this section help illustrate what the sensor network is doing at any given time. A complete set of use case diagrams captures all the functions the sensor network should perform [37].

Figure 12 displays the use case diagram for the sensor-network problem description. The sensor platforms (i.e., MPA radar tower, PV, CPC, and MPA) are depicted as actors performing their own SAW detection and tracking. The respective HQs (i.e., MPA, RSN, PCG, and RSAF) of the sensor platforms may or may not carry out the tracking of the SAW. Depending where data fusion is conducted, the fusing of data and classifying of vessels may be performed either at the sensor platforms, the respective HQs of the sensor platforms, or the SMSC. Like the HQs, SMSC may or may not track the SAW.

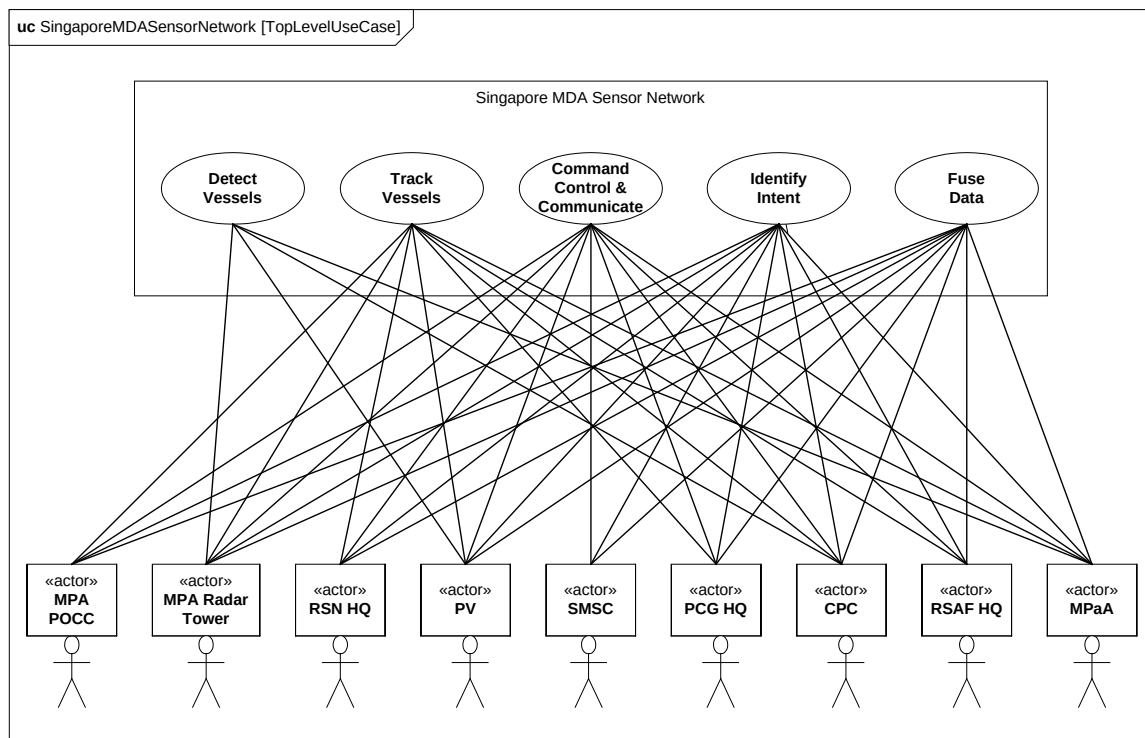


Figure 12. Singapore MDA Sensor Network, Use Case Diagram

b. Functional Decomposition

The requirements diagram (Figure 13) captures the functional decomposition of the sensor network. Each Tier 1 function is further decomposed into sub-functions to provide better resolution concerning what is required of the sensor network. For example, C3 is deliberately treated as a separate function, but asserts command and control via the communications (Comm) sub-function.

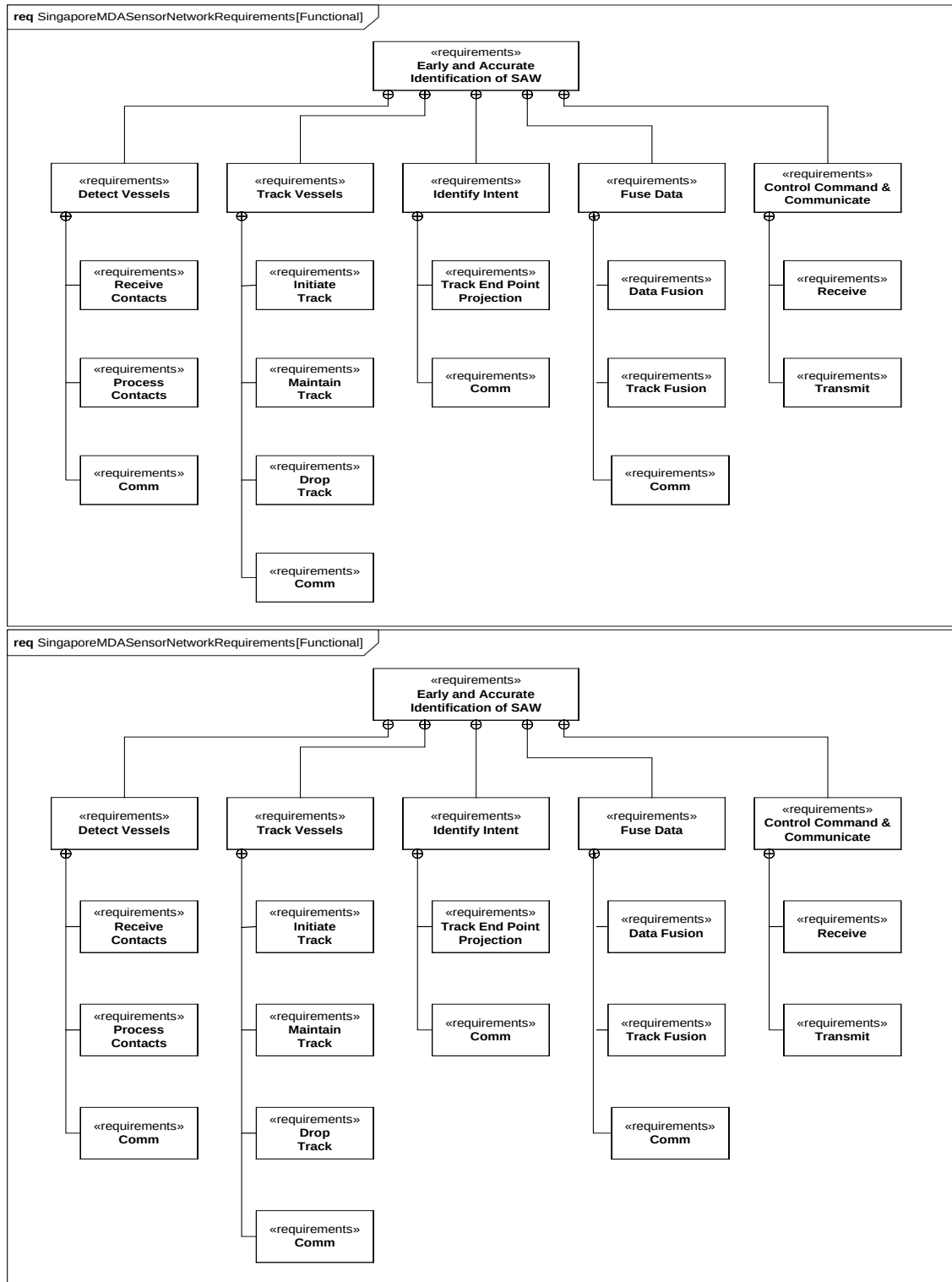


Figure 13. Singapore MDA Sensor Network Requirements Diagram

3. Measures of Performance

The mission of the sensor network is to provide accurate identification of SAW intent and impact warning well before it reaches Jurong Island. In order to achieve these MOEs, one possible measure of performance (MOP) would be how fast the sensor fusion algorithms can be executed. This MOP will not be explored further for the purpose of this thesis, but it is assumed to be identical regardless of the DSN architecture chosen and sensor-fusion algorithm used.

F. SUMMARY

This chapter discusses the relationships between maritime domain security, maritime domain awareness, network-centric operations, and data fusion. It is followed by an elucidation of the problem statement for this research and the mission analysis. The mission analysis details the threats and their possible attack approaches, rationalizes the potential targets, and describes the available sensors, sensor platforms, and fusion centers, and the physical environment in which the mission is to be executed. As a result of the needs and requirements analyses, the MOE and MOP are developed.

THIS PAGE INTENTIONALLY LEFT BLANK

III. DISTRIBUTED SENSOR NETWORK ARCHITECTURES

A. INTRODUCTION

The problem statement, mission analysis, needs analysis, and requirements analysis now lead to the generation of alternative sensor network architectures. The existing sensors and sensor platforms described in Chapter II form the various DSN architectures. The key differences between the architectural alternatives depend on “where” the fusion centers are placed in the network and the data type (i.e., raw data or processed tracks) to be fused at the fusion centers. In the following sections, three broad DSN architectures are presented, followed by a discussion of how the current suite of sensors, sensor platforms, and fusion centers are embedded into each of the three architectures.

B. FUSION ARCHITECTURES

Data fusion involves the integration of sensors, data processing, and estimation. It demands organization, whether structured or unstructured, between the sensors and the sensor platforms that acquire and process the data. This organization is dictated by the sensor network architecture, and there are three main types.

1. Centralized Data Fusion Architecture

For centralized data fusion architecture [38], there is only one fusion center, serving as a central processor that collects all raw data from the different sensors (Figure 14a). Though this research does not discuss the dissemination of information from the fusion center, it should be understood that data or information flows do not necessarily end there.

The main advantage of centralized fusion is it offers the best theoretical performance in terms of minimizing errors and data loss. The disadvantages, however, are many. Communications bottleneck is possible because all sensors vie to communicate directly with the fusion center, which ironically leads to data loss and reduced performance. Vulnerability is a concern because failure in the central processor

or of communications links could cripple the entire sensor network. Finally, centralized architecture affords poor modularity and scalability, as any addition of sensors will usually require more computing power from the fusion center and a higher network-communications bandwidth.

2. Decentralized Data Fusion Architecture

Decentralized data fusion architecture [38–39] (Figure 14b) features an organization of sensors directly opposite to that of centralized data fusion architecture. For this architecture, each sensor node (i.e., sensor platform) takes on the role of fusing data captured by its own sensors and data from the rest of the sensor nodes connected to it, rather than relying on a single, central fusion center. The advantages of the decentralized data fusion architecture are the increased survivability of the sensor network (by eliminating the single point failure of the single central processor), improved modularity and scalability (by making local changes to the network), and the absence of constraints imposed by a central computing power.

The decentralized data fusion architecture is, however, not without shortfalls. One serious problem is redundant data transmission. When this occurs, data may be reused (i.e., double counted) and result in fused estimates being overly optimistic.

3. Hybrid Data Fusion Architecture

A hybrid sensor network architecture [38, 40] (Figure 14b), also referred to as hierarchical architecture, involves both centralized and decentralized data fusion schemes. In general, the motivation for using decentralized data fusion is to reduce computational workload and communication demands, while the incentive for using centralized data fusion is better accuracy. The hybrid data fusion offers a middle ground, combining the advantages of the centralized and decentralized architectures without some of their disadvantages.

In a hybrid architecture, there are often several hierarchical levels, where the top level contains a single, centralized fusion center and the lower levels contain several

decentralized (i.e., local) fusion nodes. Depending on the organization of the sensors and sensor platforms, each sensor platform may receive data from a cluster of sensors or from other sensor platforms.

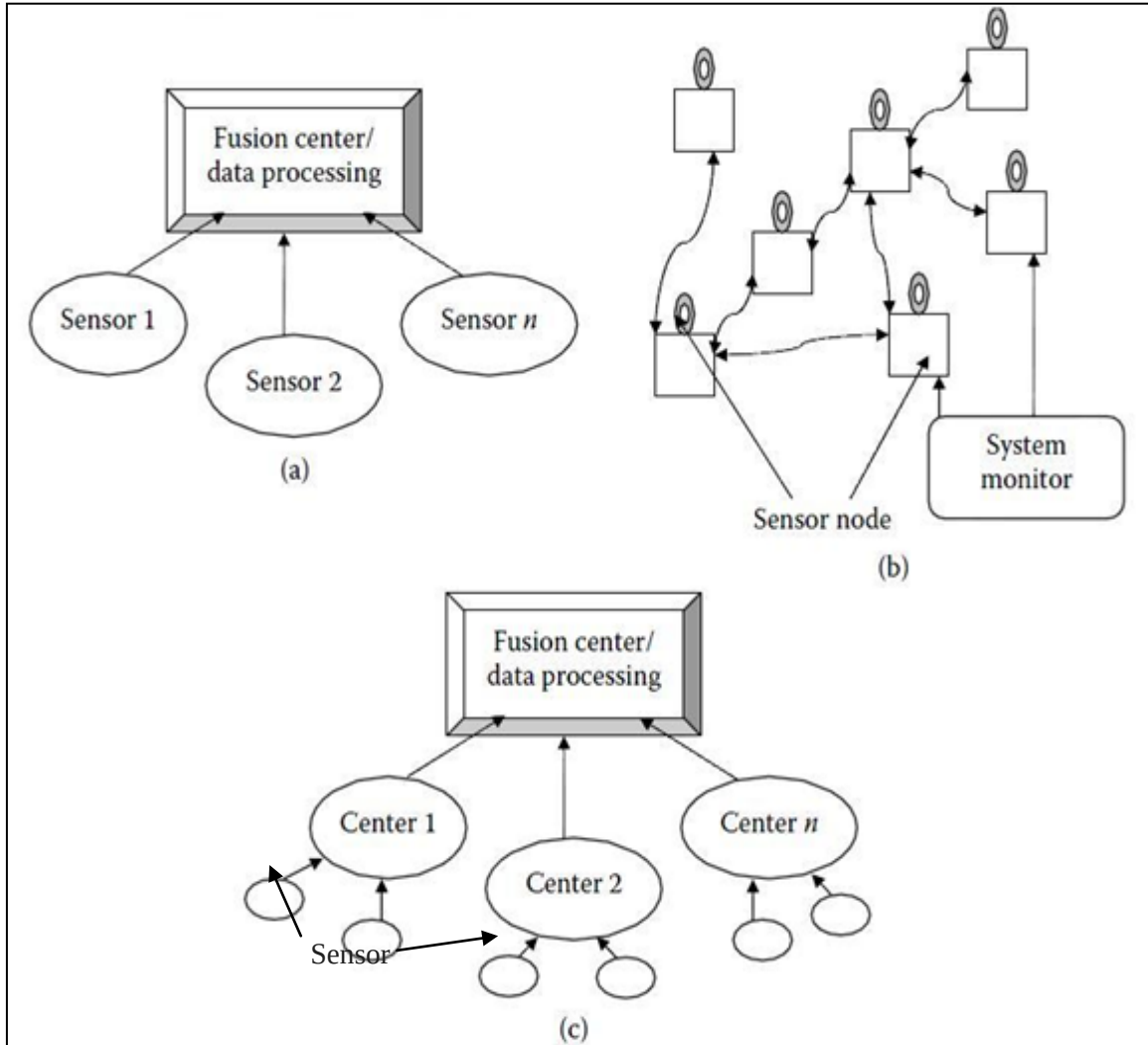


Figure 14. Three Sensor Data-fusion Architectures: (a) Centralized Processing at the Fusion Center, (b) Decentralized Processing at Each Node-processing Unit, and (c) Hierarchical Processing. (After: [40])

C. ARCHITECTURE ALTERNATIVES

In this section, different architectures are developed, taking into consideration existing sensors, sensor platforms, and fusion centers used by Singapore, as described in the previous chapter. Figure 15 illustrates a simplified command and control (C2) structure within and between the various maritime-security agencies. The vertical divider depicts the intra-agency C2 structure, and the horizontal divider shows the chain-of-command equivalency between the agencies.

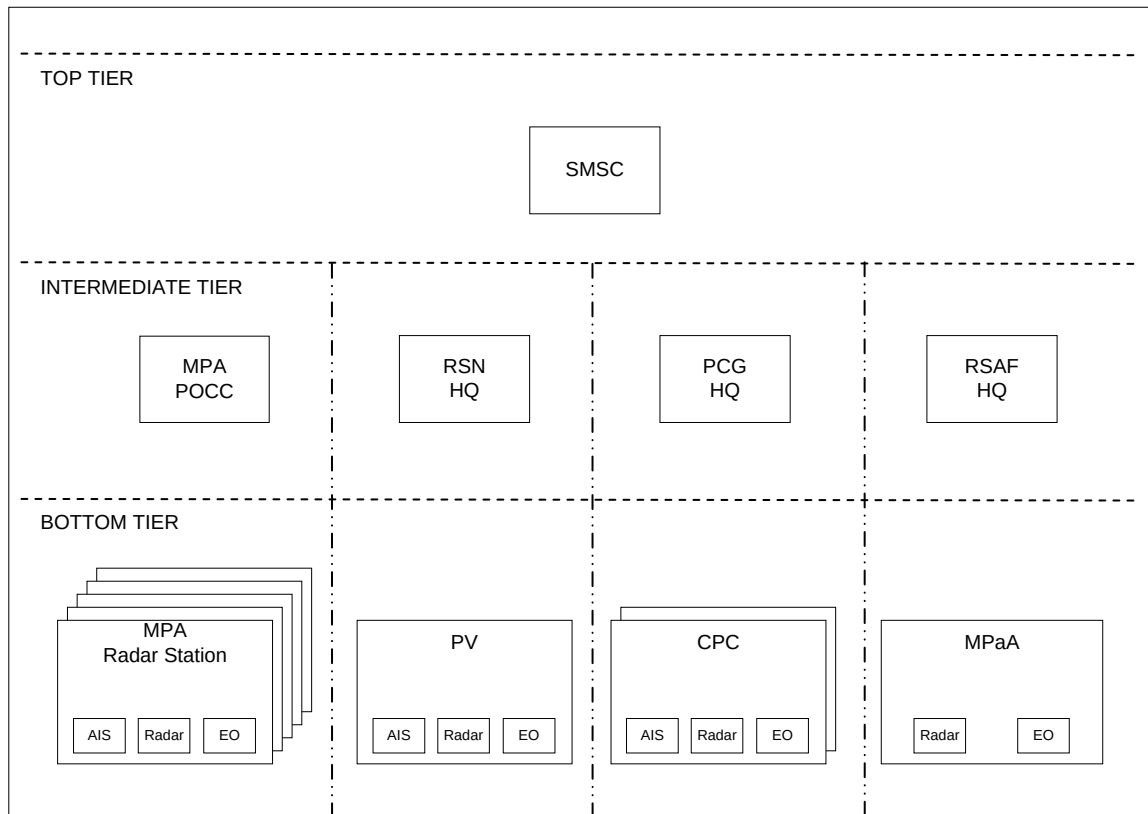


Figure 15. Command and Control (C2) Structure within and between the Various Maritime Security Agencies

1. Centralized Architecture

In a centralized DSN architecture based on C2 structure (Figure 15), the SMSC is assigned as the central fusion center. Raw data captured by the bottom tier is sent to the SMSC via the intermediate tier. For this architecture, the intermediate tier does not carry out fusion of data, but simply forwards it to the SMSC for fusion. On completion of data fusion by the SMSC, the fused track (i.e., processed data) is disseminated to the intermediate and bottom tiers (Figure 16).

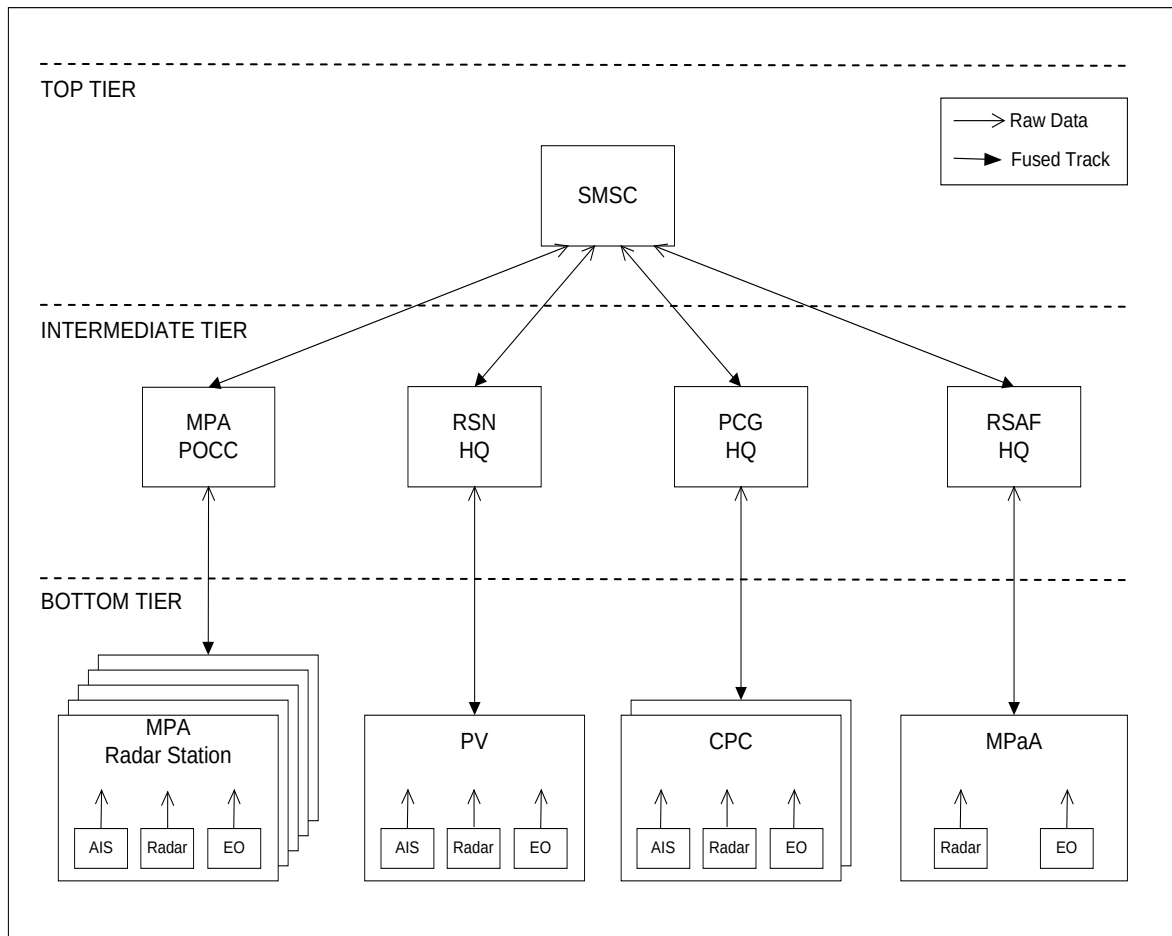


Figure 16. Data Flow for Centralized Architecture

2. Decentralized Architecture

For decentralized DSN architecture, each individual sensor platform in the bottom tier is a fusion node. As illustrated in Figure 17, each sensor platform fuses raw data captured by its own sensors and by other sensor platforms. The fusion algorithm used by the sensor platforms is assumed to be identical. Consequently, given the same raw data input at each sensor platform, the fused track will be identical. Hence, dissemination of the fused track is simplified, as there is no requirement for cross dissemination (e.g., the fused tracks from a PV do not need to be sent to PCG HQ). The sensor platform is only required to disseminate the fused track to its own chain of command for its awareness. The SMSC receives multiple identical tracks, which is a redundancy feature expected of the decentralised architecture. No processing is required by the intermediate and top tiers in this architecture.

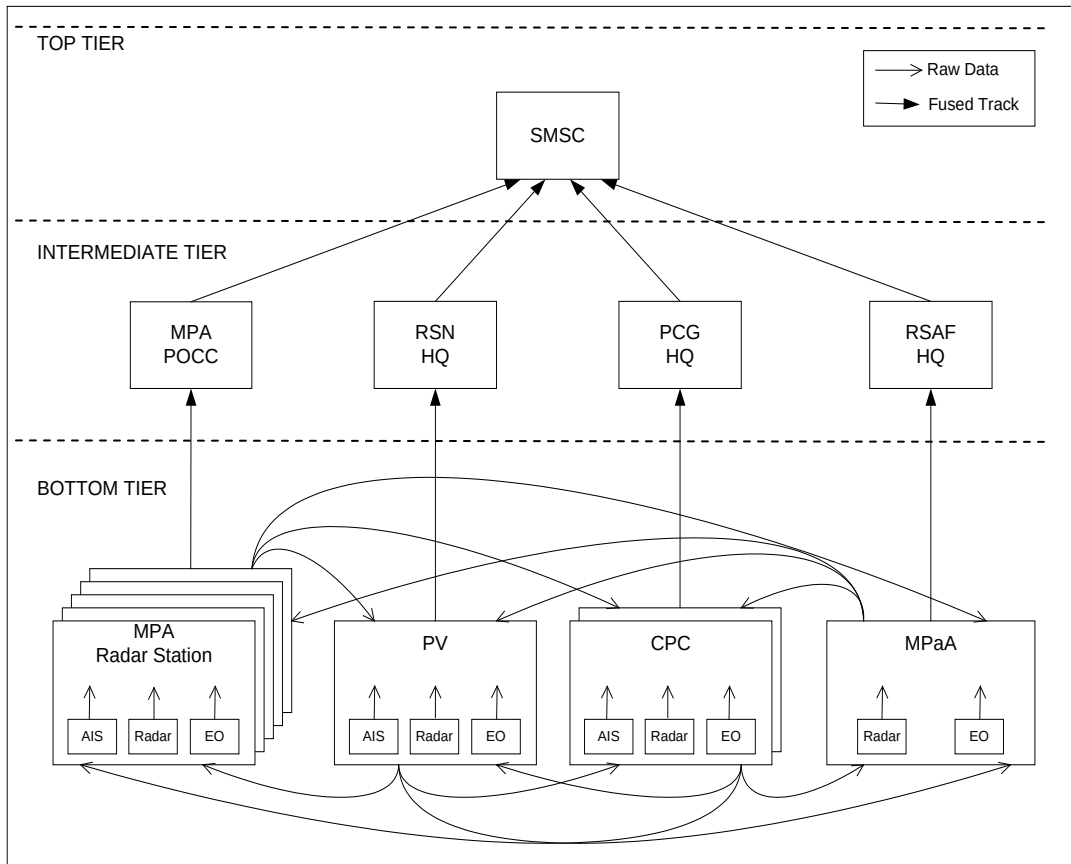


Figure 17. Data Flow for Decentralized Architecture

3. Hybrid Architecture

The hybrid DSN architecture, unlike the previous two, has both data fusion (i.e., fusion of raw data) and track-to-track fusion (i.e., fusion of tracks), carried out at different tiers. Each individual sensor platform in the bottom tier first fuses the raw data captured by its own sensors, then disseminates the fused track for fusion at either the intermediate or top tier. In this work, the SMSC is selected to carry out track-to-track fusion. The choice between the intermediate tier and the top tier for track fusion, as will be seen later, results in very little, if any, difference in this simulative study. However, since the SMSC is used as the fusion center for the centralized architecture, for consistency the SMSC is also assumed to carry out the “final” fusion for the hybrid architecture. As illustrated in Figure 18, every sensor platform fuses raw data captured by its own sensors. The fused track is sent to the SMSC via its own chain of command. The SMSC pools the individually fused tracks and carries out track-to-track fusion. The final fused track is then disseminated to the chains of command for their awareness. No processing is required by the intermediate tier in this case.

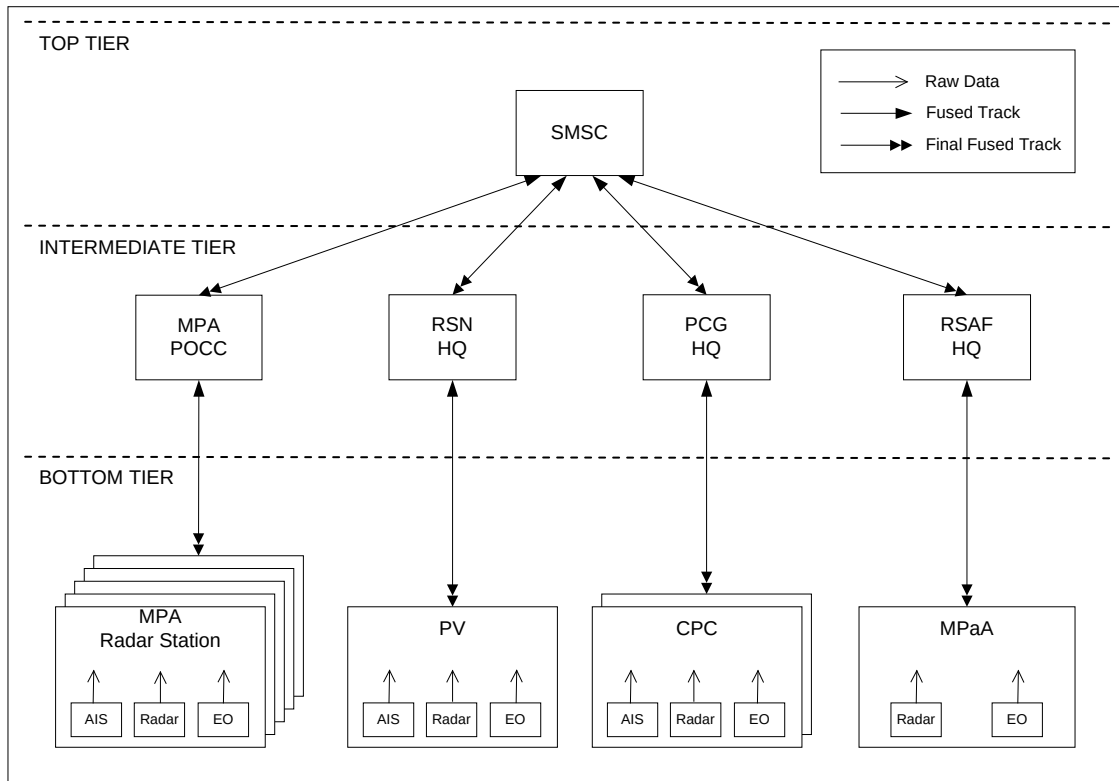


Figure 18. Data Flow for Hybrid Architecture

D. SUMMARY

In this chapter, three types of DSN architectures—centralized, decentralized, and hybrid—are described. These architectures are formed using the existing sensors, sensor platforms, and fusion center used by Singapore. The choice of data-fusion algorithm to be employed in each architectural alternative is discussed in the next chapter. Together, they are evaluated using modeling and simulation, discussed in Chapter V, to determine which sensor-network architecture affords the earliest warning and best identification accuracy of a SAW intending to attack Jurong Island.

IV. SENSOR FUSION ALGORITHMS

A. INTRODUCTION

In their revision of the Joint Directors of Laboratories (JDL) Data Fusion Group's data fusion model, Steinberg *et al.* offer a succinct definition of data fusion as “the process of combining data or information to estimate or predict entity states” [41]. In the following sections, the focus is on the sensor fusion algorithms used in this research to combine sensor data to estimate the SAW state. A detailed derivation of the algorithms is not provided here, as these algorithms can be found in most sensor-fusion (e.g., [38], [40], [43] and [46]).

B. DATA FUSION MODELS

Many existing data fusion models [40] help generalize and guide the data fusion process. Referencing the JDL fusion model (Figure 19), the process of data fusion is divided into levels or hierarchies. At the lower levels (levels 0-2), the processes are concerned with track formation, identity, or estimated information and the fusion of information from several sources. At the higher levels (levels 3-4), the process is concerned with the extraction of “knowledge” or decisional information.

In this work, Level 1 deals with fusion of ship kinematics, which involves fusion of local information to determine the position, velocity, and acceleration of a detected ship. Level 2 seeks to discover the relationship between the ship and its navigational pattern. Level 3, based on predictions about the future situation, is an estimation of danger (i.e., threat assessment), as to whether a vessel is aiming for Jurong Island (i.e., a SAW). Level 4 forms the process of dynamic resource reallocation. For evaluation purposes, Level 4 is predetermined under sensor-coverage analysis in Chapter II and used as fixed input for the simulative study discussed in the next chapter. In other words, the commitment and allocation of the number of sensors is classified into best and worst-case scenarios up front, and they form the basis for evaluation for each DSN architecture.

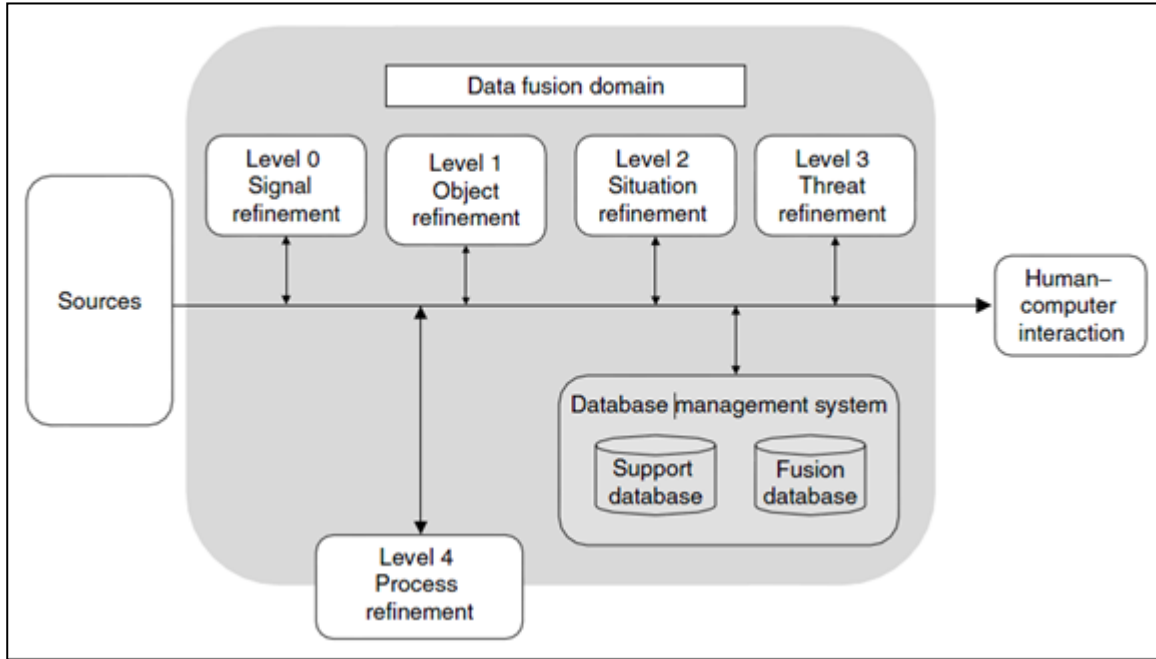


Figure 19. Joint Directors of Laboratories Data Fusion Model (From: [42])

C. STATE MODEL

In this research, ship motion is simulated using a constant-velocity model (CVM). The ship's acceleration can be accounted for in the process noise. Therefore, the explicit use of a constant acceleration model is not necessary. In this model, the state (or state vector) at time t_{k+1} is obtained estimated according to

$$\mathbf{x}(k+1) = \mathbf{F}(k)\mathbf{x}(k), \quad (1)$$

where

$$\mathbf{x}(k+1) = \begin{pmatrix} x \\ \dot{x} \\ y \\ \dot{y} \end{pmatrix} \text{ with } \begin{pmatrix} x: \text{x-component of the ship's position vector} \\ \dot{x}: \text{x-component of the ship's velocity vector} \\ y: \text{y-component of the ship's position vector} \\ \dot{y}: \text{y-component of the ship's velocity vector} \end{pmatrix} \text{ at time } t_{k+1},$$

$$\mathbf{F}(k) = \begin{pmatrix} 1\Delta T & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & \Delta T & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \text{ the state transition matrix,}$$

ΔT is the time step between time t_k and time t_{k+1} , for $k = 0, 1, 2, 3, \dots$

D. MEASUREMENT MODEL

The measurement (observation) at time t_k , $\mathbf{z}(k)$, is modeled according to

$$\mathbf{z}(k) = \mathbf{H}(k)\mathbf{x}(k) + \mathbf{v}(k), \quad (2)$$

where

$$\mathbf{z}(k) = \begin{pmatrix} x \\ y \end{pmatrix} \text{ with components } x \text{ and } y,$$

$$\mathbf{H}(k) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \text{ is the measurement matrix, and}$$

$\mathbf{v}(k) \sim N(0, \mathbf{R}(k))$, the measurement noise, is assumed to be Gaussian, temporally uncorrelated, and of zero-mean and covariance $\mathbf{R}(k)$.

Note that observations (range and bearing) made by radar and electro-optical devices expressed in a polar coordinate system are converted to those expressed in a Cartesian coordinate system before filtering or fusion takes place.

E. ESTIMATION FOR DATA FUSION

Estimation is the “single most important problem” [43] in data fusion. Sensors make observations (i.e., take measurements) for the purpose of producing an estimate of the true state of the environment being observed. The better the estimation, the closer the estimate will be to the truth. In addition, an estimate is considered good if its uncertainty is small.

Given time constraints, it was not possible to investigate all the available data fusion algorithms for this research. Taking into consideration that the chosen state and sensor models are linear and well defined, and that both the process and observation noises are assumed to be Gaussian, temporally uncorrelated, and of zero-mean, the Kalman filter (KF) and information filter (IF) are chosen from among other alternatives (Bayesian filter, particle filter, extended Kalman filter, etc.). Following are the KF and IF algorithms for state and uncertainty estimation, using a set of observations captured by the sensors.

1. Kalman Filter Algorithm

The Kalman filter (KF) algorithm, extracted from [43], is now stated without proof. Derivations of the KF equations can also be found in many books on KF[40,44-45]. The state is assumed to evolve in time according to Equation 1. Observations of this state are modeled according to Equation 2. The assumptions previously made about the noises are applied.

The predicted state $\hat{\mathbf{x}}(k|k-1)$ and covariance matrix $\mathbf{P}(k|k-1)$ at time t_k are obtained according to

$$\hat{\mathbf{x}}(k|k-1) = \mathbf{F}(k) \hat{\mathbf{x}}(k-1|k-1), \quad (3)$$

$$\mathbf{P}(k|k-1) = \mathbf{F}(k)\mathbf{P}(k-1|k-1)\mathbf{F}^T(k) + \mathbf{Q}(k), \quad (4)$$

where

^{T} denotes the transpose of a matrix,

$$\mathbf{Q}(k) = \begin{pmatrix} \sigma_x^2 & 0 & 0 & 0 \\ 0 & \sigma_{\dot{x}}^2 & 0 & 0 \\ 0 & 0 & \sigma_y^2 & 0 \\ 0 & 0 & 0 & \sigma_{\dot{y}}^2 \end{pmatrix} = \begin{pmatrix} \Delta T^2 \sigma_x^2 & 0 & 0 & 0 \\ 0 & \sigma_{\dot{x}}^2 & 0 & 0 \\ 0 & 0 & \Delta T^2 \sigma_y^2 & 0 \\ 0 & 0 & 0 & \sigma_{\dot{y}}^2 \end{pmatrix}, \quad \text{the}$$

process noise matrix. The process noise is assumed to be Gaussian, temporally uncorrelated, and of zero-mean and variances, σ_x^2 and σ_y^2 of the x- and y-component of the ship's position vector, respectively, and $\sigma_{\dot{x}}^2$ and $\sigma_{\dot{y}}^2$ are the variances of the x- and y-component of the ship's velocity vector, respectively.

The measurement $\mathbf{z}(k)$ that is available at time t_k is used to update the state and the covariance matrix. The updated state $\hat{\mathbf{x}}(k|k)$ and covariance matrix $\mathbf{P}(k|k)$ at time t_k are computed according to

$$\hat{\mathbf{x}}(k|k) = \hat{\mathbf{x}}(k|k-1) + \mathbf{W}(k)[\mathbf{z}(k) - \mathbf{H}(k)\hat{\mathbf{x}}(k|k-1)], \quad (5)$$

$$\mathbf{P}(k|k) = \mathbf{P}(k|k-1) - \mathbf{W}(k)\mathbf{S}(k)\mathbf{W}^T(k), \quad (6)$$

where the gain matrix $\mathbf{W}$ and innovation variance matrix $\mathbf{S}$ are given by:

$$\mathbf{S}(k) = \mathbf{H}(k)\mathbf{P}(k|k-1)\mathbf{H}^T(k) + \mathbf{R}(k),$$

$$\mathbf{W}(k) = \mathbf{P}(k|k-1)\mathbf{H}^T(k)\mathbf{S}^{-1}(k),$$

and $\mathbf{R}(k)$ is the measurement noise covariance matrix.

The KF is recursive (Figure 20). The process starts with an initial estimate of the state and the covariance matrix, propagates them, and updates them when an observation becomes available.

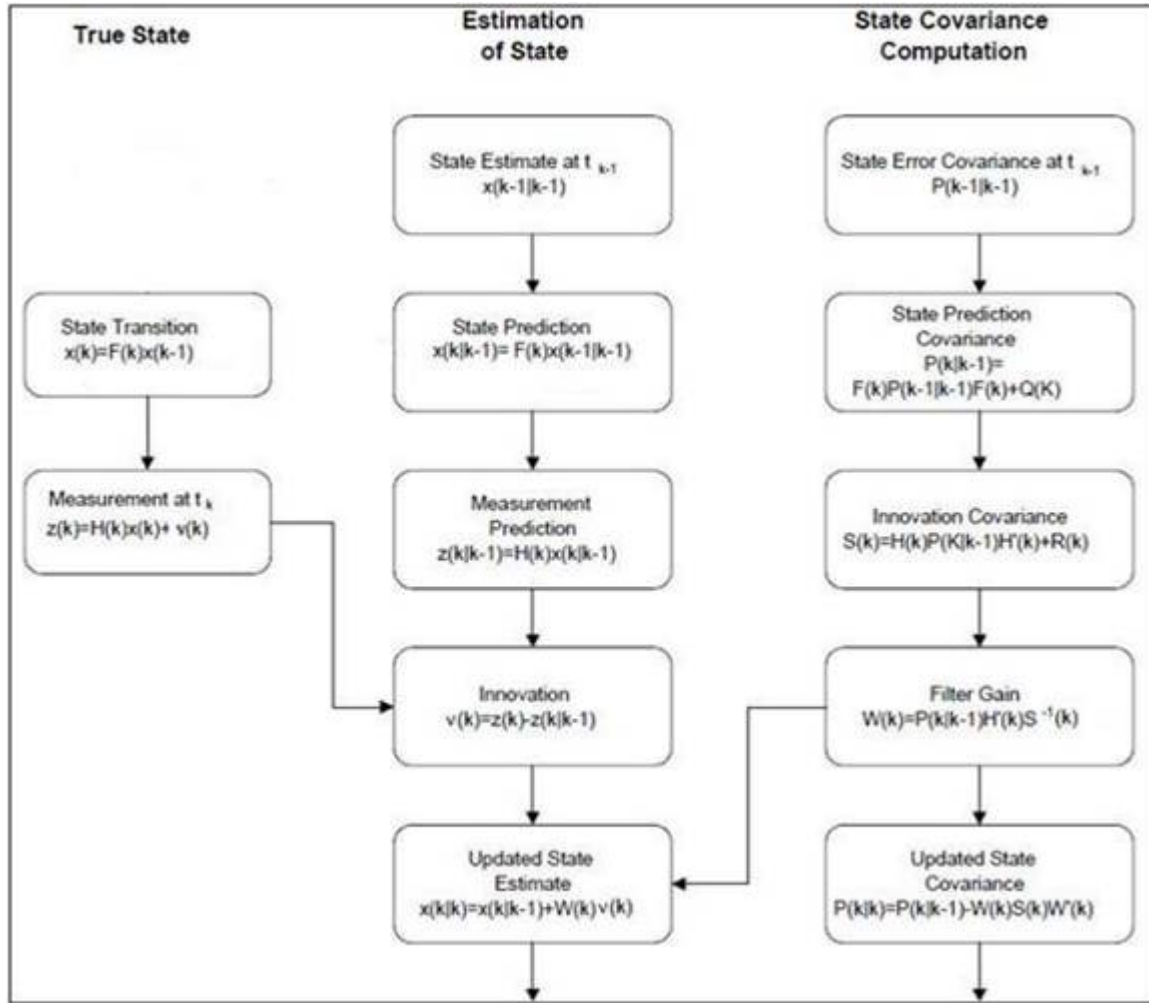


Figure 20. Block diagram of the Kalman filter cycle (After: [44])

2. Information Filter Algorithm

Whereas the KF deals with the estimation of the state vector $\mathbf{x}$ and its covariance matrix $\mathbf{P}$, the information filter (IF) algorithm, extracted from [43], deals with the information state vector $\mathbf{y} = \mathbf{P}^{-1} \mathbf{x}$ and the information matrix $\mathbf{Y} = \mathbf{P}^{-1}$.

The predicted information state vector $\hat{\mathbf{y}}(k|k-1)$ and the predicted information matrix $\mathbf{Y}(k|k-1)$ at time t_k are obtained according to

$$\hat{\mathbf{y}}(k|k-1) = [1 - \mathbf{\Omega}(k)\mathbf{G}^T(k)]\mathbf{F}^T(k)\hat{\mathbf{y}}(k-1|k-1), \quad (7)$$

$$\mathbf{Y}(k|k-1) = \mathbf{M}(k) - \mathbf{\Omega}(k)$$

3. Comparing KF and IF

A comparison of the updated algorithms between KF (equations 5 and 6) and IF (Equations 9 and 10) highlights the simplicity of the IF update over the KF update. In information form, the update stage is a straight addition of information from an observation to a prediction. This simplicity gives the IF its advantage over KF in multisensor estimation problems. Any change to the number of sensors is merely an addition or subtraction of information. However, the simplicity of the update stage of the information filter comes at the cost of increased complexity in the prediction stage.

4. Track-to-Track Fusion

The track-to-track fusion algorithm combines track estimates from sensors, making it distinctly different from the KF and IF algorithms, which combine observations (to form tracks) from different sensors. In this thesis, track-to-track fusion is referred to as track fusion, while fusion of observations is called data fusion.

To carry out track fusion, data fusion using either the KF or IF must be executed first. Each sensor generates its own track locally using the KF or IF, and the track is then sent to a fusion center, where it is combined with other tracks to generate a global track estimate. The main advantage of track fusion over data fusion is that sending track information takes up less communications bandwidth, due to its smaller data size. However, due to the fusion of filtered estimates, the global track may result in a poorer global estimate with higher uncertainty. This will be discussed in detail in the next chapter.

The track-to-track fusion (also commonly referred to as track fusion) algorithm extracted from [46] is stated here without derivation. Prior to fusing the individual track estimates, a track association test is carried out to ensure that the tracks are likely to belong to the same contact of interest. Only when the tracks pass the association test will they undergo the track fusion process.

a. Track Association Test

At each time step, the algorithm uses the estimated state of the target from sensor i , $\hat{\mathbf{x}}^i(k)$, and that from sensor j , $\hat{\mathbf{x}}^j(k)$, and their corresponding covariances $\mathbf{P}^i(k)$ and $\mathbf{P}^j(k)$, respectively. The state estimation errors are assumed to be independent. For simplicity, the time argument will be omitted from here on.

The track association test is passed if

$$D \triangleq (\hat{\Delta}^{ij})^T (\mathbf{T}^{ij})^{-1} (\hat{\Delta}^{ij}) \leq D_\alpha,$$

where

$D_\alpha = \chi_n^2(1 - \alpha)$, the threshold, is determined using a chi-square distribution with n degrees of freedom and confidence level of α ,

$$\hat{\Delta}^{ij} = \hat{\mathbf{x}}^i - \hat{\mathbf{x}}^j,$$

$$\mathbf{T}^{ij} = \mathbf{P}^i - \mathbf{P}^j, \text{ the covariance of } \hat{\Delta}^{ij}.$$

b. Fusion of Estimates

Fusion of two estimates $\hat{\mathbf{x}}^i$ and $\hat{\mathbf{x}}^j$ and their respective covariances $\mathbf{P}^i$ and $\mathbf{P}^j$ results in the estimated track and its covariance:

$$\hat{\mathbf{x}} = \mathbf{P}^j(\mathbf{P}^i + \mathbf{P}^j)^{-1} \hat{\mathbf{x}}^i + \mathbf{P}^i(\mathbf{P}^i + \mathbf{P}^j)^{-1} \hat{\mathbf{x}}^j, \quad (11)$$

$$\mathbf{P} = \mathbf{P}^i(\mathbf{P}^i + \mathbf{P}^j)^{-1} \mathbf{P}^j. \quad (12)$$

F. SUMMARY

This chapter begins with a recap on data fusion and provides a brief discussion of the data fusion model developed by JDL. Next, the state and sensor models, together with the KF, IF and track-to-track fusion algorithms, are stated without derivation. In the next chapter, the materials covered in Chapters III and IV are modeled and simulated using MATLAB to find which sensor network architecture, complemented with a suitable fusion algorithm, produces the earliest and most accurate identification of a SAW intending to strike Jurong Island.

V. SIMULATIVE STUDY

A. INTRODUCTION

In this chapter, the scenario of a SAW navigating the Singapore Straits with the intention of crashing into the oil and chemical terminals on Jurong Island is modeled, followed by the simulation of alternative sensor network architectures and different sensor fusion algorithms. The outcomes of the modeling and simulation are the estimated MOPs and MOEs for each alternative that will be used to determine the best sensor-network architecture and sensor-fusion algorithm pair to deploy against the SAW. This evaluation is a culmination of the SoSADP—the system of systems architecture ranking.

B. MODELING AND SIMULATION

In the real world, many MDS and MDA problems, such as those treated in this research, are impossible to observe and study. Even if it were possible, it would take many resources to set up and conduct actual experimental trials, validation trials, and also re-validation trials. Hence, the use of modeling and simulation (M&S) in this study is the prudent, if not the only, option for understanding and experimenting with the MDA problem.

1. Model Development

MATLAB (Ver. R2010a) software serves as the simulation environment for coding and simulation. The characteristics and attack approach of the SAW, its target on Jurong Island, the sensors and their network topology, and the sensor platforms and fusion centers and their locations, are created according to the problem statement, mission analysis, needs analysis, and requirements analysis in Chapter II.

a. The Ship as a Weapon

In this simulative study, the SAW is the only target (that is, this is a single-target simulation). It starts its passage from the east of the Singapore Strait and cruises in the TSS at 15 knots. At the closest point of approach from the TSS to Area A1

of Jurong Island, the SAW turns and heads towards Area A1 and accelerates to 20 knots. The navigation route and attack approach taken by the SAW is illustrated in Figure 5 in Chapter II.

b. Sensor Topology

The MPA radar stations are modeled as fixed sensors on stationary platforms with a line-of-sight range of 15 nm. Up to five such fixed sensors can be used in the simulation, and their predetermined locations are evenly distributed along the southern coast of Singapore (Figure 10 in Chapter II). Augmenting these fixed sensors are four mobile sensors: two CPCs, one PV, and one MPaA. Though they are mobile, each having its own patrol route, in this simulation, their sensor ranges are extended such that they have complete coverage of the Singapore Strait. This is done to simplify sensor management where there will be more sensors deployed to keep an eye on suspicious vessel behavior. This assumption does not infringe on realism because the interesting comparison here lies in the number of sensors monitoring the SAW (i.e., analyzing whether more sensors would result in earlier warning and more accurate identification of the SAW). In addition, sensor errors are assumed to be identical for all sensors, because the purpose here is to determine the best network architecture and not which types of sensor to use in the network.

c. Target Destination

As identified and explained in Chapter IV, Area A1 of Jurong Island is the most attractive and easiest to strike with a SAW. Its location is presented as a circle, using an estimated latitude and longitude as center and a radius of 0.5 nm. This circle forms the basis for threat-level assessment, which is discussed later in this chapter.

d. Sensor Network Architecture and Fusion Algorithm

In the simulative study, three sensor-network architectures—centralized, decentralized, and hybrid—are modeled in accordance with Figures 16, 17, and 18 in Chapter IV. The sensor algorithms—Kalman filter, information filter, and track-to-track fusion—are applied to the architectures as listed in Table 4.

Table 4. Applying Algorithms to Architectures

Architecture	Algorithm(s) Applied	Purpose / Remarks
Centralized Sensor Network	Kalman Filter and Information Filter	With the assumption that the measurements made are of Gaussian noise with zero mean and constant covariance, the KF and IF will produce the same data fusion results. Hence, this case seeks to validate this (which at the same time proves that the algorithms are implemented correctly).
Decentralized Sensor Network	Information Filter	This case seeks to validate that the decentralized and centralized sensor network architectures produce the same results. Only the IF is applied since both KF and IF produce the same results regardless of sensor network in the context of this thesis. The reason for choosing IF over KF will be discussed later in this chapter.
Hybrid Sensor Network	Information Filter and Track-to-track fusion algorithm	IF is used to carry out data fusion, followed by the implementation of a track-to-track fusion algorithm to carry out track fusion. The results are compared to the other two architectures using IF.

2. Simulation Program Description

The simulation program builds upon the initial MATLAB programs produced by Durrant-Whyte [43]. Many modifications and additions are made to adapt the code, but the functional flow of the program remains largely the same. The simulation program is made up of four major blocks: (1) centralized sensor-network architecture with KF, (2) centralized sensor network architecture with IF, (3) decentralized sensor network architecture with IF, and (4) hybrid sensor network architecture with IF and a track fusion algorithm. There are multiple functions and sub-functions contained within each block. Each major block is designed to be able to execute as a standalone (i.e., to generate its own sensor measurements and then carry out fusion) or in combination with each other (i.e., use previously generated sensor measurements by another block to carry out fusion). This modularity in design enables data portability between blocks and facilitates future research work. Each block and some of the significant functions are described below. The full MATLAB code is provided in Appendix B.

a. Centralized Sensor Network Architecture With Kalman Filter

This block sets up the sensor network as a centralized architecture. The measurements made by the sensors of the SAW are sent to a single fusion center for data processing. The fusion center implements the KF to fuse all the sensor data. Some of the more important functions are described in the following.

- **example_sim.m:** This script file sets up the mission environment (i.e., the location of Area A1 of Jurong Island, and the TSS and port limits of Singapore Strait), sensor topology (i.e., the disposition of the fixed and mobile sensors), the SAW trajectory (i.e., navigational passage in the Singapore Strait and attack approach), and generates the sensors measurements made of the SAW. Some of the important functions associated with this script are:
 - **ginit.m:** Sets the values used for the simulation parameters
 - **generate_platforms.m:** Creates the different types and number of platforms based on user requirements. The disposition and trajectory of the platforms are described in **platform_step_xxx.m**.
 - **generate_targets.m:** Creates the SAW and its initial location. The SAW trajectory is described in **target_step_SAW_path.m**.
 - **assign_sensors.m:** The user can specify the specifications of the sensors such as the platform they belong to, maximum range, beam width, measurement errors, etc..
 - **generate_observations.m** and **sensor_report.m:** Generates the sensor measurements made of the SAW. The true target location and true platform location are determined at each time step and the true range and bearing of the target from the sensor is computed. If this range and bearing are within the sensor range, the measurement is valid and noise is added.
- **run_filt.m:** This script file runs the KF for a multi-sensor, single-target scenario. It begins by setting up filter parameters then applies the KF on the sensor measurements made. Some of the important functions associated with this script are:
 - **init_track.m:** Extracts the first few observations and uses them to initialize the fused track
 - **xy_obs.m:** Converts the sensor measurement in range-bearing form to x-y form
 - **make_track.m:** Initializes the track-data structure and contains predicted and estimated states, respective covariances, and corresponding innovations
 - **state_pred.m:** Executes the state and covariance prediction using equations 3 and 4 shown in Chapter IV

- o **data_association.m**: Associates sensor reports to the target
- o **state_update.m**: Executes the state and covariance update using equations 5 and 6 shown in Chapter IV.

b. Centralized Sensor Network Architecture With Information Filter

The generation of sensor measurements for this block is the same as that elaborated for the centralized sensor network architecture with KF. The functions such as **example_sim.m**, **ginit.m**, and so on, are identical. However, instead of the KF, the IF is implemented here. The IF is responsible for fusing all the sensor data sent by the individual sensors to the fusion center. Some of the more important functions which are different from the above are now described.

- **run_ifilt.m**: This script file runs the IF for a multi-sensor, single-target scenario. This can be run on the same data generated for the KF so that the two methods may be compared. Like **run_filt.m**, this file begins by setting up filter parameters and then applies the IF on the sensor measurements made. Its corresponding functions are:
 - o **init_itrack.m**: Initializes the fused track.
 - o **make_itrack.m**: Initializes the track data structure and contains predicted and estimated states, and their respective covariances
 - o **info_pred.m**: Executes the state and covariance prediction using equations 7 and 8 shown in Chapter IV
 - o **info_update.m**: Executes the state and covariance update using equations 9 and 10 shown in Chapter IV.

c. Decentralized Sensor Network Architecture with Information Filter

This decentralized sensor network code simulates a SAW (i.e., single target) being tracked by multiple sensors. Each sensor acts as a fusion center capable of receiving measurement data from all the other sensors in the network and, together with its own measurement data, constructing a fused track. As with the centralized sensor network, the user is allowed to specify up to nine sensors—five fixed and four mobile—to make up the network of sensors. This block is compatible with the previous two blocks in such a way that sensor measurements generated by **example_sim.m** can be used here. Some of the important functions follow.

- **run_net.m:** This script file is similar to **example_sim.m**. It sets up the mission environment, sensor topology, SAW trajectory, and generate the sensors measurements made of the SAW. As aforementioned, the user has the option to generate new sensor measurements or reuse previously generated sensor measurements.
 - **local_predict.m:** Executes the state and covariance prediction using the equations 7 and 8 shown in Chapter IV.
 - **local_update.m:** Executes the state and covariance update using the equations 9 and 10 shown in Chapter IV
 - **set_net_topology.m:** Constructs the communications links among sensors. In this research, each sensor is connected to all other sensors in the network.
 - **communicate.m:** Communicates information between all the sensors.
 - **assimilate.m:** Adds new information gained from sensors from other sensor platforms while discarding common information.

d. *Hybrid Sensor Network Architecture With Information Filter and Track-to-Track Fusion Algorithm*

The last of the four major code blocks, this code block implements both data and track fusion. Its setup follows that of the previous three code blocks. The option of generating new sensor measurements or using previously generated sensor measurements is also available in this block. The functions to carry out data fusion using the IF are the same as those used for the decentralized sensor network, namely, **local_predict.m** and **local_update.m**. The locally fused track produced by each sensor is then integrated into a global fused track using the track-to-track fusion algorithm (equations 11 and 12 shown in Chapter IV) residing in the function **run_indep.m**.

3. Evaluation Parameters

The MOEs are derived in Chapter II. The distributed sensor network is evaluated based on its ability to provide accurate identification and early warning of a SAW intending to crash into Jurong Island. These MOEs are related to the probability of impact by the SAW on Jurong estimated as the SAW is being tracked. The estimation of

the impact probability mirrors the computation of the probability of destruction of a point target [47] and necessitates the computation of the predicted mean impact point on Jurong as a function of time.

a. Calculate the Mean Impact Point

$$\hat{\mathbf{x}}(\text{tgo}|k) = \begin{pmatrix} x_{\text{tgo}|k} \\ \dot{x}_{\text{tgo}|k} \\ y_{\text{tgo}|k} \\ \dot{y}_{\text{tgo}|k} \end{pmatrix}$$

At time, t_k , the predicted state is, $P(\text{tgo}|k)$, for $k=1, \dots, N$, where tgo stands for time-to-go, that is, the difference between the predicted time the SAW would strike Jurong and the current time at which its state is estimated. The predicted uncertainty in the x-component, $x_{\text{tgo}|k}$, is $\sigma_{x|k}^2 = [P(\text{tgo}|k)]_{11}$, and the predicted uncertainty in the y-component, $y_{\text{tgo}|k}$, is $\sigma_{y|k}^2 = [P(\text{tgo}|k)]_{33}$. Hence, the x-component, $\bar{x}_{\text{tgo}}$, and y-component, $\bar{y}_{\text{tgo}}$, of the predicted mean impact point, and their corresponding uncertainties, $\sigma_{\bar{x}_{\text{tgo}}}^2$ and $\sigma_{\bar{y}_{\text{tgo}}}^2$ can be calculated formulas according to

$$\sigma_{\bar{x}_{\text{tgo}}}^2 = (W^T C_x^{-1} W)^{-1}, \quad \sigma_{\bar{y}_{\text{tgo}}}^2 = (W^T C_y^{-1} W)^{-1},$$

$$\bar{x}_{\text{tgo}} = \sigma_{\bar{x}_{\text{tgo}}}^2 (W^T C_x^{-1} X), \quad \bar{y}_{\text{tgo}} = \sigma_{\bar{y}_{\text{tgo}}}^2 (W^T C_y^{-1} Y),$$

where

$W = (1 \ 1 \ 1 \ \dots \ 1)$, $\dim W = N$, where N is the number of simulation runs conducted,

$$C_x = \begin{pmatrix} \sigma_{x|1}^2 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sigma_{x|N}^2 \end{pmatrix}, \dim C_x = N \times N,$$

$$C_y = \begin{pmatrix} \sigma_{y|1}^2 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sigma_{y|N}^2 \end{pmatrix}, \dim C_y = N \times N,$$

$$X = \begin{pmatrix} X_{tgo|1} \\ \vdots \\ X_{tgo|N} \end{pmatrix}, \text{ and } Y = \begin{pmatrix} Y_{tgo|1} \\ \vdots \\ Y_{tgo|N} \end{pmatrix}.$$

b. Calculate the Probability of Impact

Referencing [47], the probability density function, $f(x,y)$, for the projected impact point from a point target (on Jurong Island) is given by

$$f(x,y) = \frac{1}{2\pi\sigma_{x_{tgo}}\sigma_{y_{tgo}}} \exp \left[-\frac{(x-\bar{x}_{tgo})^2}{2\sigma_{x_{tgo}}^2} - \frac{(y-\bar{y}_{tgo})^2}{2\sigma_{y_{tgo}}^2} \right],$$

where $(\bar{x}_{tgo}, \bar{y}_{tgo})$ are the mean coordinates and $\sigma_{x_{tgo}}^2, \sigma_{y_{tgo}}^2$ are the variances of $f(x,y)$.

The probability of impact within a circle of radius R , measured from Jurong Island, can be calculated from

$$P = \iint_{r < R} f(x,y) \, dx dy,$$

where $r = \sqrt{x^2 + y^2}$. Thus,

$$P = \frac{1}{2\pi\sigma_{x_{tgo}}\sigma_{y_{tgo}}} \iint_{r < R} \exp \left[-\frac{(x-\bar{x}_{tgo})^2}{2\sigma_{x_{tgo}}^2} - \frac{(y-\bar{y}_{tgo})^2}{2\sigma_{y_{tgo}}^2} \right] dx dy,$$

which is obtained numerically.

In this simulative study, it is observed that $\sigma_{x_{tgo}} \approx \sigma_{y_{tgo}}$. Hence, making the approximation of $\sigma_{x_{tgo}} \approx \sigma_{y_{tgo}} = \sigma$, the probability of impact, P , can be simplified to

$$P = \frac{1}{2\pi\sigma^2} \iint_{r < R} \exp \left[-\frac{(x-\bar{x}_{tgo})^2}{2\sigma^2} - \frac{(y-\bar{y}_{tgo})^2}{2\sigma^2} \right] dx dy.$$

To further simplify subsequent calculations, the coordinate axes are rotated so that $\mathbf{r}_o = \sqrt{\bar{x}_{tgo}^2 + \bar{y}_{tgo}^2}$ lies along the positive x-axis. Then,

$$P = \frac{1}{2\pi\sigma^2} \iint_{r < R} \exp \left[-\frac{(x-r_o)^2}{2\sigma^2} - \frac{y^2}{2\sigma^2} \right] dx dy,$$

which, following a subsequent transformation to polar coordinates, becomes

$$P \left(\frac{R}{\sigma}, \frac{r_o}{\sigma} \right) = \frac{1}{2\pi\sigma^2} \exp \left(-\frac{r_o^2}{2\sigma^2} \right) \int_0^R \int_0^{2\pi} r \exp \left[-\frac{r^2}{2\sigma^2} + \frac{rr_o \cos \theta}{\sigma^2} \right] d\theta dr.$$

Finally, $P \left(\frac{R}{\sigma}, \frac{r_o}{\sigma} \right)$ can be written as

$$P \left(\frac{R}{\sigma}, \frac{r_o}{\sigma} \right) = \sum_{n=0}^{\infty} g_n h_n,$$

where g_n and h_n are calculated recursively according to

$$g_n = \frac{1}{n} \left(\frac{r_o^2}{2\sigma^2} \right) g_{n-1}, \text{ with } g_0 = \exp \left(-\frac{r_o^2}{2\sigma^2} \right), \text{ and}$$

$$h_n = -\frac{1}{n!} \left(\frac{R^2}{2\sigma^2} \right)^n \exp \left(-\frac{R^2}{2\sigma^2} \right) + h_{n-1}, \text{ with } h_0 = \left(\frac{R^2}{2\sigma^2} \right) \int_0^1 e^{-\frac{R^2}{2\sigma^2} u} du = 1 - e^{-\frac{R^2}{2\sigma^2}}.$$

4. Test Cases

Table 5 summarizes the various test sets used in this simulative study. A total of 50 simulation runs are made for each case, in which the same set of inputs (i.e., sensor observations) is given. Hence, any difference in the results emanates solely from the choice of sensor-network architecture and sensor-fusion algorithm. Note that the numbers of sensors used is the number of sensors used in the sensor network, as indicated Table 5. The actual number of sensors whose tracking range covers the sector (i.e., Sector 73) where Jurong Island is located is smaller. For example, if the test case

indicates five fixed sensors used, only three fixed sensors have coverage over Sector 73 because this sector is out of sensor range for two of the fixed sensors located at the east entrance of the Singapore Strait. Similarly, if the test case indicates nine sensors (comprising five fixed and four mobile sensors) in actuality, only seven sensors (i.e., three fixed and four mobile sensors) are covering Sector 73.

Table 5. Various Cases Used in This Simulative Study

Case	No. of Sensors used*	Sensor Network Architecture	Algorithm	Compared to	Objective
1	5 FS	C	KF		
2	5 FS	C	IF	Case 1	Validate that KF and IF produces the same results
3	5 FS	D	IF	Case 2	Validate that C and D produces the same results for this given context. Thereafter, it is immaterial whether C or D is used.
4	9 (5FS+4MS)	C or D	IF	Case 2	More sensors equates to better results
5	5 FS	H	IF + Track-to-track fusion	Case 2	Track fusion produces poorer results
6	9 (5FS+4MS)	Hybrid	IF + Track-to-track fusion	Case 4	Track fusion produces poorer results and the gap narrows with more sensors
				Case 5	More sensors equates to better results
7	5 FS	C or D (subjected to 10%, 30%, and 50% data loss)	IF		Data loss results in poorer results
				Case 5	Data loss narrows the performance gap between centralized/decentralized and hybrid architectures.
8	8 (5FS+3MS)	Hybrid	IF + Track-to-track fusion		The number of sensors used is meant to reflect the maximum number of sensors if sensor failures are accounted for (as

					covered under sensor coverage analysis in Chapter 2).
9	8 (5FS+3MS)	C or D (subjected to 10%, 30%, and 50% data loss)	IF	Case 8	Data loss narrows the performance gap between centralized/decentralized and hybrid architectures.
10	-	-	-	Cases 8 and 9 vs Cases 5 and 7	Accounting for sensor failures, cases 8 and 9 reflects the maximum number of sensors providing sensor coverage over the sector in which Jurong Island sits, while cases 5 and 7 reflects the minimum number of sensors. The comparison between them will provide insights on the performance gap.

(*FS – Fixed Sensors, MS – Mobile Sensors, C – Centralized, D – Decentralized, H – Hybrid)

C. DISCUSSION OF RESULTS

1. Comparison Between Kalman and Information Filters

Comparing cases 1 and 2, the simulation results for these test cases validate that the KF and IF produce identical results. As indicated by Figure 21, the KF and IF produce the same probability of impact and estimated time error. This result is expected because both filters are algebraically the same, with the IF being in the logarithmic form of the KF. However, IF has some computational advantages over KF, given its simplicity in making observation updates to predictions [48]. Taking the computational advantages into consideration, the IF is used for subsequent test cases for all the three sensor network architectures. This research, however, does not seek to quantify and compare the computational efficacy of the two filters.

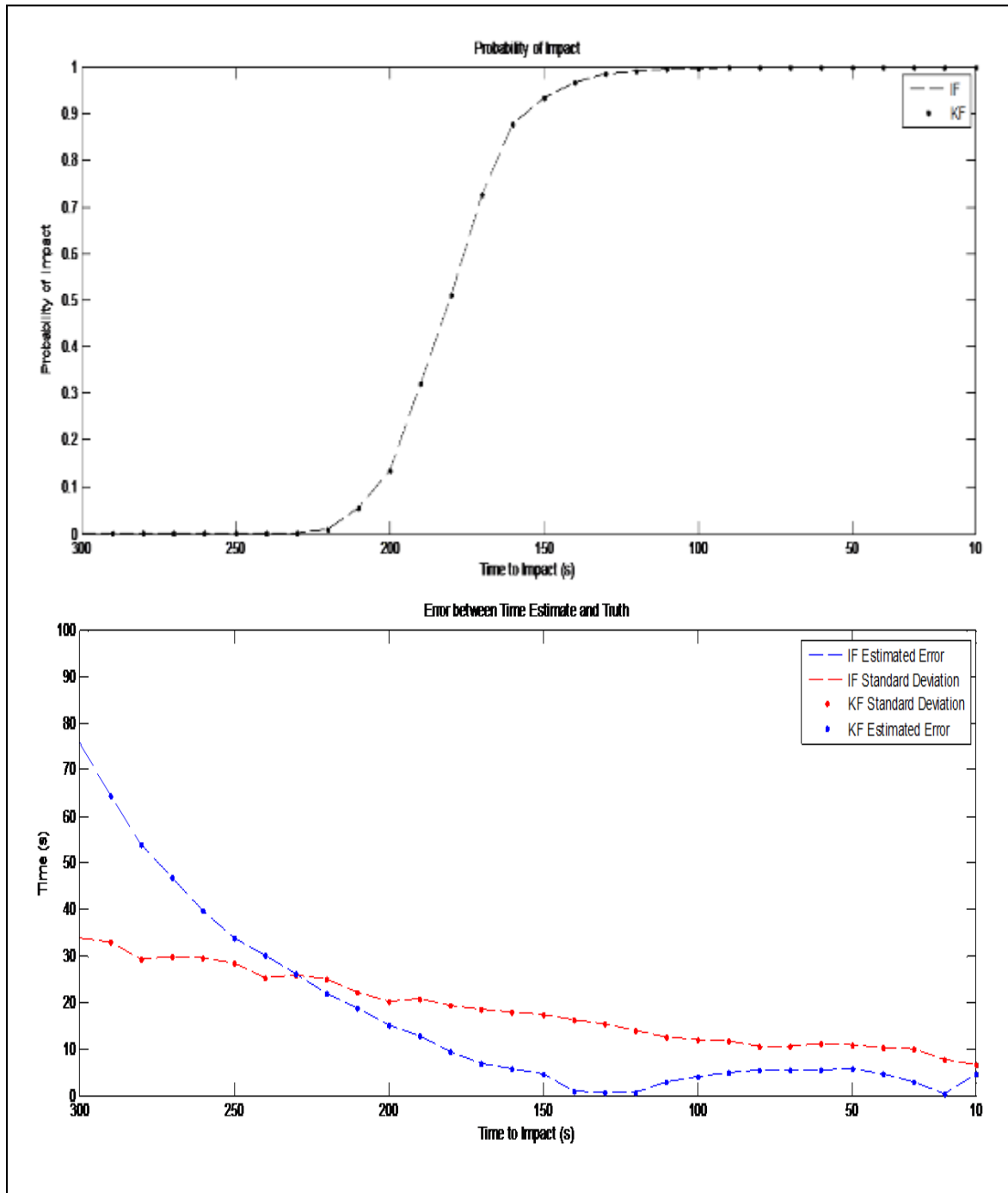


Figure 21. Comparison of KF and IF for Centralized Five-sensor Network Architecture

2. Comparison Between Centralized and Decentralized DSN Architectures

As shown by Figure 22, the centralized and decentralized DSN architectures, through test cases 2 and 3, produce the same probability of impact and estimated time error. These results, too, come as no surprise, because both architectures use the same data-fusion algorithm (i.e., IF) to process the same sensor observations. In the context of this research and as described in Chapter III, the main difference between these two architectures is that the decentralized sensor network has more fusion centers as redundancies. For the subsequent test cases, only the centralized DSN architecture are used for the simulation runs, since the centralized and decentralized DSN architectures produce the same results. It should be noted that the results of the centralized DSN architecture are representative of both architectures.

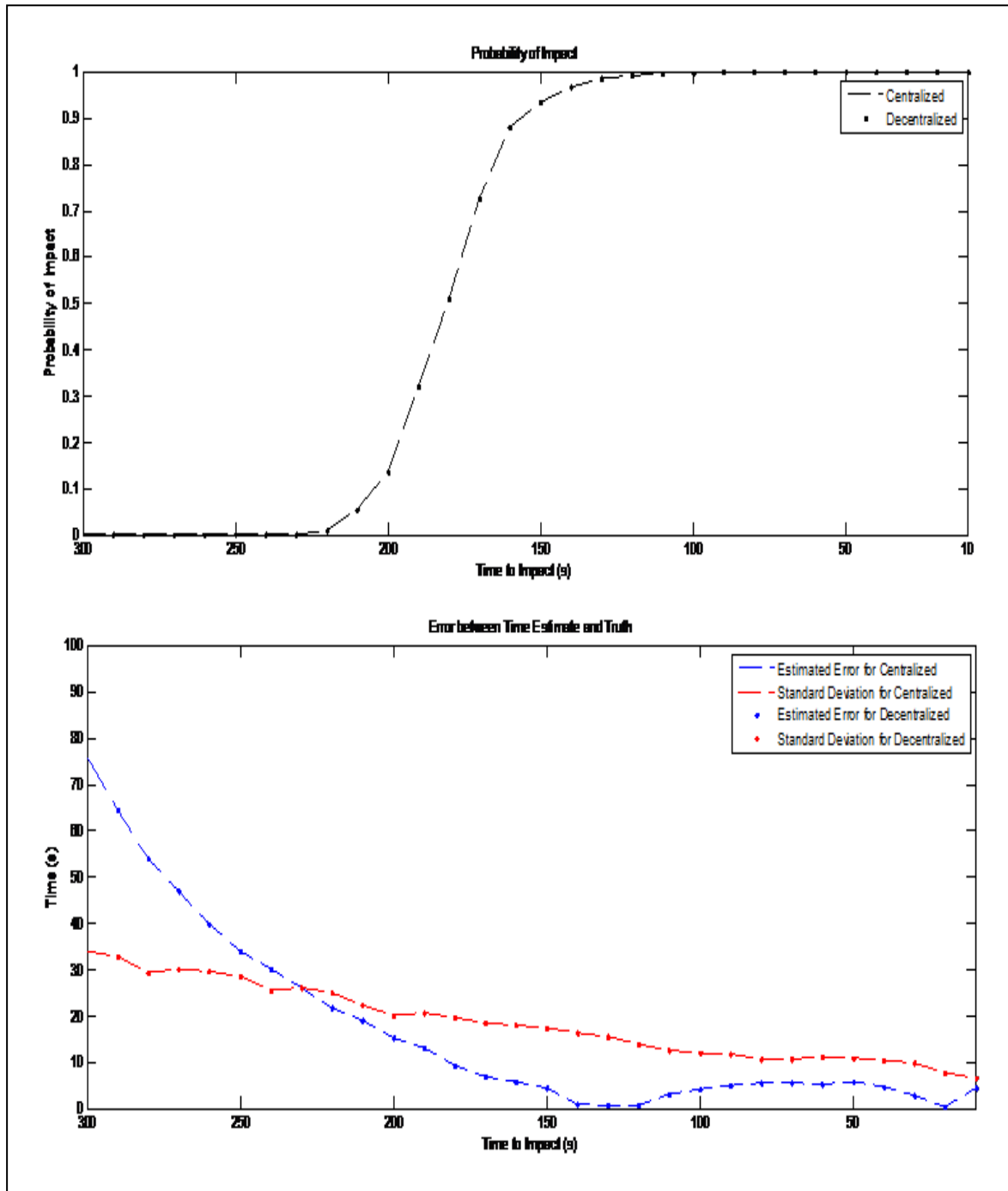


Figure 22. Comparison of Five-sensor Centralized and Decentralized DSN Architectures

3. Comparison Between Different Number of Sensors for Centralized DSN

As indicated by Figure 23, the centralized DSN architecture with nine sensors provide earlier warning of an impending attack with a lower magnitude of error in time than does the centralized DSN architecture with five sensors. Though the results may seem intuitive (i.e., more sensors equal better performance), they are true only because the additional mobile sensors (i.e., the four mobile sensors in addition to the five fixed sensors) are assumed to have specifications similar to those of the fixed sensors. If their specifications are significantly poorer, more sensors would produce a resultant fused track with bigger errors. That is, in this case, a network with more sensors would not necessarily perform better than a network with fewer sensors.

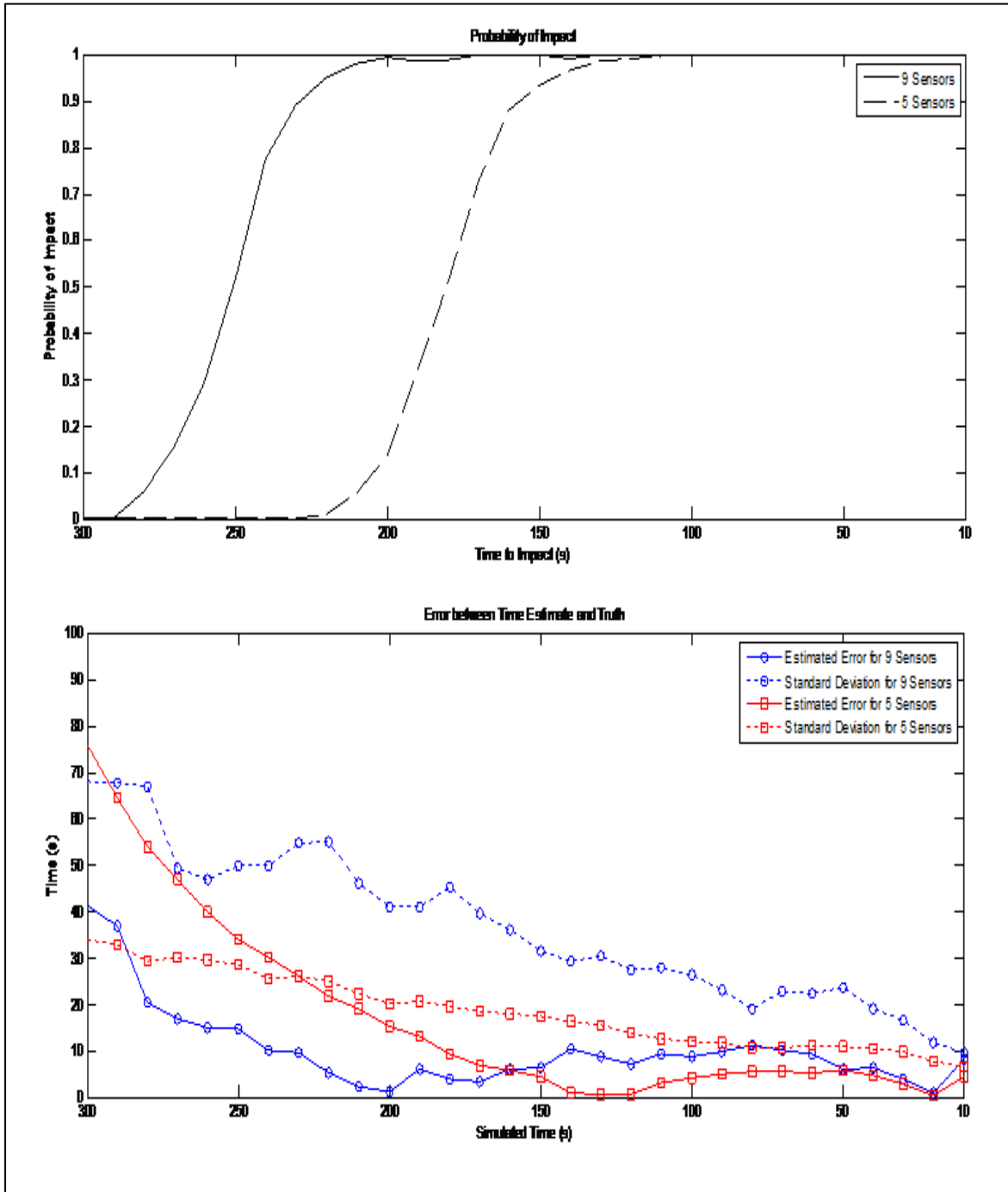


Figure 23. Comparison of Number of Sensors (Five and Nine) for a Centralized DSN

4. Comparison Between Centralized and Hybrid DSN Architectures Using Five Sensors

The results produced using the centralized and hybrid sensor network architectures for five fixed sensors are compared here (i.e., Case 5 vs. Case 2). Figure 24 indicates that the centralized DSN architecture outperforms the hybrid architecture. For the centralized DSN architecture, the probability of impact increases from 0% (at 240 seconds to impact) to 100% (at 80 seconds to impact), compared to 200 seconds and 60 seconds for the hybrid architecture. However, this comparison is made under the condition of unlimited communications bandwidth. In reality, bandwidth is limited, and disseminating loads of raw sensor data within the centralized and decentralized DSN, instead of fused-track estimates for the hybrid sensor-network architectures, would usually result in data loss or latency. Data loss will be discussed later in this chapter.

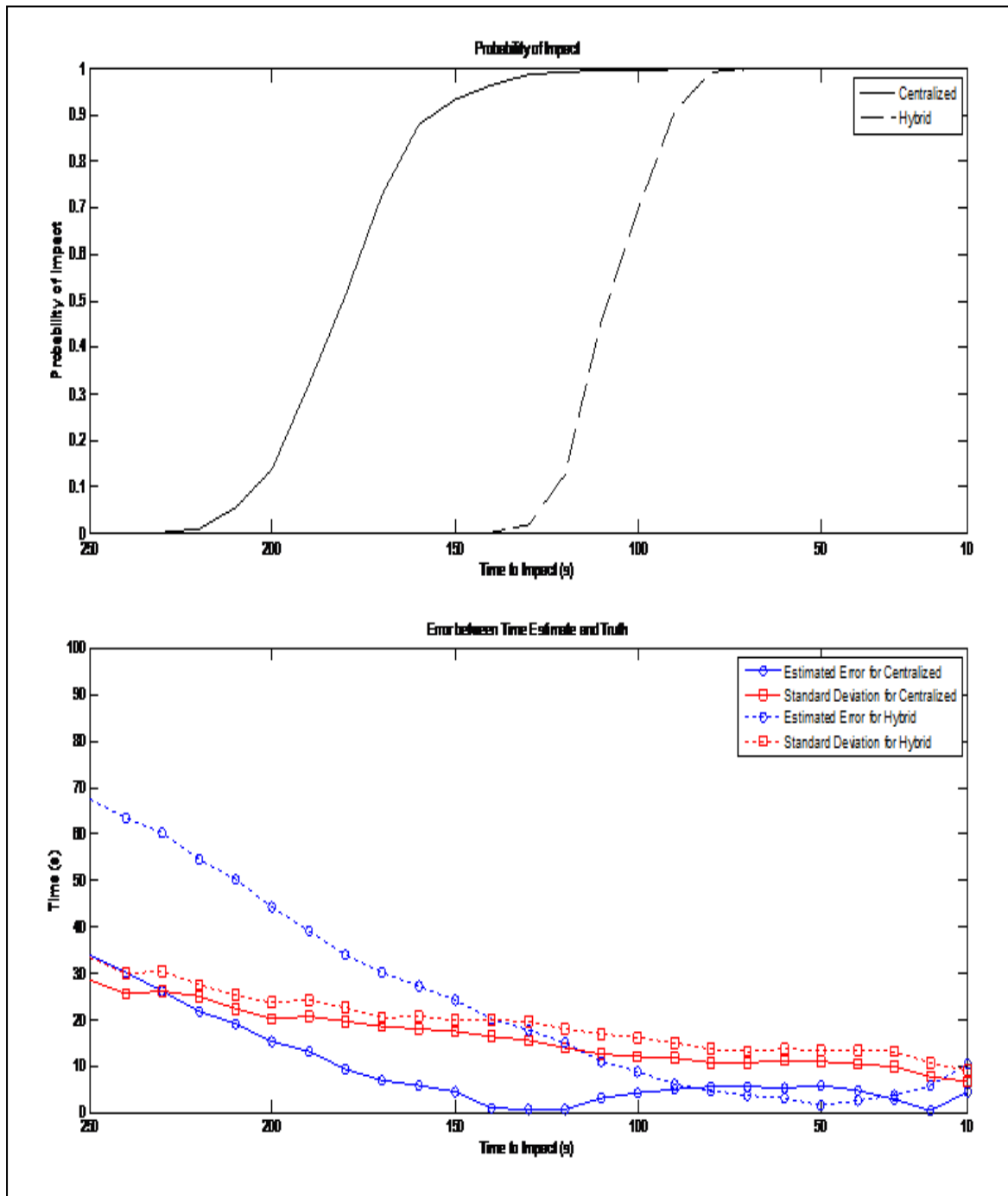


Figure 24. Comparison between Five-sensor Centralized and Hybrid DSN Architectures

5. Comparison Between Centralized and Hybrid DSN Architectures Using Nine Sensors

Using more sensors of comparable specifications (nine in this case) results in a widening of the performance gap between the centralized and hybrid sensor-network architectures. This widening is attributed to the better increase in performance for the centralized architecture with more sensors as compared to the moderate increase in performance for the hybrid architecture. Based on the error and standard deviation of the time-to-impact estimate, the hybrid architecture using the track-to-track fusion algorithm produces time estimates with bigger errors initially and a smaller standard deviation as compared to the centralized architecture. This smaller standard deviation, σ , is the result of the fused track having a smaller variance (computed using Equation 12), which results in a greater rate of change of the probability of impact.

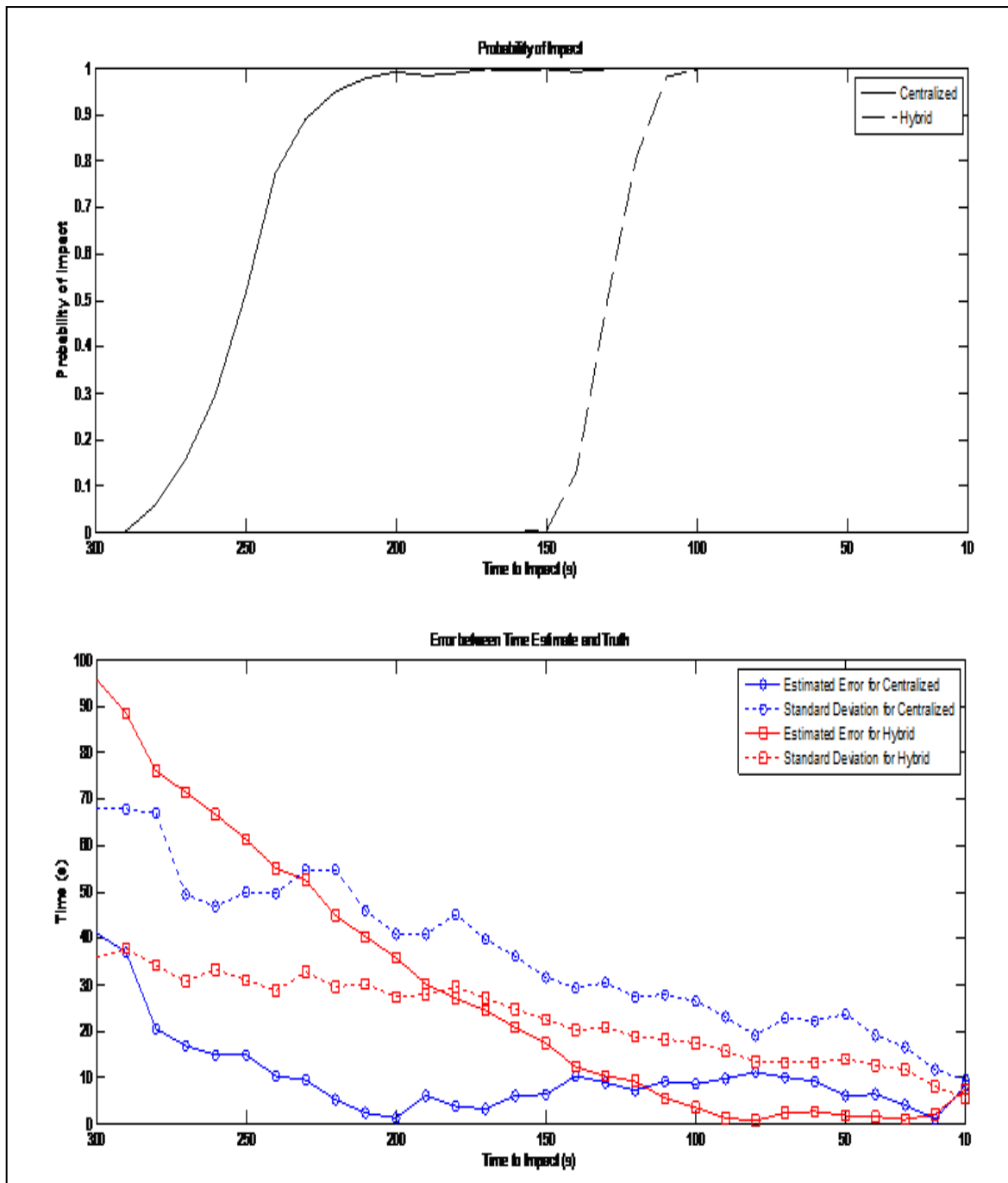


Figure 25. Comparison between Nine-sensor Centralized and Hybrid DSN Architectures

6. Comparison Between Different Number of Sensors for Hybrid DSN

As shown in Figure 25, the improvement in employing more sensors for the hybrid architecture is not significant compared to that of the centralized architecture. Comparing Figures 23 and 26 shows the stark contrast in improvement for the different architectures when more sensors are used. These results show that increasing the number of sensors produces varying degrees of performance improvement, depending on the chosen DSN architecture in use.

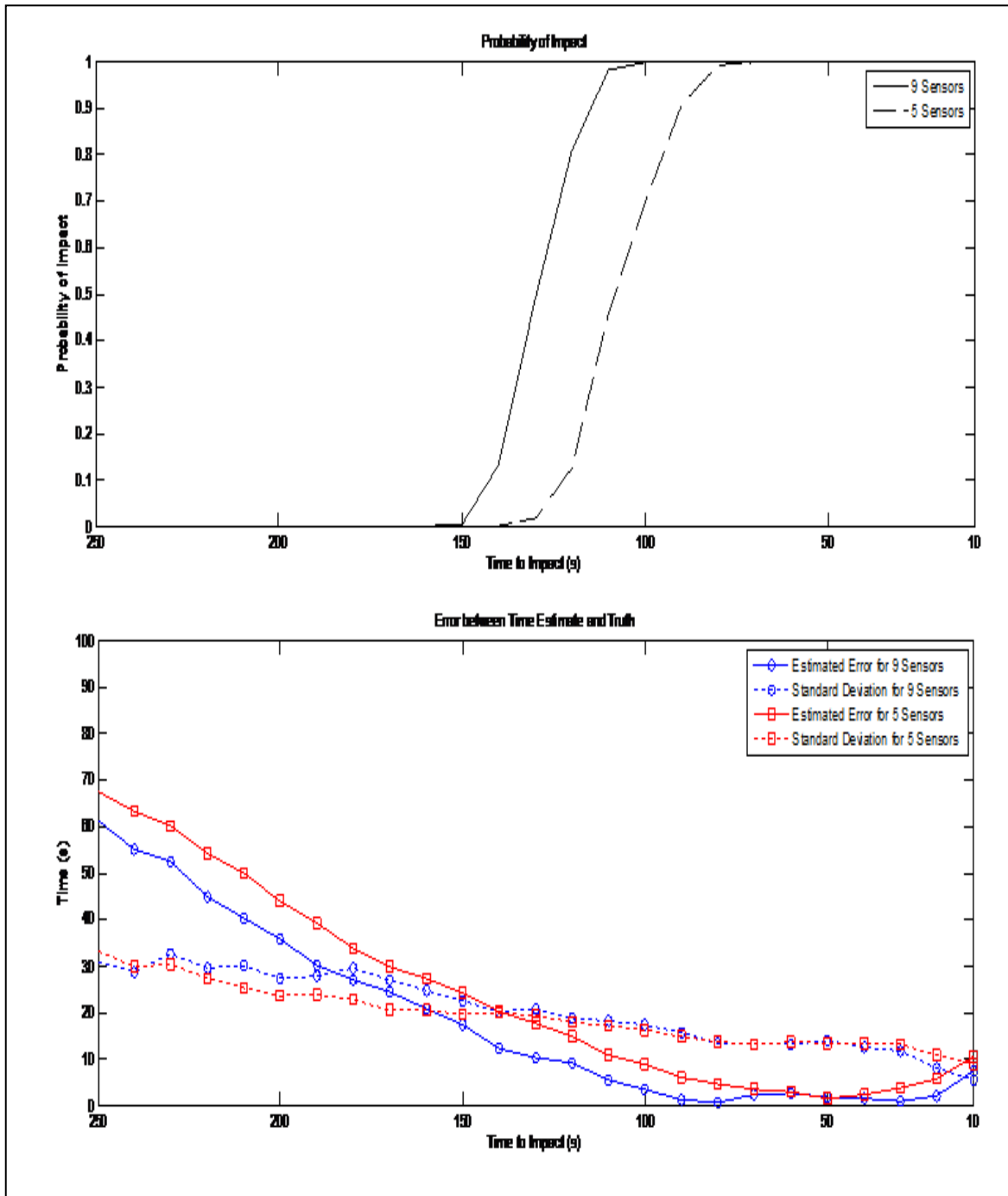


Figure 26. Comparison of Number of Sensors (Five and Nine) for a Hybrid DSN

7. Effects of Data Loss for Centralized and Decentralized DSN Architectures

a. Using Five Sensors

The dissemination of raw sensor data takes up much more communications bandwidth than that of track estimates. Communication bandwidth limitations would usually result in data loss or data latency. In this study, varying degrees of data loss are incorporated to simulate the conditions of distributing a high volume of data under limited bandwidth for the centralized and decentralized sensor-network architectures.

Figure 27 shows the performance results for the centralized/decentralized sensor-network architectures with 0%, 10%, 30%, and 50% of data loss, and for the hybrid sensor network architecture using five sensors. Data loss narrows the performance gap between the centralized/decentralized and hybrid architectures. Also noteworthy is that while the hybrid architectures do not provide as early a warning, its high rate of change of probability of impact with respect to time allows it to achieve an estimation of 100% probability of impact at almost the same time as the rest of the variants in the centralized/decentralized architecture.

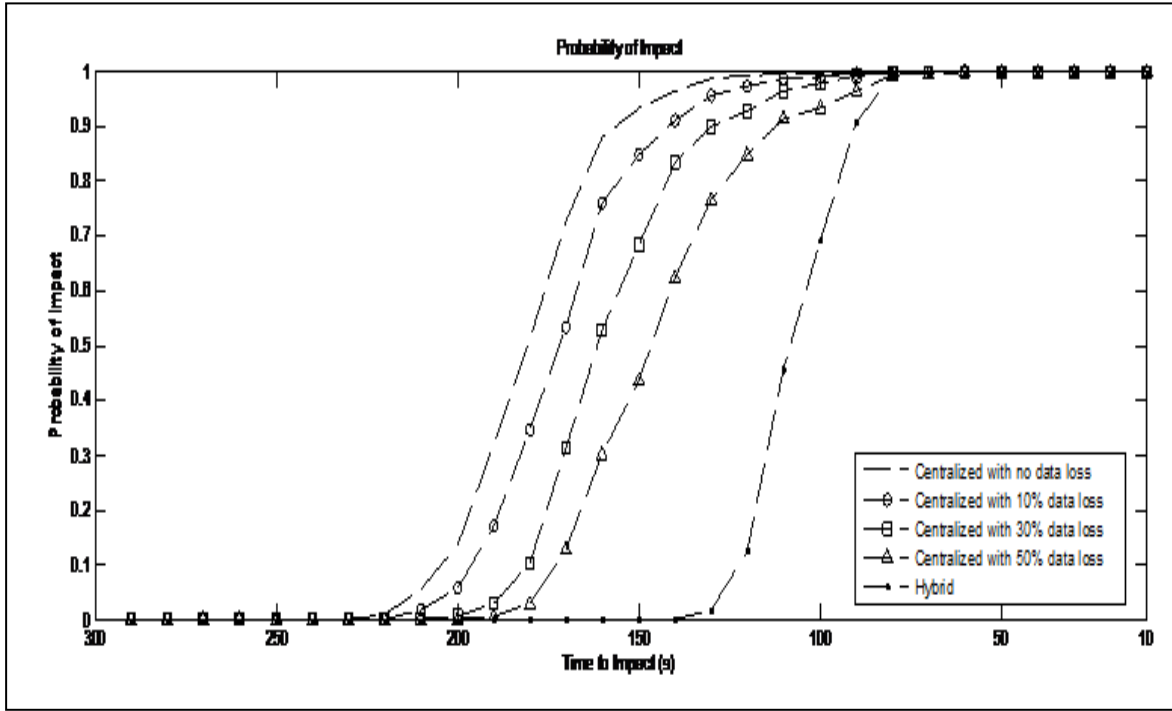


Figure 27. Comparison of the Centralized DSN with No Data Loss, 10%, 30%, and 50% Data Loss, and the Hybrid DSN of Five Sensors

b. Using Eight Sensors

Figure 11 in Chapter II show that the maximum and minimum numbers of sensors monitoring Jurong Island are six and three, respectively. As explained in Table 5, the number of sensors monitoring Jurong Island is two fewer than the total sensors used. Figure 27 shows the sensor-network performance for the lower number of sensors limit. In Figure 28, the upper number-of-sensors limit, determined from the sensor coverage analysis, is simulated. The results indicate that with more sensors employed, the wide performance gap between the centralized and hybrid sensor network architectures remains even with the injection of data loss.

The hybrid architecture in Figure 28, which is similar to Figure 27, also generates a high rate of change of probability of impact with respect to time to impact, allowing it to achieve 100% at almost the same time or even earlier (for the case of 50% data loss) than the centralized architecture.

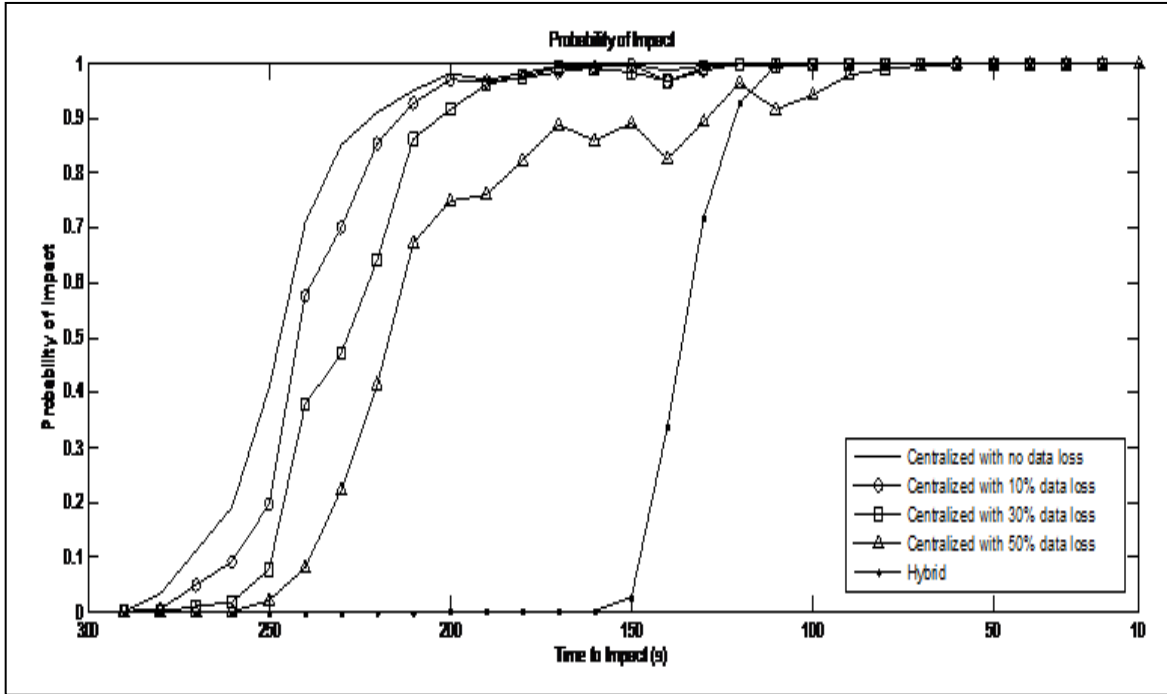


Figure 28. Comparison of the Centralized DSN with No Data Loss, 10%, 30%, and 50% Data Loss, and the Hybrid DSN of Eight Sensors

8. Comparison Between Different DSN Architectures and Different Number of Sensors

The simulation results of Figures 27 and 28 are presented together in Figure 29. Relating it to the sensor coverage analysis done in Chapter II, the best and worst case performance for any of the sensor network architectures for the given suite of sensors can be gleaned.

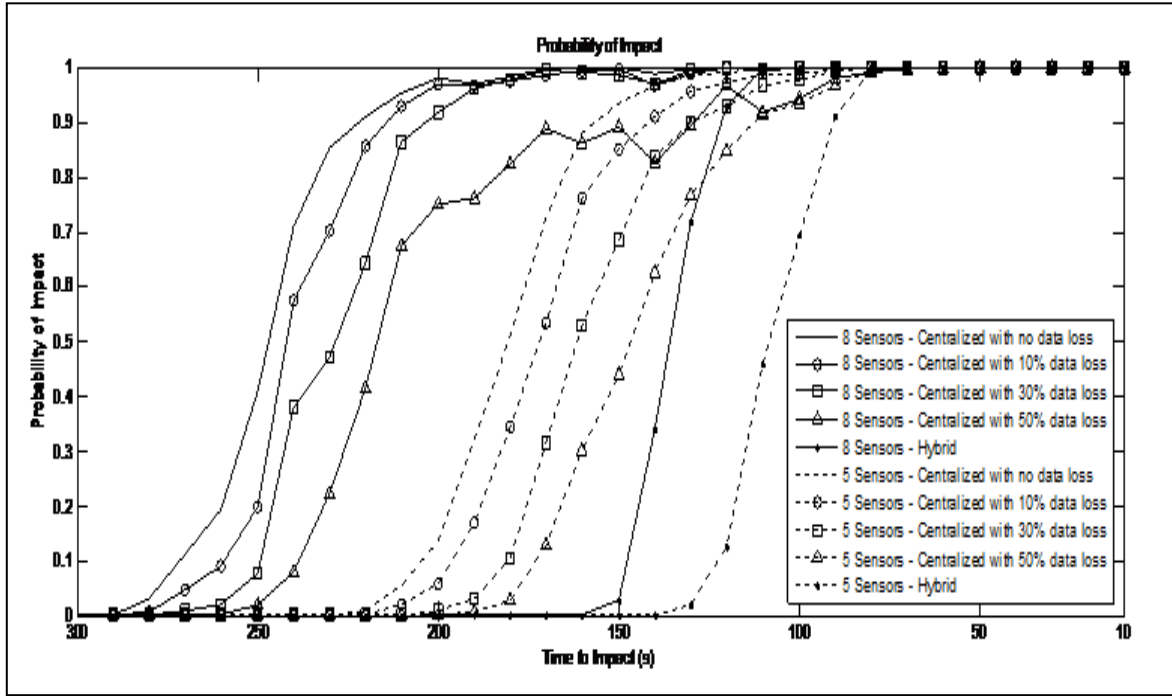


Figure 29. Comparison of the Centralized DSN with No Data Loss, 10%, 30%, and 50% Data Loss, and the Hybrid DSN of Five and Eight Sensors

Table 6 extracts some salient data points from Figure 29 to illustrate the response time from the point when the probability of impact breaches 0%, 50%, 80%, or 90%, and reaches 100%. For example, taking the case of 90% probability of impact, for the centralized DSN with no data loss, using eight sensors, the early warning (i.e., time to impact) is raised to 220 seconds, as opposed to 160 seconds if only five sensors are used.

Relating these results to the MOEs, in the worst-case scenario, a five-sensor hybrid DSN architecture can afford a early impact warning of only 150 seconds, while in the best-case scenario, a no-data-loss, eight-sensor, centralized or decentralized DSN architecture can afford up to 290 seconds. If the identification of intent “confidence threshold” is set at 80%, the worst-case result is 95 seconds (using a five-sensor hybrid DSN architecture), as compared with the best case of 235 seconds using a no-data-loss, eight-sensor, centralized or decentralized DSN architecture.

Table 6. A Selection of Salient Data Points

	Time to Impact (s)									
	5 Sensors					8 Sensors				
	IF	IF with 10% data loss	IF with 30% data loss	IF with 50% data loss	HY	IF	IF with 10% data loss	IF with 30% data loss	IF with 50% data loss	HY
1	80	70	70	60	60	120	90	90	60	90
0.9	160	140	130	110	90	220	215	200	130	120
0.8	165	155	145	125	95	235	225	215	180	125
0.5	180	170	160	145	105	255	258	230	215	135
0	240	230	220	200	150	290	290	280	270	170

D. PERFORMANCE EVALUATION

The simulation results reveal that a centralized or decentralized architecture generally outperforms the hybrid architecture. This is attributed to the hybrid architecture's larger state estimation errors which is inherent to track fusion as compared to data fusion. In other words, the fused tracks are not as accurate as they would be, if they were formed directly from the sensor measurements. This finding is consistent with those reported by Chen *et al.* [49] and Rogers [50].

In addition to the inherent disadvantage of using the track-to-track fusion algorithm, the centralized and decentralized architectures are operating under "ideal" conditions. By injecting a simple form of data loss, it is shown that the performance gap between them and the hybrid architecture can be narrowed. Hence, should additional constraints such as data latency, asynchronous data arrival, and multiple-target tracking be taken into consideration, these would add additional layers of complexity and significantly reduce the performance advantage that the centralized and decentralized architectures have now.

Next, to determine if a centralized or decentralized sensor network architecture performs better, they can be differentiated through the degree of data loss faced by each network. As there are more communications links at the bottom tier for the decentralized

sensor network architecture, it can be presumed to suffer from higher data loss compared to the centralized sensor network architecture. As an example, if the decentralized DSN suffers from 30% data loss while the centralized DSN suffers from 10% data loss, then from the results presented in Figures 27 through 29, the performance of a centralized architecture is distinctly better.

Overall, comparing all three DSN architectures, the centralized DSN architecture produces the best performance in terms of being accurate and early in identifying a SAW with the intent of striking Jurong Island. This performance evaluation is only valid under the assumptions herein and that the centralized architecture is vulnerable to single-point of failure (at the fusion center) is not a factor.

E. SUMMARY

This chapter covers the details and results of the simulative study. Guided by the SoSADP, the modeling considerations and parameters are drawn from the analysis made in Chapters II and III, which ensure that the M&S program is consistent with mission needs and operational settings. The study results indicate that the number of sensors and the type of sensor network architecture are key in providing an accurate identification of SAW intent and early warning. Under the assumptions made, neither the KF nor IF data fusion algorithm has an advantage. In addition, data fusion is proven to produce better performance than track fusion (i.e., track-to-track fusion). Finally, in the presence of data loss, the centralized architecture performs best, followed by the decentralized, and then the hybrid.

VI. CONCLUSION AND FUTURE RESEARCH

A. INTRODUCTION

This chapter summarizes the purpose and conduct of this thesis research, recaps the key findings, and proposes potential areas for future research.

B. RESEARCH SUMMARY

A successful terrorist attack using a ship as a weapon on shore infrastructure in the Malacca and Singapore straits would bring chaos to global trade since these straits carry over a quarter of the world's commerce and half the world's oil. This calamity must be prevented. Towards this goal, this thesis aims at developing and determining the best distributed sensor network (DSN) architecture and to implement a sensor fusion algorithm to effect an accurate identification of a SAW and warn of a potential attack on oil and chemical terminals at the earliest possible time.

The work in this thesis involves the application of the System-of-Systems Architecture Development Process (SoSADP) for designing alternative DSN architectures, Kalman and information filters for SAW tracking and sensor data fusion, a track-to-track fusion algorithm, and a Monte Carlo simulative study to assess the effectiveness of three distributed sensor-fusion network architectures: centralized, decentralized, and hybrid. All DSN architectures include the various sensors that Singapore deploys in and along the Strait.

The results and concepts in this thesis provide invaluable insights in the development of response capabilities against a SAW in the Singapore Strait. These results and concepts can also be extended to other key installations within Singapore and other countries.

C. KEY FINDINGS

The simulative study results (Figure 21) indicate that the Kalman filter and the information filter are equally effective in SAW tracking. Also, given the assumption that the fusion centers employ the same data-fusion algorithm, Figure 22 validates that the

centralized and decentralized sensor fusion network architectures produce the same performance in terms of early and accurate identification of a SAW. Figures 23 and 26, show proof that employing more sensors would deliver better results, albeit at varying degrees, for all three DSN architectures.

Next, Figure 29, show with or without communications bandwidth limitation (implemented in the form of data loss), a ship with the intent to attack Jurong can be identified accurately at an earlier time with the centralized and decentralized sensor-fusion network architectures than with the hybrid architecture.

Finally, to further differentiate the performance of the centralized and decentralized sensor networks, the decentralized sensor network is assumed to have a higher level of data loss due to more communications link in its network compared to the centralized sensor network. With all the assumptions made, the centralized DSN architecture is determined to perform the best, followed by the decentralized DSN architecture, and then the hybrid DSN architecture.

D. AREAS FOR FUTURE RESEARCH

The problem scenario, assumptions, and results of this research reflect only an instance of a larger and more complex MDS environment and MDA problem. All of the sensors, sensor-platform models, and threat characteristics created for this research can be scaled and adapted to suit other mission scenarios. The simulation program developed for this research hopes to serve as a launch pad for the implementation of multi-target scenarios, other data fusion algorithms, such as the particle filter, a wider variety of sensors and sensor platforms, dynamic sensor management, and more rigorous communications bandwidth constraints (i.e. data latency and asynchronous data). The incorporation of budget constraints to the simulation will also add realism to the deployment of more sensors and sensor platforms.

Accruing from the results produced in this study, one pressing area of future research is the development of a defensive or offensive response to a SAW on a crash

course towards Jurong Island. Another potential area is the integration of intelligence reports and open-source databases to identify a SAW far from the shores of Singapore and achieve a longer response time.

E. CONCLUSION

This thesis apply an integrated systems-engineering methodology to develop and determine the best distributed sensor-network architecture for a given sensor-fusion algorithm. Given the context and assumptions made, the key research findings provide valuable insights to the Singapore maritime security agencies on the benefits of employing more sensors, the advantages of using the centralized DSN architecture, and the limited time to effect a response to against a SAW.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. SENSOR COVERAGE ANALYSIS

A. MPA SHORE STATION SENSOR COVERAGE

The table below presents the estimated number of MPA sensor platforms covering each sector of the Singapore Strait for each hour of the day. These platforms are stationary. A total of five platforms are used in this research.

	FIXED MPA SENSOR PLATFORMS									
SECTOR	71	72	73	81	82	83	91	92	93	94
1		2	3	4	5	4	3	2	1	0
2	1	2	3	4	5	4	3	2	1	0
	1	2	3	4	5	4	3	2	1	0
4	1	2	3	4	5	4	3	2	1	0
5	1	2	3	4	5	4	3	2	1	0
6	1	2	3	4	5	4	3	2	1	0
7	1	2	3	4	5	4	3	2	1	0
8	1	2	3	4	5	4	3	2	1	0
9	1	2	3	4	5	4	3	2	1	0
10	1	2	3	4	5	4	3	2	1	0
11	1	2	3	4	5	4	3	2	1	0
12	1	2	3	4	5	4	3	2	1	0
13	1	2	3	4	5	4	3	2	1	0
14	1	2	3	4	5	4	3	2	1	0
15	1	2	3	4	5	4	3	2	1	0
16	1	2	3	4	5	4	3	2	1	0
17	1	2	3	4	5	4	3	2	1	0
18	1	2	3	4	5	4	3	2	1	0
19	1	2	3	4	5	4	3	2	1	0
20	1	2	3	4	5	4	3	2	1	0
21	1	2	3	4	5	4	3	2	1	0
22	1	2	3	4	5	4	3	2	1	0
23	1	2	3	4	5	4	3	2	1	0
24	1	2	3	4	5	4	3	2	1	0

* TOD: Time of the Day

B. COASTAL PATROL CRAFT SENSOR COVERAGE

The table below presents the estimated number of CPC sensor platform covering each sector of the Singapore Strait for each hour of the day. Each platform is mobile and takes on a different patrol route. Two platforms are used in this research.

TOD*	MOBILE CPC SENSOR PLATFORM									
SECTOR	71	72	73	81	82	83	91	92	93	94
1	1				1					
2		1				1				
3			1				1			
4				1				1		
5					1				1	
6				1				1		
7			1				1			
8		1				1				
9	1				1					
10		1				1				
11			1				1			
12				1				1		
13					1				1	
14				1				1		
15			1				1			
16		1				1				
17	1				1					
18		1				1				
19			1				1			
20				1				1		
21					1				1	
22				1				1		
23			1				1			
24		1				1				

* TOD: Time of the Day

C. PATROL VESSEL SENSOR COVERAGE

The table below presents the estimated PV sensor platform covering each sector of the Singapore Strait for each hour of the day, given its assumed patrol route.

TOD*	MOBILE CPC SENSOR PLATFORM									
SECTOR	71	72	73	81	82	83	91	92	93	94
1	1									
2	1									
3	1									
4	1									
5	1									
6	1									
7	1									
8	1									
9	1									
10	1									
11	1									
12	1									
13	1									
14	1									
15	1									
16	1									
17	1									
18	1									
19	1									
20	1									
21	1									
22	1									
23	1									
24	1									

* TOD: Time of the Day

D. MARITIME PATROL AIRCRAFT SENSOR COVERAGE

The table below presents the estimated MPaA sensor platform covering each sector of the Singapore Strait for each hour of the day. The aircraft is assumed to patrol the full Singapore Strait several times within an hour.

TOD*	MOBILE CPC SENSOR PLATFORM									
SECTOR	71	72	73	81	82	83	91	92	93	94
1										
2										
3										
4										
5										
6										
7										
8										
9	1									
10	1									
11										
12										
13										
14										
15										
16										
17										
18										
19										
20	1									
21										
22										
23										
24										

* TOD: Time of the Day

E. COMPLETE SENSOR COVERAGE IN THE SINGAPORE STRAIT

1. Case Without Sensor Failures

The table below presents the total number of sensor platforms covering each sector of the Singapore Strait for each hour of the day, assuming there are no sensor-platform failures.

TOD*	TOTAL SENSOR PLATFORMS									
SECTOR	71	72	73	81	82	83	91	92	93	94
1	3	2	3	4	6	5	3	2	1	0
2	3	4	3	4	5	5	4	2	1	0
3	2	4	5	4	5	4	4	3	1	0
4	2	3	5	6	5	4	3	3	2	0
5	1	3	4	6	7	4	3	2	2	1
6	1	2	5	6	6	5	3	3	2	0
7	1	3	4	5	6	5	5	3	1	0
8	2	3	3	4	6	6	5	3	1	0
9	3	3	4	5	7	7	5	4	3	1
10	3	4	4	5	6	6	6	4	3	2
11	1	3	4	4	5	4	4	4	2	1
12	1	2	4	5	5	4	3	3	3	1
13	1	2	3	5	6	4	3	2	2	2
14	1	2	4	5	5	4	3	3	3	1
15	1	3	4	4	5	4	4	4	2	1
16	2	3	3	4	5	5	5	3	2	1
17	2	2	3	4	6	6	4	3	2	0
18	2	3	3	4	6	6	5	3	1	0
19	1	3	4	5	6	5	5	3	1	0
20	2	3	6	7	7	6	4	4	3	1
21	1	3	4	6	7	4	3	2	2	1
22	2	3	5	6	5	4	3	3	2	0
23	2	4	5	4	5	4	4	3	1	0
24	3	4	3	4	5	5	4	2	1	0
MIN	1	2	3	4	5	4	3	2	1	0
MAX	3	4	6	7	7	7	6	4	3	2

* TOD: Time of the Day

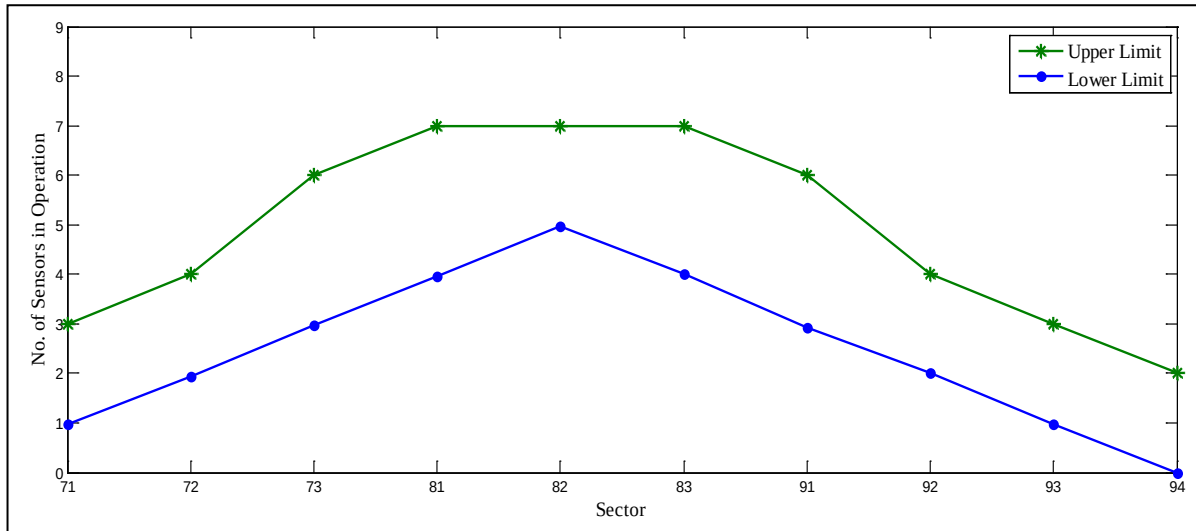
2. Case With Sensor Failures

Taking into consideration random sensor-platform failures, a Monte Carlo simulation of 50 runs over a duration of 30 days was carried out. The table below presents the total number of sensor platforms covering each sector of the Singapore Strait for each hour of the day, given random sensor-platform failures and a repair turnaround time of two hours.

TOD*	TOTAL SENSOR PLATFORMS									
SECTOR	71	72	73	81	82	83	91	92	93	94
1	3	2	3	4	6	5	3	2	1	0
2	3	4	3	4	5	5	4	2	1	0
3	2	4	5	4	5	4	4	3	1	0
4	2	3	5	6	5	4	3	3	2	0
5	1	3	4	6	7	4	3	2	2	1
6	1	2	5	6	6	5	3	3	2	0
7	1	3	4	5	6	5	5	3	1	0
8	2	3	3	4	6	6	5	3	1	0
9	3	3	4	5	7	7	5	4	3	1
10	3	4	4	5	6	6	6	4	3	2
11	1	3	4	4	5	4	4	4	2	1
12	1	2	4	5	5	4	3	3	3	1
13	1	2	3	5	6	4	3	2	2	2
14	1	2	4	5	5	4	3	3	3	1
15	1	3	4	4	5	4	4	4	2	1
16	2	3	3	4	5	5	5	3	2	1
17	2	2	3	4	6	6	4	3	2	0
18	2	3	3	4	6	6	5	3	1	0
19	1	3	4	5	6	5	5	3	1	0
20	2	3	6	7	7	6	4	4	3	1
21	1	3	4	6	7	4	3	2	2	1
22	2	3	5	6	5	4	3	3	2	0
23	2	4	5	4	5	4	4	3	1	0
24	3	4	3	4	5	5	4	2	1	0
MIN	1	2	3	4	5	4	3	2	1	0
MAX	3	4	6	7	7	7	6	4	3	2

* TOD: Time of the Day

The minimum and maximum number of sensor platforms for each sector is shown graphically below.



a. *Sensor Coverage Simulation Source Code*

```
% Store the sensor coverage data in the file "SensorCoverageData.mat"
% Variable "allsensordata" includes both fixed and mobile sensors
% Variable "fixedsensordata" includes only fixed sensors
% Variable "mobilesensordata" includes only mobile sensors
```

```
% Import the Variables into the Workspace
load SensorCoverageData
```

```
% Indicate the duration of simulation (in days)
DurSim = 30;
```

```
for s = 1:DurSim
    sensorsdatabase(1,s) = struct('day',0);
```

```
% Create new variable that accounts for the random failures
newallsensorsdata = allsensorsdata;
```

```
% Random allocation of failures to the sensors
NumF = 1; % Indicate the number of failures over a day
DurF = 2; % Indicate the duration of failure (i.e. TAT) in hours
```

```
for i = 1:NumF
```

```

    % For a 24-hour period (number of rows for the variable)
    a(i) = randi([1,(24-DurF)+1]); %Bounding the failure to within the
day

    % For 10 sectors of coverage (number of columns for the variable)
    b(i) = randi([1,10]);

    % Check for no sensor coverage for the selection
    while (newallsensorsdata(a(i), b(i)) == 0)
        a(i) = randi([1,(24-DurF)+1]);
        b(i) = randi([1,10]);
    end

    % Account for the failure duration
    for j = 1:DurF
        if (newallsensorsdata(a(i)+(j-1) , b(i)) == 0)
            newallsensorsdata(a(i)+(j-1) , b(i)) = 0;
        else
            newallsensorsdata(a(i)+(j-1) , b(i)) = newallsensorsdata(a(i)+(j-
1) , b(i))-1;
        end
    end
end

sensorsdatabase(1,s).day = newallsensorsdata;

end

% Calculate the average sensor coverage for the duration of simulation
sensorsum = 0;
for s = 1:DurSim

    sensorsum = sensorsum + sensorsdatabase(s).day;

end

sensormean = sensorsum/DurSim;

```

APPENDIX B. SIMULATION PROGRAM SOURCE CODE

A. BLOCK 1: CENTRALIZED DATA FUSION ARCHITECTURE USING KALMAN FILTER

% This file executes the Monte-Carlo simulation

% Number of Monte-Carlo simulation runs
totalruns = 50;

% Initialise variables

sumfirstleftTSS(1:totalruns) = NaN;
sumfirstleftTSSntoJIHigh(1:totalruns) = NaN;
sumfirstleftTSSntoJIModerate(1:totalruns) = NaN;
sumfirstleftPL(1:totalruns) = NaN;
sumfirstleftPLntoJIHigh(1:totalruns) = NaN;
sumfirstleftPLntoJIModerate(1:totalruns) = NaN;

% Execute each run by generating new observations or using previous ones

% Produce the fused track for each run

% Then pool the tracks together for analysis

for run =1:totalruns

 buf=sprintf('Run: =%d',run); disp(buf);

 example_sim; %generate new observations

 datastore9(run).observations=observations; % store observations to be executed by
other filters

 observations = datastore9(run).observations;

 run_filt;

 sumfirstleftTSS(run) = firstleftTSS;

 sumfirstleftTSSntoJIHigh(run) = firstleftTSSntoJIHigh;

 sumfirstleftTSSntoJIModerate(run) = firstleftTSSntoJIModerate;

 sumfirstleftPL(run) = firstleftPL;

 sumfirstleftPLntoJIHigh(run) = firstleftPLntoJIHigh;

 sumfirstleftPLntoJIModerate(run) = firstleftPLntoJIModerate;

 sumkdata(run,,:) = kdata;

 sumesttimesigma(run,1,:) = esttimesigma;

 lowest(run) = min(sumkdata(run,7,1000:MAX_TIME/10));

end

% Calculate Probability of Impact

Cxtemp = squeeze(sumkdata(:,8,:));

Cytemp = squeeze(sumkdata(:,9,:));

Xtemp = squeeze(sumkdata(:,10,:));

```

Ytemp = squeeze(sumkdata(:,11,:));
probability = zeros(1,ntime);

warning off;
for l = 1:ntime

    Cx = diag(Cxtemp(:,l))*(180^2);
    Cy = diag(Cxtemp(:,l))*(180^2);
    X = (Xtemp(:,l)- 302.1144444)*180;
    Y = (Ytemp(:,l)- 347.7716667)*180;
    W = ones(totalruns,1);
    sigma_xbar_tgo = inv((W')*inv(Cx)*W);
    sigma_ybar_tgo = inv((W')*inv(Cy)*W);
    sigma = sqrt(sigma_xbar_tgo);
    xbar_tgo = sigma_xbar_tgo*(((W')*inv(Cx)*X));% Calculate mean impact pt
    ybar_tgo = sigma_ybar_tgo*(((W')*inv(Cy)*Y));% Calculate mean impact pt
    r_not = sqrt(xbar_tgo^2 + ybar_tgo^2);
    R = 300;

    g = zeros(1,100);
    h = zeros(1,100);
    probtemp = 0;

    recur = 1;
    g_not = exp(-(r_not^2)/(2*sigma^2)); % g_not
    h_not = 1 - exp(-(R^2)/(2*sigma^2)); % h_not
    probtemp_not = g_not*h_not;

    g(1) = (1/1)*((r_not^2)/(2*sigma^2))*g_not;
    h(1) = -(1/factorial(1))*(((R^2)/(2*sigma^2))^(1))*exp(-(R^2)/(2*sigma^2)) + h_not;
    probtemp = probtemp_not + g(1)*h(1);

    while (recur <= 100)
        g(recur+1) = (1/(recur+1))*((r_not^2)/(2*sigma^2))*g(recur);
        h(recur+1) = (-1/factorial(recur+1))*(((R^2)/(2*sigma^2))^(recur+1))*exp(-(R^2)/(2*sigma^2)) + h(recur);
        probtemp = probtemp + g(recur+1)*h(recur+1);
        recur = recur + 1;
    end

    probability(1,l) = probtemp;
end

% Initialise variables
count1 = 0;

```

```

count2 = 0;
count3 = 0;
count4 = 0;
count5 = 0;
count6 = 0;
SsumfirstleftTSS = 0;
SsumfirstleftTSSntoJIHigh = 0;
SsumfirstleftTSSntoJIModerate = 0;
SsumfirstleftPL = 0;
SsumfirstleftPLntoJIHigh = 0;
SsumfirstleftPLntoJIModerate = 0;

% Take the average of the pooled tracks
for m = 1:totalruns
    if ~isnan(sumfirstleftTSS(m))
        count1 = count1+1;
        SsumfirstleftTSS=SsumfirstleftTSS+sumfirstleftTSS(m);
    end

    if ~isnan(sumfirstleftTSSntoJIHigh(m))
        count2 = count2+1;

SsumfirstleftTSSntoJIHigh=SsumfirstleftTSSntoJIHigh+sumfirstleftTSSntoJIHigh(m);
    end

    if ~isnan(sumfirstleftTSSntoJIModerate(m))
        count3 = count3+1;

SsumfirstleftTSSntoJIModerate=SsumfirstleftTSSntoJIModerate+sumfirstleftTSSntoJIM
oderate(m);
    end

    if ~isnan(sumfirstleftPL(m))
        count4 = count4+1;
        SsumfirstleftPL=SsumfirstleftPL+sumfirstleftPL(m);
    end

    if ~isnan(sumfirstleftPLntoJIHigh(m))
        count5 = count5+1;
        SsumfirstleftPLntoJIHigh=SsumfirstleftPLntoJIHigh+sumfirstleftPLntoJIHigh(m);
    end

    if ~isnan(sumfirstleftPLntoJIModerate(m))
        count6 = count6+1;

```

```

SsumfirstleftPLntoJIModerate=SsumfirstleftPLntoJIModerate+sumfirstleftPLntoJIModer
ate(m);
    end
end

```

```

meanfirstleftTSS = SsumfirstleftTSS/count1;
meanfirstleftTSSntoJIHigh = SsumfirstleftTSSntoJIHigh/count2;
meanfirstleftTSSntoJIModerate = SsumfirstleftTSSntoJIModerate/count3;
meanfirstleftPL = SsumfirstleftPL/count4;
meanfirstleftPLntoJIHigh = SsumfirstleftPLntoJIHigh/count5;
meanfirstleftPLntoJIModerate = SsumfirstleftPLntoJIModerate/count6;
meantruetimedata = mean(sumkdata(:,1,:));
meandistuncert = mean(sumkdata(:,2,:));
meandistapart = mean(sumkdata(:,3,:));
meanesttimedata = mean(sumkdata(:,6,:));

```

% Calculate the sample variance of the estimated time to impact

```

for p = 1:ntime
    sumtimediff=0;
    timediff=0;
    for q = 1:totalruns
        timediff = (sumkdata(q,6,p)-meanesttimedata(p))^2;
        sumtimediff=sumtimediff+timediff;
    end
    meansampletimesigma(p) = sqrt(sumtimediff /(totalruns-1));
end
warning on;

figure(30);
clf;
plot(meantruetimedata(:),'meandistuncert(:)','b',meantruetimedata(:),'meandistapart(:)','g')
legend('Distance Uncertainty','Distance Apart');
buf=sprintf('Estimated End Point');
title(buf)
xlim([min(lowest)*DT MAX_TIME-1*DT])
ylim([0 1000])
xlabel('Simulated Time (s)')
ylabel('Distance (m)')

```

```

figure(31);
clf;
plot(meantruetimedata(:),'abs((MAX_TIME-meantruetimedata(:))-
meanesttimedata(:))','b',meantruetimedata(:),meansampletimesigma,'r' )
legend('Estimated Error','Standard Deviation');

```



```

buf=sprintf('Error between Time Estimate and Truth');
title(buf)
xlim([min(lowest)*DT MAX_TIME-1*DT])
ylim([0 100])
xlabel('Simulated Time (s)')
ylabel('Time (s)')

figure(32);
clf;
plot(meantruetime(:)',probability(:),'k')
legend('Probability of Impact');
buf=sprintf('Probability of Impact');
title(buf)
xlim([min(lowest)*DT MAX_TIME-1*DT])
ylim([0 1])
xlabel('Simulated Time (s)')
ylabel('Probability of Impact')

```

```

% This file executes the Monte-Carlo simulation and incorporates dataloss
% (i.e. observations loss) due to communication bandwidth limitations

% Number of Monte-Carlo simulation runs
totalruns = 10;

% Initialise variables
sumfirstleftTSS(1:totalruns) = NaN;
sumfirstleftTSSntoJIHigh(1:totalruns) = NaN;
sumfirstleftTSSntoJIModerate(1:totalruns) = NaN;
sumfirstleftPL(1:totalruns) = NaN;
sumfirstleftPLntoJIHigh(1:totalruns) = NaN;
sumfirstleftPLntoJIModerate(1:totalruns) = NaN;

% Execute each run by generating new observations or using previous ones
% Inject data loss
% Produce the fused track for each run
% Then pool the tracks together for analysis
for run = 1:totalruns
    buf=sprintf('Run: =%d',run); disp(buf);
    example_sim; %generate new observations
    observations = datastore(run).observations; % using previous observations

    % Since previous observations (with loss) is used, there is no need to
    % re-inject data loss. If new observations (with no loss) are made, the
    % following codes are necessary to inject data loss.
    %-----
    %   % simulate data loss due to bandwidth limitations
    %   for i = 1:10
    %       j = randi(MAX_TIME/DT-1060-3)+1060;
    %
    %       datalossduration = randi(3);
    %       for k = 1:datalossduration
    %           for n = 1:NUM_PLATFORMS
    %               observations(n,j+k).report(2:5) = 99999;
    %           end
    %       end
    %   end
    %   datastoreloss(run).observations=observations;
    %-----

run_filt;
sumfirstleftTSS(run) = firstleftTSS;
sumfirstleftTSSntoJIHigh(run) = firstleftTSSntoJIHigh;

```

```

sumfirstleftTSSntoJIModerate(run) = firstleftTSSntoJIModerate;
sumfirstleftPL(run) = firstleftPL;
sumfirstleftPLntoJIHigh(run) = firstleftPLntoJIHigh;
sumfirstleftPLntoJIModerate(run) = firstleftPLntoJIModerate;
sumkdata(run,,:) = kdata;
sumesttimesigma(run,1,:) = esttimesigma;
lowest(run) = min(sumkdata(run,7,1000:MAX_TIME/10));
end

% Calculate Probability of Impact
Cxtemp = squeeze(sumkdata(:,8,:));
Cytemp = squeeze(sumkdata(:,9,:));
Xtemp = squeeze(sumkdata(:,10,:));
Ytemp = squeeze(sumkdata(:,11,:));
probability = zeros(1,ntime);

warning off;
for l = 1:ntime

    Cx = diag(Cxtemp(:,l))*(180^2);
    Cy = diag(Cytemp(:,l))*(180^2);
    X = (Xtemp(:,l)- 302.1144444)*180;
    Y = (Ytemp(:,l)- 347.7716667)*180;
    W = ones(totalruns,1);
    sigma_xbar_tgo = inv((W')*inv(Cx)*W);
    sigma_ybar_tgo = inv((W')*inv(Cy)*W);
    sigma = sqrt(sigma_xbar_tgo);
    xbar_tgo = sigma_xbar_tgo*(((W')*inv(Cx)*X)); % Calculate mean impact pt
    ybar_tgo = sigma_ybar_tgo*(((W')*inv(Cy)*Y)); % Calculate mean impact pt
    r_not = sqrt(xbar_tgo^2 + ybar_tgo^2);
    R = 300;

    g = zeros(1,100);
    h = zeros(1,100);
    probtemp = 0;

    recur = 1;
    g_not = exp(-(r_not^2)/(2*sigma^2)); % g_not
    h_not = 1 - exp(-(R^2)/(2*sigma^2)); % h_not
    probtemp_not = g_not*h_not;

    g(1) = (1/1)*((r_not^2)/(2*sigma^2))*g_not;
    h(1) = -(1/factorial(1))*(((R^2)/(2*sigma^2))^(1))*exp(-(R^2)/(2*sigma^2)) + h_not;
    probtemp = probtemp_not + g(1)*h(1);

```

```

while (recur <= 100)
    g(recur+1) = (1/(recur+1))*((r_not^2)/(2*sigma^2))*g(recur);
    h(recur+1) = (-1/factorial(recur+1))*(((R^2)/(2*sigma^2))^(recur+1))*exp(-(R^2)/(2*sigma^2)) + h(recur);
    probtemp = probtemp + g(recur+1)*h(recur+1);
    recur = recur + 1;
end

probability(1,l) = probtemp;
end

% Initialise variables
count1 = 0;
count2 = 0;
count3 = 0;
count4 = 0;
count5 = 0;
count6 = 0;
SsumfirstleftTSS = 0;
SsumfirstleftTSSntoJIHigh = 0;
SsumfirstleftTSSntoJIModerate = 0;
SsumfirstleftPL = 0;
SsumfirstleftPLntoJIHigh = 0;
SsumfirstleftPLntoJIModerate = 0;

% Take the average of the pooled tracks
for m = 1:totalruns
    if ~isnan(sumfirstleftTSS(m))
        count1 = count1+1;
        SsumfirstleftTSS=SsumfirstleftTSS+sumfirstleftTSS(m);
    end

    if ~isnan(sumfirstleftTSSntoJIHigh(m))
        count2 = count2+1;

        SsumfirstleftTSSntoJIHigh=SsumfirstleftTSSntoJIHigh+sumfirstleftTSSntoJIHigh(m);
    end

    if ~isnan(sumfirstleftTSSntoJIModerate(m))
        count3 = count3+1;

        SsumfirstleftTSSntoJIModerate=SsumfirstleftTSSntoJIModerate+sumfirstleftTSSntoJIModerate(m);
    end
end

```

```

if ~isnan(sumfirstleftPL(m))
    count4 = count4+1;
    SsumfirstleftPL=SsumfirstleftPL+sumfirstleftPL(m);
end

if ~isnan(sumfirstleftPLntoJIHigh(m))
    count5 = count5+1;
    SsumfirstleftPLntoJIHigh=SsumfirstleftPLntoJIHigh+sumfirstleftPLntoJIHigh(m);
end

if ~isnan(sumfirstleftPLntoJIModerate(m))
    count6 = count6+1;

SsumfirstleftPLntoJIModerate=SsumfirstleftPLntoJIModerate+sumfirstleftPLntoJIModerate(m);
end
end

meanfirstleftTSS = SsumfirstleftTSS/count1;
meanfirstleftTSSntoJIHigh = SsumfirstleftTSSntoJIHigh/count2;
meanfirstleftTSSntoJIModerate = SsumfirstleftTSSntoJIModerate/count3;
meanfirstleftPL = SsumfirstleftPL/count4;
meanfirstleftPLntoJIHigh = SsumfirstleftPLntoJIHigh/count5;
meanfirstleftPLntoJIModerate = SsumfirstleftPLntoJIModerate/count6;
meantruetimedata = mean(sumkdata(:,1,:));
meandistuncert = mean(sumkdata(:,2,:));
meandistapart = mean(sumkdata(:,3,:));
meanesttimedata = mean(sumkdata(:,6,:));

% Calculate the sample variance of the estimated time to impact
for p = 1:ntime
    sumtimediff=0;
    timediff=0;
    for q = 1:totalruns
        timediff = (sumkdata(q,6,p)-meanesttimedata(p))^2;
        sumtimediff=sumtimediff+timediff;
    end
    meansampletimesigma(p) = sqrt(sumtimediff /(totalruns-1));
end

figure(30);
clf;
plot(meantruetimedata(:)',meandistuncert(:)', 'b',meantruetimedata(:)',meandistapart(:)', 'g')
legend('Distance Uncertainty','Distance Apart');
buf=sprintf('Estimated End Point');

```

```

title(buf)
xlim([min(lowest)*DT MAX_TIME-1*DT])
ylim([0 1000])
xlabel('Simulated Time (s)')
ylabel('Distance (m)')

figure(31);
clf;
plot(meantruetimedata(:)',abs((MAX_TIME-meantruetimedata(:))-
meanesttimedata(:)'),'b',meantruetimedata(:)',meansampletimesigma','r' )
legend('Estimated Error','Standard Deviation');
buf=sprintf('Error between Time Estimate and Truth');
title(buf)
xlim([min(lowest)*DT MAX_TIME-1*DT])
ylim([0 100])
xlabel('Simulated Time (s)')
ylabel('Time (s)')

figure(32);
clf;
plot(meantruetimedata(:)',probability(:),'k')
legend('Probability of Impact');
buf=sprintf('Probability of Impact');
title(buf)
xlim([min(lowest)*DT MAX_TIME-1*DT])
ylim([0 1])
xlabel('Simulated Time (s)')
ylabel('Probability of Impact')

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This is a script file allows for multi-target multi-sensor tracking

% Define and initialise global variables
globals;
ginit;

buf=sprintf('Generating Target Information..Please Wait\n'); disp(buf);
% Set up time base for target simulations
times=0:DT:MAX_TIME;

% Generate all target's attack destinations
destinations=generate_destinations(times);

% Generate all target trajectories
targets=generate_targets(times);

% Generate platform information
platforms=generate_platforms(times);

% Assign sensors to platforms
sensors=assign_sensors(platforms);

buf=sprintf('Generating Observations...Please Wait\n'); disp(buf);

% Generate observations
observations=generate_observations(sensors,platforms,targets,times);

buf=sprintf('Displaying Information\n'); disp(buf);

% Display track information
post_sim(targets,observations,sensors,platforms,destinations);

% This file defines the global variables

% General information
global TSS_X;      % Traffic Separation Scheme x-coordinates
global TSS_Y;      % Traffic Separation Scheme y-coordinates

```

```

global PL_X;      % Port Limits x-coordinates
global PL_Y;      % Port Limits y-coordinates
global DT;        % simulation time-step
global MAX_TIME;  % maximum simulation time

% Target definitions
global TARGET_TYPE; % defines type of target (xy or V phi)
global NUM_TARGETS; % number of targets to be simulated

% Platform definitions
global PLATFORM; % Platform data [x0, y0, phi0, vel]
global NUM_PLATFORMS; % number of platforms
global MAX_RANGE; % maximum observable range
global SIGMA_XDOT;
global SIGMA_YDOT;

% Destination definitions
global DESTINATION; % Destination data [x0, y0, phi0, vel]
global NUM_DESTINATIONS; % number of destinations

% State and Sensor definitions
global SIGMA_Q; % process noise standard deviation for feature model
global SIGMA_RANGE_MPA; % Range observation noise
global SIGMA_BEARING_MPA; % Bearing observation noise
global SIGMA_RANGE_MPAC; % Range observation noise
global SIGMA_BEARING_MPAC; % Bearing observation noise
global SIGMA_RANGE_PV; % Range observation noise
global SIGMA_BEARING_PV; % Bearing observation noise
global SIGMA_RANGE_PCG; % Range observation noise
global SIGMA_BEARING_PCG; % Bearing observation noise
global Rfilter_MPA; % observation noise covariance for MPA
global Rfilter_MPAC; % observation noise covariance for MPAC
global Rfilter_PV; % observation noise covariance for PV
global Rfilter_PCG; % observation noise covariance for PCG
global F; % state transition equation
global XSIZE; % state dimension
global ZSIZE; % observation dimension

% Results "database"
global kdata;
global esttimesigma;
global startplot;
global firstleftTSS;
global firstleftTSSntoJIHigh;
global firstleftTSSntoJIModerate;

```



```

global firstleftPL;
global firstleftPLntoJIHigh;
global firstleftPLntoJIModerate;

% This file initializes the global variables

% General polygon for Traffic Separation Scheme (TSS)
X = [738.9629; 700.8405; 503.0354; 343.3085; 301.0754; 188.7756; 142.3798;
291.7565; 356.6971; 474.1785; 703.9221; 738.9538; 738.9629];
Y = [443.8223; 412.1040; 377.3690; 303.7337; 330.3691; 359.0929; 316.1193;
230.0498; 289.3964; 351.7903; 379.3511; 408.0001; 443.8223];
TSS_X = X;
TSS_Y = Y;

% General polygon for Port Limits, estimated 0.5 mile beyond the TSS
[PL_X, PL_Y] = bufferm2('xy',X,Y,5,'out');

MAX_TIME=11500;          % simulation duration
DT=10;                   % simulation time-step

% Target definitions
NUM_TARGETS=1;           % number of targets to be tracked

% Platform definitions
NUM_PLATFORMS=5;

% Destination definitions
NUM_DESTINATIONS=1;

% State and Sensor definitions
SIGMA_RANGE_MPA=5;        % 0.5nm range error equivalent
SIGMA_BEARING_MPA=0.0525; % 3 degrees bearing error in radians
SIGMA_RANGE_MPAC=6;       % 0.6nm range error equivalent
SIGMA_BEARING_MPAC=0.0875; % 5 degrees bearing error in radians
SIGMA_RANGE_PV=3;         % 0.3nm range error equivalent
SIGMA_BEARING_PV=0.0525;  % 3 degrees bearing error in radians
SIGMA_RANGE_PCG=4;        % 0.4nm range error equivalent
SIGMA_BEARING_PCG=0.0875; % 5 degrees bearing error in radians
Rfilter_MPA = [SIGMA_RANGE_MPA^2 0;0 SIGMA_RANGE_MPA^2];
Rfilter_MPAC = [SIGMA_RANGE_MPAC^2 0;0 SIGMA_RANGE_MPAC^2];
Rfilter_PV = [SIGMA_RANGE_PV^2 0;0 SIGMA_RANGE_PV^2];
Rfilter_PCG = [SIGMA_RANGE_PCG^2 0;0 SIGMA_RANGE_PCG^2];
XSIZE=4;                  % number of state components
ZSIZE=2;                  % number of sensor measurement components
SIGMA_XDOT=0.00286;       % std dev of 1knot in the x-direction

```

```

SIGMA_YDOT=0.00286;          % std dev of 1knot in the y-direction

% Results "database"
kdata = [7 MAX_TIME/DT+1];
function [latb,lonb] = bufferm2(varargin) %lat,lon,dist,direction,npts,outputformat)
%BUFFERM2 Computes buffer zone around a polygon
%
% [latb,lonb] = bufferm2(lat,lon,dist,direction)
% [latb,lonb] = bufferm2(lat,lon,dist,direction,npts)
% [latb,lonb] = bufferm2(lat,lon,dist,direction,npts,outputformat)
% [xb, yb] = bufferm2('xy',x,y,dist,direction,npts,outputformat)
%
% This function was originally designed as a replacement for the Mapping
% Toolbox function bufferm, which calculates a buffer zone around a
% polygon. The original bufferm function had some serious bugs that could
% result in incorrect buffer results and/or errors, and was also very slow.
% As of R2006b, those bugs have been fixed. However, this version still
% maintains a few advantages over the original:
%
% - Can be applied to polygons in either geographical space (as in
%   bufferm) or in cartesian coordinates.
%
% - Better treatment of polygon holes. The original function simply
%   filled in all holes; this version trims or pads holes according to the
%   buffer width given.
%
% Input and output format is identical to bufferm unless the 'xy' option is
% specified, so it can be used interchangeably.
%
% Input variables:
%
% lat:      Latitude values defining the polygon to be buffered.
%           This can be either a NaN-delimited vector, or a cell
%           array containing individual polygonal contours (each of
%           which is a vector). External contours should be listed
%           in a clockwise direction, and internal contours (holes)
%           in a counterclockwise direction.
%
% lon:      Longitude values defining the polygon to be buffered.
%           Same format as lat.
%
% dist:     Width of buffer, in degrees of arc along the surface
%           (unless 'xy' is used, in which case units correspond to
%           x-y coordinates)
%

```

```

% direction:    'in' or 'out'
%
% npts:         Number of points used to construct the circles around
%               each polygon vertex. If omitted, default is 13.
%
% outputformat: 'vector' (NaN-delimited vectors), 'cutvector'
%               (NaN-clipped vectors with cuts connecting holes to the
%               exterior of the polygon), or 'cell' (cell arrays in
%               which each element of the cell array is a separate
%               polygon), defining format of output. If omitted,
%               default is 'vector'.
%
% 'xy':         If first input is 'xy', then data will be assumed to
%               lie on a cartesian plane rather than on a sphere. Use
%               x and y coordinates as first two inputs rather than lat
%               and lon. Units of x, y, and distance should be the
%               same.
%
% Output variables:
%
% latb:         Latitude values for buffer polygon
%
% lonb:         Longitude values for buffer polygon
%
% Example:
%
% load conus
% tol = 0.1; % Tolerance for simplifying polygon outlines
% [reducedlat, reducedlon] = reducem(gtlakelat, gtlakelon, tol);
% dist = 1; % Buffer distance in degrees
% [latb, lonb] = bufferm2(reducedlat, reducedlon, dist, 'out');
% figure('Renderer','painters')
% usamap({'MN','NY'})
% geoshow(latb, lonb, 'DisplayType', 'polygon', 'FaceColor', 'yellow')
% geoshow(gtlakelat, gtlakelon,...
%         'DisplayType', 'polygon', 'FaceColor', 'blue')
% geoshow(uslat, uslon)
% geoshow(statelat, statelon)
%
% See also:
%
% bufferm, polybool
%
% Copyright 2010 Kelly Kearney

```

```

%-----
% Check input
%-----

error(nargchk(3,7,nargin));

% Determine if geographic or cartesian

if ischar(varargin{1}) && strcmp(varargin{1}, 'xy')
    geo = false;
    param = varargin(2:end);
else
    geo = true;
    param = varargin;
end

% Set defaults if not provided as input

nparam = length(param);

if geo
    [lat, lon, dist] = deal(param{1:3});
else
    [lon, lat, dist] = deal(param{1:3}); % lon = x, lat = y for mental clarity, will switch
back at end
end

if nparam < 4
    direction = 'out';
else
    direction = param{4};
end

if nparam < 5
    npts = 13;
else
    npts = param{5};
end

if nparam < 6
    outputformat = 'vector';
else
    outputformat = param{6};
end

```

```

% Check format and dimensions of input

if ~ismember(direction, {'in', 'out'})
    error('Direction must be either "in" or "out".');
end

if ~ismember(outputformat, {'vector', 'cutvector', 'cell'})
    error('Unrecognized output format flag. ');
end

if ~isnumeric(dist) || numel(dist) > 1
    error('Distance must be a scalar. ');
end

if ~isnumeric(npts) || numel(npts) > 1
    error('Number of points must be a scalar. ');
end

if iscell(lat)
    for il = 1:numel(lat)
        if ~isvector(lat{il}) || ~isvector(lon{il}) || ~isequal(length(lat{il}), length(lon{il}))
            error('Lat (or x) and lon (or y) must be vectors or cells of vectors with identical dimensions');
        end
        lat{il} = lat{il}(:);
        lon{il} = lon{il}(:);
    end
else
    if ~isvector(lat) || ~isvector(lon) || ~isequal(length(lat), length(lon))
        error('Lat (or x) and lon (or y) must be vectors or cells of vectors with identical dimensions');
    end
    lat = lat(:);
    lon = lon(:);
end

%-----
% Split polygon(s) into
% separate faces
%-----

if iscell(lat)
    [lat, lon] = polyjoin(lat, lon); % In case multiple faces in one cell.
end

```

```

[latcells, loncells] = polysplit(lat, lon);

%-----
% Create buffer shapes
%-----

plotflag = 0;

if plotflag

    Plt.x = lon;
    Plt.y = lat;

end

latcrall = cell(0);
loncrall = cell(0);

for ipoly = 1:length(latcells)

    % Circles around each vertex

    if geo
        [latc, lonc] = calccircgeo(latcells{ipoly}, loncells{ipoly}, dist, npts);
    else
        [lonc, latc] = calccirccart(loncells{ipoly}, latcells{ipoly}, dist, npts);
    end

    % Rectangles around each edge

    if geo
        [latr, lonr] = calcrecgeo(latcells{ipoly}, loncells{ipoly}, dist);
    else
        [lonr, latr] = calcreccart(loncells{ipoly}, latcells{ipoly}, dist);
    end

    % Union of circles and rectangles

    if plotflag
        Plt.rectx = lonr;
        Plt.recty = latr;
        Plt.circx = lonc;
        Plt.circy = latc;
    end
end

```

```

[latc, lonc] = multipolyunion(latc, lonc);
[latr, lonr] = multipolyunion(latr, lonr);

if plotflag
    Plt.rectcombox = lonr;
    Plt.rectcomboy = latr;
    Plt.circcombox = lonc;
    Plt.circcomboy = latc;
end

[loncr, latcr] = polybool('+', lonr, latr, lonc, latc);

% Union of new circle/rectangle combo with that from other faces

[loncrall, latcrall] = polybool('+', loncrall, latcrall, loncr, latcr);

% Plotting (for debugging only)

if plotflag

    Plt.allx = loncrall;
    Plt.ally = latcrall;

    if ipoly == 1
        figure;
        plot(Plt.x, Plt.y, 'k', 'linewidth', 2);
        hold on
    end

    plot(cat(2, Plt.rectx{:}), cat(2, Plt.recty{:}), 'b');
    plot(cat(2, Plt.circx{:}), cat(2, Plt.circy{:}), 'r');
    plot(Plt.allx{1}, Plt.ally{1}, 'g', 'linewidth', 2);

end

end

%-----
% Calculate union/difference
%-----

switch direction
case 'out'
    [lonb, latb] = polybool('+', loncells, latcells, loncrall, latcrall);
case 'in'

```

```

    [lonb, latb] = polybool('-', loncells, latcells, loncrall, latcrall);
end

if plotflag
    [Plt.yfinal, Plt.xfinal] = polyjoin(latb, lonb);
    plot(Plt.xfinal, Plt.yfinal, 'linestyle', '--', 'color', [0 .5 0], 'linewidth', 2);
end

%-----
% Reformat output
%-----

if ~geo
    y = latb; % Switch, since cartesian uses opposite order
    x = lonb;
    latb = x;
    lonb = y;
end

switch outputformat
    case 'vector'
        [latb, lonb] = polyjoin(latb, lonb);
    case 'cutvector'
        [latb, lonb] = polycut(latb, lonb);
    case 'cell'
end

%*****
%*****

function [latc, lonc] = calccircgeo(lat, lon, radius, npts)
% lat and lon: n x 1 vectors
% radius: scalar

radius = ones(length(lat),1) * radius;
[latc, lonc] = scircle1(lat, lon, radius, [], [], [], npts);
latc = num2cell(latc, 1);
lonc = num2cell(lonc, 1);

function [latr, lonr] = calcrecgeo(lat, lon, halfwidth)
% lat and lon: n x 1 vectors
% halfwidth: scalar

range = halfwidth * ones(length(lat)-1, 1);

```



```

az = azimuth(lat(1:end-1), lon(1:end-1), lat(2:end), lon(2:end));

[latbl1,lonbl1] = reckon(lat(1:end-1), lon(1:end-1), range, az-90);
[latbr1,lonbr1] = reckon(lat(1:end-1), lon(1:end-1), range, az+90);
[latbl2,lonbl2] = reckon(lat(2:end), lon(2:end), range, az-90);
[latbr2,lonbr2] = reckon(lat(2:end), lon(2:end), range, az+90);

latr = [latbl1 latbl2 latbr2 latbr1 latbl1]';
lonr = [lonbl1 lonbl2 lonbr2 lonbr1 lonbl1]';
latr = num2cell(latr, 1);
lonr = num2cell(lonr, 1);

function [latu, lonu] = multipolyunion(lat, lon)
% lat and lon are n x 1 cell arrays of vectors

latu = lat{1};
lonu = lon{1};

for ip = 2:length(lat)
    [lonu, latu] = polybool('+', lonu, latu, lon{ip}, lat{ip});
end
[latu, lonu] = polysplit(latu, lonu);

function [xc, yc] = calccirccart(x, y, radius, npts)

ang = linspace(0, 2*pi, npts+1);
ang = ang(end-1:-1:1);
xc = bsxfun(@plus, x, radius * cos(ang));
yc = bsxfun(@plus, y, radius * sin(ang));
xc = num2cell(xc', 1);
yc = num2cell(yc', 1);

% if ~ispolycw(x,y)
%   [xc,yc] = poly2ccw(xc,yc);
% end

function [xrec, yrec] = calcreccart(x, y, halfwidth)

dx = diff(x);
dy = diff(y);

is1 = dx >= 0 & dy >= 0;
is2 = dx < 0 & dy >= 0;

```

```

is3 = dx < 0 & dy < 0;
is4 = dx >= 0 & dy < 0;

ish1 = dy == 0 & dx > 0;
ish2 = dy == 0 & dx < 0;

theta = zeros(5,1);
theta(is1 | is3) = atan(dy(is1 | is3)./dx(is1 | is3));
theta(is2 | is4) = -atan(dy(is2 | is4)./dx(is2 | is4));

[xl,xr,yl,yr] = deal(zeros(size(dx)));

xl(is1) = -halfwidth * sin(theta(is1));
xr(is1) = halfwidth * sin(theta(is1));
yl(is1) = halfwidth * cos(theta(is1));
yr(is1) = -halfwidth * cos(theta(is1));

xl(is2) = -halfwidth * sin(theta(is2));
xr(is2) = halfwidth * sin(theta(is2));
yl(is2) = -halfwidth * cos(theta(is2));
yr(is2) = halfwidth * cos(theta(is2));

xl(is3) = halfwidth * sin(theta(is3));
xr(is3) = -halfwidth * sin(theta(is3));
yl(is3) = -halfwidth * cos(theta(is3));
yr(is3) = halfwidth * cos(theta(is3));

xl(is4) = halfwidth * sin(theta(is4));
xr(is4) = -halfwidth * sin(theta(is4));
yl(is4) = halfwidth * cos(theta(is4));
yr(is4) = -halfwidth * cos(theta(is4));

xrec = [xl+x(1:end-1) xl+x(2:end) xr+x(2:end) xr+x(1:end-1) xl+x(1:end-1)];
yrec = [yl+y(1:end-1) yl+y(2:end) yr+y(2:end) yr+y(1:end-1) yl+y(1:end-1)];

xrec = num2cell(xrec, 2);
yrec = num2cell(yrec, 2);

```

% This function draws a circle for a given center and radius

function bubble = circle(center,radius)

Resolution = 100;

THETA=linspace(0,2*pi,Resolution);

RHO=ones(1,Resolution)*radius;

[X,Y] = pol2cart(THETA,RHO);

bubble.X=X+center(1);

bubble.Y=Y+center(2);

```
% This function generates the SAW's intended destinations  
% (i.e. Areas A1, A2, and A3 of Jurong Island)
```

```
function destinations = generate_destinations(times)
```

```
% Define the global variables  
globals;
```

```
% Define the destination locations
```

```
for i=1:NUM_DESTINATIONS
```

```
    % Area A1
```

```
    if i==1
```

```
        destinations(1,i) = make_destination;
```

```
        destinations(1,i).time=times(1);
```

```
        destinations(1,i).id=i;
```

```
        destinations(1,i).x = 302.1144444;
```

```
        destinations(1,i).y = 347.7716667;
```

```
    % Now iterate for all times
```

```
    [temp,ntimes]=size(times);
```

```
    for n=2:ntimes
```

```
        destinations(n,i)=destination_step(destinations(n-1,i),times(n));
```

```
    end
```

```
end
```

```
    % Area A2
```

```
    if i==2
```

```
        destinations(1,i) = make_destination;
```

```
        destinations(1,i).time=times(1);
```

```
        destinations(1,i).id=i;
```

```
        destinations(1,i).x = 322.7311111;
```

```
        destinations(1,i).y = 367.2116667;
```

```
    % Now iterate for all times
```

```
    [temp,ntimes]=size(times);
```

```
    for n=2:ntimes
```

```
        destinations(n,i)=destination_step(destinations(n-1,i),times(n));
```

```
    end
```

```
end
```

```
    % Area A3
```

```
    if i==3
```

```
        destinations(1,i) = make_destination;
```

```
        destinations(1,i).time=times(1);
```

```
destinations(1,i).id=i;

destinations(1,i).x = 343.3450000;
destinations(1,i).y = 382.5572222;

% Now iterate for all times
[temp,ntimes]=size(times);
for n=2:ntimes
    destinations(n,i)=destination_step(destinations(n-1,i),times(n));
end
end
end
```

```
% This function initialises a new destination  
  
function new=make_destination  
  
new=struct('id',0,'time',0,'x',0,'y',0,'phi',0,'vel',0,'gamma',0);
```

```

% This function is to step a destination

function next_destination=destination_step(destination,time)

globals;

% This is a no motion model

dt=time-destination.time;
next_destination = make_destination; % space for destination data structure
next_destination.id=destination.id;
next_destination.time = time;
next_destination.x = destination.x;
next_destination.y = destination.y;
next_destination.phi = destination.phi;
next_destination.vel = destination.vel;
next_destination.gamma = destination.gamma;

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function generates true target information for later observation
% and tracking. Dynamics of target are contained in "target_step", number
% targets is defined by the globally defined variable NUM_TARGETS. A data
% point is generated at each of a set of "times."

function targets = generate_targets(times)

% Define the global variables
globals;

% Define a SAW at pre-fixed location in the TSS
for i=1
    targets(1,i) = make_target;
    targets(1,i).time=times(1);
    targets(1,i).id=i;
    targets(1,i).x = 738.9616667;
    targets(1,i).y = 438.7050000;
    targets(1,i).gamma=0;

    % Now iterate for all times
    [temp,ntimes]=size(times);
    for n=2:ntimes
        targets(n,i)=target_step_SAW_path(targets(n-1,i),times(n));
    end
end

% Create non-SAW contacts at random locations within the TSS
% Not used since this thesis covers a single target
for i=2:NUM_TARGETS
    targets(1,i) = make_target;
    targets(1,i).time=times(1);
    targets(1,i).id=i;

    x = TSS_X(1)*rand;
    y = TSS_Y(1)*rand;

    while (inpolygon (x, y,TSS_X, TSS_Y) ~= 1)

```



```

    x = TSS_X(1)*rand;
    y = TSS_Y(1)*rand;
end

    targets(1,i).x = x;
    targets(1,i).y = y;
    targets(1,i).phi = 2*pi*rand -pi;
    targets(1,i).vel = MIN_TARGET_VEL + (MAX_TARGET_VEL-
MIN_TARGET_VEL)*rand;
    targets(1,i).gamma=0;

    % Now iterate for all times
    [temp,ntimes]=size(times);
    for n=2:ntimes
        targets(n,i)=target_step(targets(n-1,i),times(n));
    end
end

```

```
% Acknowledgement:  
% This code originates from H. Durrant-Whyte  
% Australian Centre for Field Robotics  
% The University of Sydney, Australia  
  
% This function initialises a new target  
  
function new=make_target  
  
new=struct('id',0,'time',0,'x',0,'y',0,'phi',0,'vel',0,'gamma',0);
```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This is the trajectory taken by the SAW

function next_target=target_step_SAW_path(target,time)

globals;

dt=time-target.time;
next_target = make_target;
next_target.id=target.id;
next_target.time = time;

% target.vel = 0.042870370 (in m/s which equates to 15knots)
% target.vel = 0.059160494 (in m/s which equates to 20knots)
% target.vel = 0.060018519 (in m/s which equates to 21knots)
% target.vel = 0.062876543 (in m/s which equates to 22knots)

if (next_target.time >= 0 && next_target.time < 1086)
    next_target.x = target.x + dt*0.042870370*cos(-pi+0.720);
    next_target.y = target.y + dt*0.042870370*sin(-pi+0.720);
    next_target.phi = -pi+0.720;
    next_target.vel = 0.042870370;

elseif (next_target.time >= 1086 && next_target.time < 5890)
    next_target.x = target.x + dt*0.042870370*cos(-pi+0.170);
    next_target.y = target.y + dt*0.042870370*sin(-pi+0.170);
    next_target.phi = -pi+0.170;
    next_target.vel = 0.042870370;

elseif (next_target.time >= 5890 && next_target.time < 10199)
    next_target.x = target.x + dt*0.042870370*cos(-pi+0.430);
    next_target.y = target.y + dt*0.042870370*sin(-pi+0.430);
    next_target.phi = -pi+0.430;
    next_target.vel = 0.042870370;

elseif (next_target.time >= 10199 && next_target.time < 11051)
    next_target.x = target.x + dt*0.042870370*cos(pi-0.770);
    next_target.y = target.y + dt*0.042870370*sin(pi-0.770);

```

```
next_target.phi = pi-0.770;
next_target.vel = 0.042870370;

elseif (next_target.time >= 11051 && next_target.time < 11500)
    next_target.x = target.x + dt*0.059160494*cos(pi-1.360);
    next_target.y = target.y + dt*0.059160494*sin(pi-1.360);
    next_target.phi = pi-1.360;
    next_target.vel = 0.059160494;

elseif (next_target.time >= 11500)
    next_target.x = target.x;
    next_target.y = target.y;
    next_target.phi = 0;
    next_target.vel = 0;
end
```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function generates trajectories for platforms

function platforms = generate_platforms(times)

% Define the global variables
globals;

% Define 5 fixed platform initial locations

% Model the MPA radar stations
for i=1:NUM_PLATFORMS
    if i==1
        platforms(1,i) = make_platform;
        platforms(1,i).time=times(1);
        platforms(1,i).id=i;
        platforms(1,i).x = 285.4700000;
        platforms(1,i).y = 360.8527778;
        platforms(1,i).phi = 2*pi*rand -pi;
        platforms(1,i).vel = 0;
        platforms(1,i).gamma=0;

        % Now iterate for all times
        [temp,ntimes]=size(times);
        for n=2:ntimes
            platforms(n,i)=platform_step(platforms(n-1,i),times(n));
        end
    end

    if i==2
        platforms(1,i) = make_platform;
        platforms(1,i).time=times(1);
        platforms(1,i).id=i;
        platforms(1,i).x = 342.7972222;
        platforms(1,i).y = 311.9850000;
        platforms(1,i).phi = 2*pi*rand -pi;
        platforms(1,i).vel = 0;
        platforms(1,i).gamma=0;
    end
end

```

```

% Now iterate for all times
[temp,ntimes]=size(times);
for n=2:ntimes
    platforms(n,i)=platform_step(platforms(n-1,i),times(n));
end
end

if i==3
    platforms(1,i) = make_platform;
    platforms(1,i).time=times(1);
    platforms(1,i).id=i;
    platforms(1,i).x = 413.1333333;
    platforms(1,i).y = 349.8194444;
    platforms(1,i).phi = 2*pi*rand -pi;
    platforms(1,i).vel = 0;
    platforms(1,i).gamma=0;

    % Now iterate for all times
    [temp,ntimes]=size(times);
    for n=2:ntimes
        platforms(n,i)=platform_step(platforms(n-1,i),times(n));
    end
end

if i==4
    platforms(1,i) = make_platform;
    platforms(1,i).time=times(1);
    platforms(1,i).id=i;
    platforms(1,i).x = 461.4761111;
    platforms(1,i).y = 403.4261111;
    platforms(1,i).phi = 2*pi*rand -pi;
    platforms(1,i).vel = 0;
    platforms(1,i).gamma=0;

    % Now iterate for all times
    [temp,ntimes]=size(times);
    for n=2:ntimes
        platforms(n,i)=platform_step(platforms(n-1,i),times(n));
    end
end

if i==5
    platforms(1,i) = make_platform;
    platforms(1,i).time=times(1);

```

```

platforms(1,i).id=i;
platforms(1,i).x = 517.0844444;
platforms(1,i).y = 421.9305556;
platforms(1,i).phi = 2*pi*rand -pi;
platforms(1,i).vel = 0;
platforms(1,i).gamma=0;

% Now iterate for all times
[temp,ntimes]=size(times);
for n=2:ntimes
    platforms(n,i)=platform_step(platforms(n-1,i),times(n));
end
end

% Define 3 mobile platform initial locations

% Model the CPC
if i==6
    platforms(1,i) = make_platform;
    platforms(1,i).time=times(1);
    platforms(1,i).id=i;
    platforms(1,i).x = 474.1785056;
    platforms(1,i).y = 351.7903000;
    platforms(1,i).phi = 2*pi*rand -pi;
    platforms(1,i).vel = 0;
    targets(1,i).gamma=0;
    % Now iterate for all times
    [temp,ntimes]=size(times);
    for n=2:ntimes
        platforms(n,i)=platform_step_PCG1_path(platforms(n-1,i),times(n));
    end
end

if i==7
    platforms(1,i) = make_platform;
    platforms(1,i).time=times(1);
    platforms(1,i).id=i;
    platforms(1,i).x = 474.1785056;
    platforms(1,i).y = 351.7903000;
    platforms(1,i).phi = 2*pi*rand -pi;
    platforms(1,i).vel = 0;
    targets(1,i).gamma=0;

    % Now iterate for all times
    [temp,ntimes]=size(times);

```

```

    for n=2:ntimes
        platforms(n,i)=platform_step_PCG2_path(platforms(n-1,i),times(n));
    end
end

% Model the PV
if i==8
    platforms(1,i) = make_platform;
    platforms(1,i).time=times(1);
    platforms(1,i).id=i;
    platforms(1,i).x = 474.1785056;
    platforms(1,i).y = 351.7903000;
    platforms(1,i).phi = 2*pi*rand -pi;
    platforms(1,i).vel = 0;
    targets(1,i).gamma=0;

    % Now iterate for all times
    [temp,ntimes]=size(times);
    for n=2:ntimes
        platforms(n,i)=platform_step_PV_path(platforms(n-1,i),times(n));
    end
end

% Given the speed and sensor coverage of the MPaA,
% its surveying of the small TSS can be modeled as a "fixed" sensor
% with complete of the TSS during its operation.
% Model the MPaA
if i==9
    platforms(1,i) = make_platform; % make the platform data structure
    platforms(1,i).time=times(1);
    platforms(1,i).id=i;
    platforms(1,i).x = 474.1785056;
    platforms(1,i).y = 351.7903000;
    platforms(1,i).phi = 2*pi*rand -pi;
    platforms(1,i).vel = 0;
    targets(1,i).gamma=0;

    % Now iterate for all times
    [temp,ntimes]=size(times);
    for n=2:ntimes
        platforms(n,i)=platform_step_MPAC(platforms(n-1,i),times(n));
    end
end
end
end

```



```
% Acknowledgement:  
% This code originates from H. Durrant-Whyte  
% Australian Centre for Field Robotics  
% The University of Sydney, Australia  
  
% This function initialises a new platform  
  
function new=make_platform  
  
new=struct('id',0,'time',0,'x',0,'y',0,'phi',0,'vel',0,'gamma',0);
```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% This is a no motion platform model for MPA radar station

function next_platform=platform_step(platform,time)

globals;

dt=time-platform.time;
next_platform = make_platform;
next_platform.id=platform.id;
next_platform.time = time;
next_platform.x = platform.x;
next_platform.y = platform.y;
next_platform.phi = platform.phi;
next_platform.vel = platform.vel;
next_platform.gamma = platform.gamma;

```

```

% This is the platform motion model for one of the two CPCs for a day

function next_platform=platform_step_PCG1_path(platform,time)

globals;

dt=time-platform.time;
next_platform = make_platform;
next_platform.id=platform.id;
next_platform.time = time;

if (next_platform.time >= 0)
    next_platform.x = 474.1785056;
    next_platform.y = 351.7903000;
    next_platform.phi = platform.phi;
    next_platform.vel = platform.vel;
    next_platform.gamma = platform.gamma;
end

%-----
% This models the platform's patrol path. However, by making the assumption
% that the sensors have extended range to simplify sensor management, this
% modeling is not used here. This will be useful for future work with the
% inclusion of sensor management.
%
% if (next_platform.time >= 0 && next_platform.time < 18000)
%     next_platform.x = platform.x + dt*0.015061617*cos(1.519574188-0.5*pi);
%     next_platform.y = platform.y + dt*0.015061617*sin(1.519574188-0.5*pi);
%     next_platform.phi = 1.519574188-0.5*pi;
%     next_platform.vel = 0.015061617;
%     next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 18000 && next_platform.time < 32400)
%     next_platform.x = platform.x + dt*0.025102695*cos(1.519574188+0.5*pi);
%     next_platform.y = platform.y + dt*0.025102695*sin(1.519574188+0.5*pi);
%     next_platform.phi = 1.519574188+0.5*pi;
%     next_platform.vel = 0.025102695;
%     next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 32400 && next_platform.time < 46800)
%     next_platform.x = platform.x + dt*0.015061617*cos(1.519574188-0.5*pi);
%     next_platform.y = platform.y + dt*0.015061617*sin(1.519574188-0.5*pi);
%     %next_platform.phi = 1.519574188-0.5*pi;
%     %next_platform.vel = 0.015061617;

```

```

%   %next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 46800 && next_platform.time < 61200)
%   next_platform.x = platform.x + dt*0.025102695*cos(1.519574188+0.5*pi);
%   next_platform.y = platform.y + dt*0.025102695*sin(1.519574188+0.5*pi);
%   next_platform.phi = 1.519574188+0.5*pi;
%   next_platform.vel = 0.025102695;
%   next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 61200 && next_platform.time < 75600)
%   next_platform.x = platform.x + dt*0.015061617*cos(1.519574188-0.5*pi);
%   next_platform.y = platform.y + dt*0.015061617*sin(1.519574188-0.5*pi);
%   %next_platform.phi = 1.519574188-0.5*pi;
%   %next_platform.vel = 0.015061617;
%   %next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 75600 && next_platform.time < 86400)
%   next_platform.x = platform.x + dt*0.025102695*cos(1.519574188+0.5*pi);
%   next_platform.y = platform.y + dt*0.025102695*sin(1.519574188+0.5*pi);
%   next_platform.phi = 1.519574188+0.5*pi;
%   next_platform.vel = 0.025102695;
%   next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 86400)
%   next_platform.x = platform.x;
%   next_platform.y = platform.y;
%   next_platform.phi = 0;
%   next_platform.vel = 0;
%   next_platform.gamma = 0;
% end
%-----

```

```

% This is the platform motion model for one of the two CPCs for a day

function next_platform=platform_step_PCG2_path(platform,time)

globals;

dt=time-platform.time;
next_platform = make_platform;
next_platform.id=platform.id;
next_platform.time = time;

if (next_platform.time >= 0)
    next_platform.x = 474.1785056;
    next_platform.y = 351.7903000;
    next_platform.phi = platform.phi;
    next_platform.vel = platform.vel;
    next_platform.gamma = platform.gamma;
end

%-----
% This models the platform's patrol path. However, by making the assumption
% that the sensors have extended range to simplify sensor management, this
% modeling is not used here. This will be useful for future work with the
% inclusion of sensor management.
%
% if (next_platform.time >= 0 && next_platform.time < 18000)
%     next_platform.x = platform.x + dt*0.013140537*cos(-1.330818664+0.5*pi);
%     next_platform.y = platform.y + dt*0.013140537*sin(-1.330818664+0.5*pi);
%     next_platform.phi = -1.330818664+0.5*pi;
%     next_platform.vel = 0.013140537;
%     next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 18000 && next_platform.time < 32400)
%     next_platform.x = platform.x + dt*0.021900894*cos(-1.330818664-0.5*pi);
%     next_platform.y = platform.y + dt*0.021900894*sin(-1.330818664-0.5*pi);
%     next_platform.phi = -1.330818664-0.5*pi;
%     next_platform.vel = 0.021900894;
%     next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 32400 && next_platform.time < 46800)
%     next_platform.x = platform.x + dt*0.013140537*cos(-1.330818664+0.5*pi);
%     next_platform.y = platform.y + dt*0.013140537*sin(-1.330818664+0.5*pi);
%     next_platform.phi = -1.330818664+0.5*pi;
%     next_platform.vel = 0.013140537;

```

```

%   next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 46800 && next_platform.time < 61200)
%   next_platform.x = platform.x + dt*0.021900894*cos(-1.330818664-0.5*pi);
%   next_platform.y = platform.y + dt*0.021900894*sin(-1.330818664-0.5*pi);
%   next_platform.phi = -1.330818664-0.5*pi;
%   next_platform.vel = 0.021900894;
%   next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 61200 && next_platform.time < 75600)
%   next_platform.x = platform.x + dt*0.013140537*cos(-1.330818664+0.5*pi);
%   next_platform.y = platform.y + dt*0.013140537*sin(-1.330818664+0.5*pi);
%   next_platform.phi = -1.330818664+0.5*pi;
%   next_platform.vel = 0.013140537;
%   next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 75600 && next_platform.time < 86400)
%   next_platform.x = platform.x + dt*0.021900894*cos(-1.330818664-0.5*pi);
%   next_platform.y = platform.y + dt*0.021900894*sin(-1.330818664-0.5*pi);
%   next_platform.phi = -1.330818664-0.5*pi;
%   next_platform.vel = 0.021900894;
%   next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 86400)
%   next_platform.x = platform.x;
%   next_platform.y = platform.y;
%   next_platform.phi = 0;
%   next_platform.vel = 0;
%   next_platform.gamma = 0;
% end
%-----

```

```

% This is the platform motion model for PV for a day

function next_platform=platform_step_PV_path(platform,time)

globals;

dt=time-platform.time;
next_platform = make_platform;
next_platform.id=platform.id;
next_platform.time = time;

if (next_platform.time >= 0)
    next_platform.x = 474.1785056;
    next_platform.y = 351.7903000;
    next_platform.phi = platform.phi;
    next_platform.vel = platform.vel;
    next_platform.gamma = platform.gamma;
end

%-----
% This models the platform's patrol path. However, by making the assumption
% that the sensors have extended range to simplify sensor management, this
% modeling is not used here. This will be useful for future work with the
% inclusion of sensor management.
%
% if (next_platform.time >= 0 && next_platform.time < 43200)
%     next_platform.x = platform.x + dt*0.013972407*cos(0.152813367);
%     next_platform.y = platform.y + dt*0.013972407*sin(0.152813367);
%     next_platform.phi = 0.152813367;
%     next_platform.vel = 0.013972407;
%     next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 43200 && next_platform.time < 86400)
%     next_platform.x = platform.x + dt*0.013972407*cos(0.152813367+pi);
%     next_platform.y = platform.y + dt*0.013972407*sin(0.152813367+pi);
%     next_platform.phi = 0.152813367+pi;
%     next_platform.vel = 0.013972407;
%     next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 86400)
%     next_platform.x = platform.x;
%     next_platform.y = platform.y;
%     next_platform.phi = 0;
%     next_platform.vel = 0;

```

```
% next_platform.gamma = 0;  
% end  
%-----
```



```

% This is the platform motion model for MPaA

function next_platform=platform_step_MPAC(platform,time)

globals;

next_platform = make_platform;
next_platform.id=platform.id;
next_platform.time = time;

if (next_platform.time >= 0)
    next_platform.x = 474.1785056;
    next_platform.y = 351.7903000;
    next_platform.phi = platform.phi;
    next_platform.vel = platform.vel;
    next_platform.gamma = platform.gamma;
end

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function creates and assigns sensors to platforms
% It assigns one sensor to one platform

function sensors=assign_sensors(platforms)

globals;

% Define sensors for the 5 fixed platforms

for i=1:NUM_PLATFORMS
    if i==1
        sensors(i)=make_sensor;
        sensors(i).id=i;
        sensors(i).platform=i;
        sensors(i).r_err=SIGMA_RANGE_MPA;
        sensors(i).b_err=SIGMA_BEARING_MPA;
        sensors(i).point=0;      % zero point angle
        sensors(i).beam_view=2*pi;
        sensors(i).max_range=154; % an equivalent of 15nm

    elseif i==2
        sensors(i)=make_sensor;
        sensors(i).id=i;
        sensors(i).platform=i;
        sensors(i).r_err=SIGMA_RANGE_MPA;
        sensors(i).b_err=SIGMA_BEARING_MPA;
        sensors(i).point=0;
        sensors(i).beam_view=2*pi;
        sensors(i).max_range=154;

    elseif i==3
        sensors(i)=make_sensor;
        sensors(i).id=i;
        sensors(i).platform=i;
        sensors(i).r_err=SIGMA_RANGE_MPA;
        sensors(i).b_err=SIGMA_BEARING_MPA;
        sensors(i).point=0;

```

```

sensors(i).beam_view=2*pi;
sensors(i).max_range=154;

elseif i==4
    sensors(i)=make_sensor;
    sensors(i).id=i;
    sensors(i).platform=i;
    sensors(i).r_err=SIGMA_RANGE_MPA;
    sensors(i).b_err=SIGMA_BEARING_MPA;
    sensors(i).point=0;
    sensors(i).beam_view=2*pi;
    sensors(i).max_range=154;

elseif i==5
    sensors(i)=make_sensor;
    sensors(i).id=i;
    sensors(i).platform=i;
    sensors(i).r_err=SIGMA_RANGE_MPA;
    sensors(i).b_err=SIGMA_BEARING_MPA;
    sensors(i).point=0;
    sensors(i).beam_view=2*pi;
    sensors(i).max_range=154;

% Define sensors for the 4 mobile platforms

% Define the CPC sensor
elseif i==6
    sensors(i)=make_sensor;
    sensors(i).id=i;
    sensors(i).platform=i;
    sensors(i).r_err=SIGMA_RANGE_PCG;
    sensors(i).b_err=SIGMA_BEARING_PCG;
    sensors(i).point=0;
    sensors(i).beam_view=2*pi;
    sensors(i).max_range=1000; % range extended to simulate
                              % presence at Jurong Island

% Define the CPC sensor
elseif i==7
    sensors(i)=make_sensor;
    sensors(i).id=i;
    sensors(i).platform=i;
    sensors(i).r_err=SIGMA_RANGE_PCG;
    sensors(i).b_err=SIGMA_BEARING_PCG;
    sensors(i).point=0;

```

```

sensors(i).beam_view=2*pi;
sensors(i).max_range=1000;

% Define the PV sensor
elseif i==8
    sensors(i)=make_sensor;
    sensors(i).id=i;
    sensors(i).platform=i;
    sensors(i).r_err=SIGMA_RANGE_PV;
    sensors(i).b_err=SIGMA_BEARING_PV;
    sensors(i).point=0;
    sensors(i).beam_view=2*pi;
    sensors(i).max_range=1000;

% Define the MPaA sensor
elseif i==9
    sensors(i)=make_sensor;
    sensors(i).id=i;
    sensors(i).platform=i;
    sensors(i).r_err=SIGMA_RANGE_MPAC;
    sensors(i).b_err=SIGMA_BEARING_MPAC;
    sensors(i).point=0;
    sensors(i).beam_view=2*pi;
    sensors(i).max_range=1000;
end
end

```

```
% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% This function initialises a new sensor

function new=make_sensor

new=struct('id',0,'platform',0,'r_err',0,'b_err',0,'point',0,...
          'beam_view',0,'max_range',0);
```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function generates observations from sensors located on
% platforms observing targets for the stipulated time-steps

function observations=generate_observations(sensors,platforms,tracks,times)

[temp,nsensors]=size(sensors);
for s=1:nsensors
    ntimes=1;
    for t=times
        observations(s,ntimes).sensor=s;
        observations(s,ntimes).time=t;
        observations(s,ntimes).report=sensor_report(sensors(s),platforms,tracks,t);
        ntimes=ntimes+1;
    end
end

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function constructs a report for a sensor at a particular time

function report=sensor_report(sensor,platforms,tracks,time)

globals;

report=zeros(NUM_TARGETS,5);

% Find platform location at this time
[px,py]=get_platform_loc(sensor.platform,time,platforms);

for tnum=1:NUM_TARGETS
    % Find target location
    [tx,ty]=get_target_loc(tnum,time,tracks);
    % Compute range and bearing
    dx=tx-px;
    dy=ty-py;
    range=sqrt(dx*dx + dy*dy);
    bearing=atan2(dy,dx);
    % Determine if this is actually visible
    if range < sensor.max_range
        % Add error from sensor model
        report(tnum,1)=tnum;
        report(tnum,2)=range + sensor.r_err*(-1+2*rand);
        report(tnum,3)=bearing + sensor.b_err*(-1+2*rand);
        report(tnum,4)=px;
        report(tnum,5)=py;
    else
        report(tnum,1)=tnum;
        report(tnum,2)=NaN;
        report(tnum,3)=NaN;
        report(tnum,4)=NaN;
        report(tnum,5)=NaN;
    end
end
end

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function retrieves the location of the platform

function [px,py]=get_platform_loc(pnum,time,platforms)

[nsamps,nplats]=size(platforms);
if (pnum> nplats) | (pnum <1)
    error('Incorrect platform number specified in get_platform_loc');
end
if (time<platforms(1,pnum).time)|(time>platforms(nsamps,pnum).time)
    error('Time out of range in get_platform_loc');
end

it=1;
while platforms(it,pnum).time < time
    it=it+1;
end
px=platforms(it,pnum).x;
py=platforms(it,pnum).y;

```



```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function retrieves the location of the target

function [tx,ty]=get_target_loc(tnum,time,tracks)

[nsamps,ntracks]=size(tracks);
if (tnum> ntracks) | (tnum <1)
    error('Incorrect track number specified in get_target_loc');
end
if (time<tracks(1,tnum).time)|(time>tracks(nsamps,tnum).time)
    error('Time out of range in get_target_loc');
end

it=1;
while tracks(it,tnum).time < time
    it=it+1;
end
tx=tracks(it,tnum).x;
ty=tracks(it,tnum).y;

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function displays true target trajectories, platform trajectories,
% target destinations, and sensor observations following a simulation run

function
[true,plat,obs,dest]=post_sim(targets,observations,sensors,platforms,destinations)

globals;

% Set plotting size
[nsamps,nplatforms]=size(platforms);
[nsamps,ndestinations]=size(destinations);
[nsamps,ntargets]=size(targets);
[nsensors,nsamps]=size(observations);

figure(1)
clf;
hold on
data=zeros(2,nsamps);
title('True Target Motions')
xlabel('x-position (m)')
ylabel('y-position (m)')

% Plot true target trajectories
for n=1:ntargets
    for i=1:nsamps
        data(1,i)=targets(i,n).x;
        data(2,i)=targets(i,n).y;
    end
    true=plot(data(1,:),data(2,:),'b-');
end

% Plot true platform trajectories
for n=1:nplatforms
    for i=1:nsamps
        data(1,i)=platforms(i,n).x;
        data(2,i)=platforms(i,n).y;
    end
end

```

```

if n == 1
    plat=plot(data(1,:),data(2:),'ko');
elseif n == 2
    plat=plot(data(1,:),data(2:),'ko');
elseif n == 3
    plat=plot(data(1,:),data(2:),'ko');
elseif n == 4
    plat=plot(data(1,:),data(2:),'ko');
elseif n == 5
    plat=plot(data(1,:),data(2:),'ko');
elseif n == 6
    plat=plot(data(1,:),data(2:),'r*');
elseif n == 7
    plat=plot(data(1,:),data(2:),'r*');
elseif n == 8
    plat=plot(data(1,:),data(2:),'gh');
elseif n == 9
    plat=plot(data(1,:),data(2:),'bo');
end
end

% Plot true destination locations
for n=1:ndestinations
    for i=1:nsamps
        data(1,i)=destinations(i,n).x;
        data(2,i)=destinations(i,n).y;
    end
    dest=plot(data(1,:),data(2:),'rh');
end

% Plot sensor observations
for n=1:nsensors % for all sensors
    for i=1:nsamps % for all time
        report=observations(n,i).report;
        for m=1:ntargets % for all targets
            if report(m,1) > 0 % if they are seen
                [zx,zy,R]=xy_obs(report,m,n); % convert observation to global xy coordinates
                odata(1,m)=zx;
                odata(2,m)=zy;
            end
        end
    end
    if n == 1
        obs=plot(odata(1,:),odata(2:),'k. ');
    elseif n == 2
        obs=plot(odata(1,:),odata(2:),'k. ');
    end
end

```

```

elseif n == 3
    obs=plot(odata(1,:),odata(2:),'k.');
```

```
elseif n == 4
    obs=plot(odata(1,:),odata(2:),'k.');
```

```
elseif n == 5
    obs=plot(odata(1,:),odata(2:),'k.');
```

```
elseif n == 6
    obs=plot(odata(1,:),odata(2:),'r.');
```

```
elseif n == 7
    obs=plot(odata(1,:),odata(2:),'r.');
```

```
elseif n == 8
    obs=plot(odata(1,:),odata(2:),'g.');
```

```
elseif n == 9
    obs=plot(odata(1,:),odata(2:),'b.');
```

```
end
end
end

legend([true,plat,obs,dest],'Target      True      Position','Tracking      Stations','Target
Observations','Intended Target Destination')
hold off

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function extract an xy observation from a range-bearing report

function [zx,zy,R]=xy_obs(rep,n,sensor_id)

globals;

sr=sin(rep(n,3));
cr=cos(rep(n,3));
zx=rep(n,4)+rep(n,2)*cr;
zy=rep(n,5)+rep(n,2)*sr;
ROT=[cr -sr; sr cr];
range2=rep(n,2)*rep(n,2);
if sensor_id==1
    sigma=[SIGMA_RANGE_MPA^2 0;0 range2*SIGMA_BEARING_MPA^2];
elseif sensor_id==2
    sigma=[SIGMA_RANGE_MPA^2 0;0 range2*SIGMA_BEARING_MPA^2];
elseif sensor_id==3
    sigma=[SIGMA_RANGE_MPA^2 0;0 range2*SIGMA_BEARING_MPA^2];
elseif sensor_id==4
    sigma=[SIGMA_RANGE_MPA^2 0;0 range2*SIGMA_BEARING_MPA^2];
elseif sensor_id==5
    sigma=[SIGMA_RANGE_MPA^2 0;0 range2*SIGMA_BEARING_MPA^2];
elseif sensor_id==6
    sigma=[SIGMA_RANGE_PCG^2 0;0 range2*SIGMA_BEARING_PCG^2];
elseif sensor_id==7
    sigma=[SIGMA_RANGE_PCG^2 0;0 range2*SIGMA_BEARING_PCG^2];
elseif sensor_id==8
    sigma=[SIGMA_RANGE_PV^2 0;0 range2*SIGMA_BEARING_PV^2];
elseif sensor_id==9
    sigma=[SIGMA_RANGE_MPAC^2 0;0 range2*SIGMA_BEARING_MPAC^2];
end
R=ROT*sigma*ROT';

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This file executes the filter algorithm

globals;
% Set up filter parameters
filter.Q= [DT^2*SIGMA_XDOT^2 0 0 0;
           0 SIGMA_XDOT^2 0 0;
           0 0 DT^2*SIGMA_YDOT^2 0;
           0 0 0 SIGMA_YDOT^2];
[nsensors,ntime]=size(observations);

% Use sensors
for i=1:nsensors
    filter.use_sensors(i)=1;
end

% Run filter
tracks=tracker(observations,sensors,platforms,filter,times);

% Plot the track trajectory
post_tracks;

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function generates tracks from a sequence of observations and
% assesses its threat to Jurong Island "real-time"

function tracks=tracker(observations,sensors,platforms,filter,times)

globals;

% Number of sensors and number of reports
[nsensors,ntimes]=size(observations);

% Search for first target first observation
target = 1;
for k = 1:nsensors
    for j =1:ntimes;
        if (~isnan(observations(k,j).report(target,2:5)))
            temp(j)=j;
        else
            temp(j)=99999;
        end
    end
    starttime(k) = min(temp);
    if (starttime(k) <= starttime(1))
        startstep = starttime(k);
    end
end

% Initialise the target
for j=1
    tracks(startstep,j)=init_tracks(j,observations,filter,startstep);
end

% Initialise variables
for n=1:ntimes
    kdata(1,n) = 0;
    kdata(2,n) = 0;
    kdata(3,n) = 0;
    kdata(4,n) = 0;
end

```

```

kdata(5,n) = 0;
kdata(6,n) = 0;
kdata(7,n) = 99999;
kdata(8,n) = 0;
kdata(9,n) = 0;
kdata(10,n) = 0;
kdata(11,n) = 0;
leftTSS(n) = NaN;
leftTSSntoJIHigh(n) = NaN;
leftTSSntoJIModerate(n) = NaN;
leftPL(n) = NaN;
leftPLntoJIHigh(n) = NaN;
leftPLntoJIModerate(n) = NaN;
end

% Initialise variables
distapart = 0;
distuncert = 0;
disttoeach = 0;
speedtoeach = 0;
timetoreach = 0;

for i=(startstep+1):ntimes
    startplot=99999;
    buf=sprintf('Step: time=%d',i-1); disp(buf);
    % Do state prediction for the target
    for j=1
        tracks(i,j)=state_pred(tracks(i-1,j),times(i),filter);
    end

    % Do data association, returning an obs*track array of associations
    da_array=data_association(tracks,observations,i);

    % Finally, do update for the target
    for j=1
        tracks(i,j)=state_update(tracks(i,j),observations,filter,da_array,i,j);
    end

    % Check if SAW has entered Port Limits
    if ~isnan(tracks(i,1).state_est(1)) && ~isnan(tracks(i,1).state_est(3))

        % Project the position uncertainty of the target at final destination
        FF = [1 (ntimes-i)*DT 0 0;
              0 1 0 0;
              0 0 1 (ntimes-i)*DT;

```



```

0 0      0 1];

QQ = [((ntimes-i)*DT)^2*SIGMA_XDOT^2  0      0 0;
      0 SIGMA_XDOT^2      0 0;
      0 0 ((ntimes-i)*DT)^2*SIGMA_YDOT^2  0;
      0 0      0 SIGMA_YDOT^2];

projstate_pred = FF*tracks(i,1).state_est;
projuncer_pred = FF*tracks(i,1).uncer_est*FF' + QQ;
projxuncert = sqrt(projuncer_pred(1,1));
projyuncert = sqrt(projuncer_pred(3,3));
projdistuncert = sqrt(projxuncert^2 + projyuncert^2);
% Calculate the position uncertainty of the target at present location
xuncert = sqrt(tracks(i,1).uncer_est(1,1));
yuncert = sqrt(tracks(i,1).uncer_est(3,3));
distuncert = sqrt(xuncert^2 + yuncert^2);
b = circle([tracks(i,1).state_est(1), tracks(i,1).state_est(3)],distuncert);

if min(inpolygon (b.X, b.Y, TSS_X, TSS_Y))== 0 &&...
min(inpolygon (b.X, b.Y, PL_X, PL_Y)) == 0
    buf=sprintf('Warning: Entered Port Limits at Timestep=%d',i-1); disp(buf);

% Check if SAW is turning towards Jurong Island
dittoreach=sqrt(abs(tracks(i,1).state_est(1)-302.1144444)^2+...
                abs(tracks(i,1).state_est(3)-347.7716667)^2);
speedtoreach=sqrt(tracks(i,1).state_est(2)^2+...
                tracks(i,1).state_est(4)^2);
timetoreach=dittoreach/speedtoreach;
% Calculate the time target left Port Limits
leftPL(i) = MAX_TIME-timetoreach;

% Calculate the distance between the projected target end point
% and Jurong Island
distapart = sqrt((projstate_pred(1) - 302.1144444)^2 +...
                (projstate_pred(3) - 347.7716667)^2);

% Threat level assessment
% Send warning if target is within 0.5nm to Jurong Island
if distapart < 5 && MAX_TIME - (i-1)*DT >= 0
    buf=sprintf('Warning: Heading Towards Jurong Island. Time to Impact = %.2f
sec',timetoreach); disp(buf);
    if projdistuncert > distapart
        buf=sprintf('Warning: High Confidence of Impact');disp(buf);
        startplot = i-1;
        leftPLntoJIHigh(i) = MAX_TIME-timetoreach;

```

```

else
    buf=sprintf('Warning: Moderate Confidence of Impact');disp(buf);
    startplot = i-1;
    leftPLntoJIModerate(i) = MAX_TIME-timetoreach;
end
end

% Check if SAW has left Traffic Separation Scheme
elseif min(inpolygon (b.X, b.Y, TSS_X, TSS_Y))== 0 &&...
    min(inpolygon (b.X, b.Y, PL_X, PL_Y)) == 1
    buf=sprintf('Warning: Left Traffic Separation Scheme at Timestep=%d',i-1);
disp(buf);

% Check if SAW is turning towards Jurong Island
dittoreach=sqrt(abs(tracks(i,1).state_est(1)-302.1144444)^2+...
    abs(tracks(i,1).state_est(3)-347.7716667)^2);
speedtoreach = sqrt(tracks(i,1).state_est(2)^2+...
    tracks(i,1).state_est(4)^2);
timetoreach = dittoreach/speedtoreach;
% Calculate the time target left TSS
leftTSS(i) = MAX_TIME-timetoreach;

% Calculate the distance between the projected target
% end pointand Jurong Island
distapart = sqrt((projstate_pred(1) - 302.1144444)^2+...
    (projstate_pred(3) - 347.7716667)^2);

% Threat level assessment
% Send warning if target is within 0.5nm to Jurong Island
if distapart < 5 && MAX_TIME - (i-1)*DT >= 0
    buf=sprintf('Warning: Heading Towards Jurong Island. Time to Impact = %.2f
sec',timetoreach); disp(buf);
    if projdistuncert > distapart
        buf=sprintf('Warning: High Confidence of Impact');disp(buf);
        startplot=i-1;
        leftTSSntoJIHigh(i) = MAX_TIME-timetoreach;
    else
        buf=sprintf('Warning: Moderate Confidence of Impact');disp(buf);
        startplot=i-1;
        leftTSSntoJIModerate(i) = MAX_TIME-timetoreach;
    end
end
end
end

kdata(1,i) = tracks(i,1).time;

```

```

kdata(2,i) = projdistuncert*180;
kdata(3,i) = distapart*180;
kdata(4,i) = disttoeach*180;
kdata(5,i) = speedtoeach*180;
kdata(6,i) = timetoreach;
kdata(7,i) = startplot;
kdata(8,i) = projuncer_pred(1,1);
kdata(9,i) = projuncer_pred(3,3);
kdata(10,i) = projstate_pred(1);
kdata(11,i) = projstate_pred(3);

end

% Future work for multi-targets
% Search for first observation for other targets
if NUM_TARGETS ~=1
    for target=2:NUM_TARGETS
        for k = 1:nsensors
            for j =1:ntimes;
                if (~isnan(observations(k,j).report(target,2:5)))
                    temp(j)=j;
                else
                    temp(j)=99999;
                end
            end
            starttime(k) = min(temp);
            if (starttime(k) <= starttime(1))
                startstep = starttime(k);
            end
        end
    end
end

% Initialisation
for j=2:NUM_TARGETS
    tracks(startstep,j)=init_tracks(j,observations,filter,startstep);
end

% Track
for i=(startstep+1):ntimes
    % Prediction for each target
    for j=2:NUM_TARGETS
        tracks(i,j)=state_pred(tracks(i-1,j),times(i),filter);
    end
end

```

```

% Data association, returning an obs*track array of associations
da_array=data_association(tracks,observations,i);

% Update for each target
for j=2:NUM_TARGETS
    tracks(i,j)=state_update(tracks(i,j),observations,filter,da_array,i,j);
end
end
end

figure(20);
clf;
plot(kdata(1,:),(kdata(2,:),'b',kdata(1,:),kdata(3,:),'g')
legend('Distance Uncertainty','Distance Apart');
buf=sprintf('Estimated End Point');
title(buf)
%set(gca,'XDir','reverse')
xlim([min(kdata(7,1000:MAX_TIME/DT))*DT MAX_TIME-1*DT])
axis 'auto y'
xlabel('Simulated Time (s)')
ylabel('Distance (m)')

figure(21);
clf;
plot(kdata(1,:),kdata(4,:),'b',kdata(1,:),kdata(5,:),'g',kdata(1,:),kdata(6,:),'r')
legend('Distance to Impact','Speed to Impact', 'Time to Impact');
buf=sprintf('Projected Distance, Speed and Time to Impact');
title(buf)
xlim([min(kdata(7,1000:MAX_TIME/DT))*DT MAX_TIME-1*DT])
axis 'auto y'
xlabel('Simulated Time (s)')

% Record the times the target crossed TSS, PL, & corresponding threat level
firstleftTSS = min(leftTSS(1000:MAX_TIME/DT));
firstleftTSSntoJIHigh = min(leftTSSntoJIHigh(1000:MAX_TIME/DT));
firstleftTSSntoJIModerate = min(leftTSSntoJIModerate(1000:MAX_TIME/DT));
firstleftPL = min(leftPL(1000:MAX_TIME/DT));
firstleftPLntoJIHigh = min(leftPLntoJIHigh(1000:MAX_TIME/DT));
firstleftPLntoJIModerate = min(leftPLntoJIModerate(1000:MAX_TIME/DT));

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function initialises a set of tracks from the first few observations

function track=init_tracks(ntarget,observations,filter,startstep)

globals;

% Find the number of sensors
[nsensors,ntimes]=size(observations);

% Now extract all sensor data in the use list associated with ntarget
if ntarget ==1
    nz=0;
    for i=1:nsensors
        if filter.use_sensors(i) == 1
            nz=nz+1;
            %check for NaN first
            if (~isnan(observations(i,startstep).report(ntarget,2:5)))
                report=observations(i,startstep).report;
                [z(1,nz),z(2,nz),R]=xy_obs(report,ntarget,i);
            end
        end
    end
end

% Specific for the x-y problem, now initialise the track
state_pred=zeros(XSIZE,1);
state_pred(1)=738.9629; %initial conditions
state_pred(2)=0.0303;
state_pred(3)=443.8223;
state_pred(4)=0.0303;
state_est=state_pred;

uncer_pred=[0.30864  0      0      0;
            0      0.000408  0      0;
            0      0      0.30864  0;
            0      0      0      0.000408]; %10 knots
uncer_est=uncer_pred;
ninnov=nz;

```

```

    innov=zeros(ninnov,1);
    innov_var=zeros(ninnov,ninnov);

    % put together data structure
    track=make_track(ntarget,1,startstep,state_pred,state_est,...
        uncer_pred,uncer_est,innov,innov_var);
end

if ntarget ~= 1
    nz=0;
    for i=1:nsensors
        if filter.use_sensors(i) == 1
            nz=nz+1;
            %check for NaN first
            if (~isnan(observations(i,startstep).report(ntarget,2:5)))
                report=observations(i,startstep).report;
                [z(1,nz),z(2,nz),R]=xy_obs(report,ntarget,i);
            end
        end
    end
    state_pred=zeros(XSIZE,1);
    state_pred(1)=max(z(1,:));
    state_pred(2)=MIN_TARGET_VEL/sqrt(2);
    state_pred(3)=max(z(2,:));
    state_pred(4)=MIN_TARGET_VEL/sqrt(2);
    state_est=state_pred;

    uncer_pred=10*filter.Q;
    uncer_est=uncer_pred;

    ninnov=nz;
    innov=zeros(ninnov,1);
    innov_var=zeros(ninnov,ninnov);

    % put together data structure
    track=make_track(ntarget,1,startstep,state_pred,state_est,...
        uncer_pred,uncer_est,innov,innov_var);
end

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% This function initialises a new track

function track=make_track(id,type,time,state_pred,state_est,...
    uncer_pred,uncer_est,innov,innov_var)

track=struct('id',id,'type',type,'time',time,'state_pred',state_pred,...
    'state_est',state_est,'uncer_pred',uncer_pred,...
    'uncer_est',uncer_est,'innov',innov,'innov_var',innov_var);

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function performs prediction for a CVM

function track_pred=state_pred(track,time,filter)

globals;

dt=time-track.time;
if dt < 0
    error('Negative prediction step in state_pred');
end

% Compute transition matrix
F=[1 dt 0 0;
    0 1 0 0;
    0 0 1 dt;
    0 0 0 1];

% Do prediction
track_pred.time=time;
state_pred=F*track.state_est;
uncer_pred=F*track.uncer_est*F' + filter.Q;

% Construct the new track
ninnoy=sum(filter.use_sensors);
innoy=zeros(ninnoy,1);
innoy_var=zeros(ninnoy,ninnoy);
track_pred=make_track(track.id,1,time,state_pred,state_pred,...
    uncer_pred,uncer_pred,innoy,innoy_var);

```



```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function sets up the data association array

function da=data_association(tracks,observations,time)

% This is an array of size ntargets to nsensors
% For each sensor i, the entry states which sensor report
% is associated with a target j
[nsensors,ntimes]=size(observations);
[ntimes,ntargets]=size(tracks);
da=zeros(ntargets,nsensors);

% For each sensor
for i=1:nsensors
    % Associate a track with a report
    for j=1:ntargets
        da(j,i)=j;
    end
end

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function performs kalman filter update

function track=state_update(track,observations,filter,da_array,ntime,ntarget)

globals;

[nsensors,temp]=size(observations);

% Construct observation array
nz=0;
for i=1:nsensors
    if filter.use_sensors(i) == 1
        % Check for valid observations first
        if (~isnan(observations(i,ntime).report(ntarget,2:5)))
            if (observations(i,ntime).report(ntarget,2:5) ~= 99999)
                % Extract observation
                rep=observations(i,ntime).report;
                na=da_array(ntarget,i);
                [z(1+nz),z(2+nz),Ri]=xy_obs(rep,na,i); % Ri is unused
                if i==1
                    R(1+nz:2+nz,1+nz:2+nz)=Rfilter_MPA;
                elseif i==2
                    R(1+nz:2+nz,1+nz:2+nz)=Rfilter_MPA;
                elseif i==3
                    R(1+nz:2+nz,1+nz:2+nz)=Rfilter_MPA;
                elseif i==4
                    R(1+nz:2+nz,1+nz:2+nz)=Rfilter_MPA;
                elseif i==5
                    R(1+nz:2+nz,1+nz:2+nz)=Rfilter_MPA;
                elseif i==6
                    R(1+nz:2+nz,1+nz:2+nz)=Rfilter_PCG;
                elseif i==7
                    R(1+nz:2+nz,1+nz:2+nz)=Rfilter_PCG;
                elseif i==8
                    R(1+nz:2+nz,1+nz:2+nz)=Rfilter_PV;
                elseif i==9
                    R(1+nz:2+nz,1+nz:2+nz)=Rfilter_MPAC;
            end
        end
    end
end

```

```

    end
    H(1+nz,:)= [1 0 0 0];
    H(2+nz,:)= [0 0 1 0];
    nz=nz+2;

    elseif (observations(i,ntime).report(ntarget,2:5) == 99999)
        z(1+nz)= track.state_pred(1);
        z(2+nz)= track.state_pred(3);
        H(1+nz,:)= [1 0 0 0];
        H(2+nz,:)= [0 0 1 0];
        nz=nz+2;
    end
end
end
end

% Do update
innov=z'-H*track.state_pred;
if (innov ~= 0)
    S=H*track.uncer_pred*H'+ R;
    W=track.uncer_pred*H'*inv(S);
    track.innov=innov;
    track.innov_var=S;
    track.state_est=track.state_pred + W*innov;
    track.uncer_est=track.uncer_pred - W*S*W';
else % Propagate prediction if there is no sensor observation
    track.state_est=track.state_pred;
    track.uncer_est=track.uncer_pred;
end

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This file displays the fused target track following a simulation run

globals;

% Set plotting size
[nsamps,ntargets]=size(tracks);
figure(1)
xlim([150 750])
ylim([280 440])
hold on
data=zeros(2,nsamps);

j =1;
while (isempty(tracks(j,1).state_pred(:)))
    j = j+1;
end
trackstarttime = j;

for n=1:ntargets
    for i=trackstarttime:nsamps
        data(1,i)=tracks(i,n).state_est(1);
        data(2,i)=tracks(i,n).state_est(3);
    end
    estp=plot(data(1,trackstarttime:nsamps),data(2,trackstarttime:nsamps),'g-');
end

legend([estp],'Estimated Track Position')
hold off

plot_errors(tracks, ntargets, 2, trackstarttime, targets)

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function produces various plots with regards to the SAW parameters

function plot_errors(tracks,ntarget,nfigure,trackstarttime,targets)

globals;

% Initialise
[nsamps,ntargets]=size(tracks);
zdata=zeros(10,nsamps);
idata=zeros(10,nsamps);
pdata=zeros(10,nsamps); %predicted data
edata=zeros(10,nsamps); %estimated data
tdata=zeros(10,nsamps); %true data
esttimesigma=zeros(1,nsamps);
xmin = 4500;
xmax = MAX_TIME;

warning off;
% Calculating and storing the data for each time-step
for i=trackstarttime:nsamps

    zdata(1,i)=tracks(i,ntarget).time;
    zdata(2,i)=(tracks(i,ntarget).state_est(1)-targets(i,ntarget).x)*180;
    zdata(3,i)=(tracks(i,ntarget).state_est(3)-targets(i,ntarget).y)*180;
    zsigma=real(sqrt(tracks(i,ntarget).uncer_est));
    zdata(4,i)=zsigma(1,1)*180;
    zdata(5,i)=zsigma(3,3)*180;
    zdata(6,i)=(real(sqrt...
        (tracks(i,ntarget).state_est(2)...
        *tracks(i,ntarget).state_est(2)...
        +tracks(i,ntarget).state_est(4)...
        *tracks(i,ntarget).state_est(4)))...
        - targets(i,ntarget).vel)*180;
    zdata(7,i)=atan2(tracks(i,ntarget).state_est(4),...
        tracks(i,ntarget).state_est(2))-targets(i,ntarget).phi;
    if abs(zdata(7,i)) > pi
        zdata(7,i) = 2*pi-abs(zdata(7,i));

```

```

end
zdata(8,i)=sqrt(zsigma(1,1)*zsigma(1,1)+zsigma(3,3)*zsigma(3,3))*180;
zdata(9,i)=sqrt(zsigma(2,2)*zsigma(2,2)+zsigma(4,4)*zsigma(4,4))*180;
zdata(10,i)=real(...
    sqrt(...
        ((tracks(i,ntarget).state_est(2)...
        /(tracks(i,ntarget).state_est(2)^2 + ...
        tracks(i,ntarget).state_est(4)^2))^2)...
        *(zsigma(4,4)^2)...
    + ...
        ((tracks(i,ntarget).state_est(4)...
        /(tracks(i,ntarget).state_est(2)^2 + ...
        tracks(i,ntarget).state_est(4)^2))^2)...
        *(zsigma(2,2)^2)...
    )...
);

tdata(1,i)=tracks(i,ntarget).time;
tdata(2,i)=targets(i,ntarget).x*180;
tdata(3,i)=targets(i,ntarget).y*180;
tdata(6,i)=targets(i,ntarget).vel*180;
tdata(7,i)=targets(i,ntarget).phi;

pdata(1,i)=tracks(i,ntarget).time;
pdata(2,i)=tracks(i,ntarget).state_pred(1)*180;
pdata(3,i)=tracks(i,ntarget).state_pred(3)*180;
psigma=real(sqrt(tracks(i,ntarget).uncer_pred));
pdata(4,i)=psigma(1,1)*180;
pdata(5,i)=psigma(3,3)*180;
pdata(6,i)=(real(sqrt...
    (tracks(i,ntarget).state_pred(2)...
    *tracks(i,ntarget).state_pred(2)...
    +tracks(i,ntarget).state_pred(4)...
    *tracks(i,ntarget).state_pred(4)))...
    )*180;
pdata(7,i)=atan2(tracks(i,ntarget).state_pred(4),...
    tracks(i,ntarget).state_pred(2));
if abs(pdata(7,i)) > pi
    pdata(7,i) = 2*pi-abs(pdata(7,i));
end
pdata(8,i)=sqrt(psigma(1,1)*psigma(1,1)+psigma(3,3)*psigma(3,3))*180;
pdata(9,i)=sqrt(psigma(2,2)*psigma(2,2)+psigma(4,4)*psigma(4,4))*180;
pdata(10,i)=real(...
    sqrt(...
        ((tracks(i,ntarget).state_pred(2)...

```

```

/(tracks(i,ntarget).state_pred(2)^2 + ...
tracks(i,ntarget).state_pred(4)^2))^2)...
*(psigma(4,4)^2)...
+ ...
(((tracks(i,ntarget).state_pred(4)...
/(tracks(i,ntarget).state_pred(2)^2 + ...
tracks(i,ntarget).state_pred(4)^2))^2)...
*(psigma(2,2)^2)...
)...
);

edata(1,i)=tracks(i,ntarget).time;
edata(2,i)=tracks(i,ntarget).state_est(1)*180;
edata(3,i)=tracks(i,ntarget).state_est(3)*180;
esigma=real(sqrt(tracks(i,ntarget).uncer_est));
edata(4,i)=esigma(1,1)*180;
edata(5,i)=esigma(3,3)*180;
edata(6,i)=(real(sqrt(...
(tracks(i,ntarget).state_est(2)...
*tracks(i,ntarget).state_est(2)...
+tracks(i,ntarget).state_est(4)...
*tracks(i,ntarget).state_est(4)))...
)*180;
edata(7,i)=atan2(tracks(i,ntarget).state_est(4),...
tracks(i,ntarget).state_est(2));
if abs(edata(7,i)) > pi
    edata(7,i) = 2*pi-abs(edata(7,i));
end
edata(8,i)=sqrt(esigma(1,1)*esigma(1,1)+esigma(3,3)*esigma(3,3))*180;
edata(9,i)=sqrt(esigma(2,2)*esigma(2,2)+esigma(4,4)*esigma(4,4))*180;
edata(10,i)=real(...
sqrt(...
(((tracks(i,ntarget).state_est(2)...
/(tracks(i,ntarget).state_est(2)^2 + ...
tracks(i,ntarget).state_est(4)^2))^2)...
*(esigma(4,4)^2)...
+ ...
(((tracks(i,ntarget).state_est(4)...
/(tracks(i,ntarget).state_est(2)^2 + ...
tracks(i,ntarget).state_est(4)^2))^2)...
*(esigma(2,2)^2)...
)...
);

esttimesigma(1,i)=sqrt(edata(8,i)^2+...

```

```

        (kdata(6,i)^2)*(edata(9,i)^2))/...
        edata(6,i);
end

% merging the heading standard deviations for plotting
p=trackstarttime;q=trackstarttime;i=trackstarttime;
for j=1:(nsamps-trackstarttime+1)*2
    if j/2-round(j/2)==0
        combtimedata(1,j)= tracks(i,ntarget).time;
        combddata(8,j)=edata(8,p);
        combddata(9,j)=edata(9,p);
        combddata(10,j)=edata(10,p);
        p=p+1;
        i=i+1;
    else
        combtimedata(1,j)= tracks(i,ntarget).time;
        combddata(8,j)=pdata(8,q);
        combddata(9,j)=pdata(9,q);
        combddata(10,j)=pdata(10,q);
        q=q+1;
    end
end

% Plot
figure(nfigure);
clf;
plot(zdata(1,:),abs(zdata(2,:)),'b',zdata(1,:),zdata(4,:),'r')
legend('Estimated Error','Standard Deviation');
buf=sprintf('Error between X-component Estimate and Truth');
title(buf)
xlim([xmin xmax])
axis 'auto y'
xlabel('Simulated Time (s)')
ylabel('X error (m)')

figure(nfigure+1);
clf;
plot(zdata(1,:),abs(zdata(3,:)),'b',zdata(1,:),zdata(5,:),'r')
legend('Estimated Error','Standard Deviation');
buf=sprintf('Error between Y-component Estimate and Truth');
title(buf)
xlim([xmin xmax])
axis 'auto y'
xlabel('Simulated Time (s)')
ylabel('Y error (m)')

```



```

figure(nfigure+2);
clf;
plot(zdata(1,:),abs(zdata(6,:)), 'b', zdata(1,:), zdata(9,:), 'r')
legend('Estimated Error', 'Standard Deviation');
buf=sprintf('Error between Velocity Estimate and Truth');
title(buf)
xlim([xmin xmax])
axis 'auto y'
xlabel('Simulated Time (s)')
ylabel('Velocity error (m/s)')

figure(nfigure+3);
clf;
plot(zdata(1,:),abs(zdata(7,:)), 'b', zdata(1,:), zdata(10,:), 'r')
legend('Estimated Error', 'Standard Deviation');
buf=sprintf('Error between Heading Estimate and Truth');
title(buf)
xlim([xmin xmax])
axis 'auto y'
xlabel('Simulated Time (s)')
ylabel('Heading error (rads)')

figure(nfigure+4);
clf;
plot(pdata(1,:),pdata(2,:), 'r', edata(1,:), edata(2,:), 'g', tdata(1,:), tdata(2,:), 'b')
legend('Predicted X', 'Estimated X', 'True X');
buf=sprintf('X position');
title(buf)
xlim([xmin xmax])
axis 'auto y'
xlabel('Simulated Time (s)')
ylabel('X Position (m)')

figure(nfigure+5);
clf;
plot(pdata(1,:),pdata(3,:), 'r', edata(1,:), edata(3,:), 'g', tdata(1,:), tdata(3,:), 'b')
legend('Predicted Y', 'Estimated Y', 'True Y');
buf=sprintf('Y position');
title(buf)
xlim([xmin xmax])
axis 'auto y'
xlabel('Simulated Time (s)')
ylabel('Y Position (m)')

```

```

figure(nfigure+6);
clf;
plot(pdata(1,:),pdata(6,:), 'r', edata(1,:), edata(6,:), 'g', tdata(1,:), tdata(6,:), 'b')
legend('Predicted Velocity', 'Estimated Velocity', 'True Velocity');
buf=sprintf('Velocity');
title(buf)
xlim([xmin xmax])
axis 'auto y'
xlabel('Simulated Time (s)')
ylabel('Velocity (m/s)')

figure(nfigure+7);
clf;
plot(pdata(1,:),pdata(7,:), 'r', edata(1,:), edata(7,:), 'g', tdata(1,:), tdata(7,:), 'b')
legend('Predicted Heading', 'Estimated Heading', 'True Heading');
buf=sprintf('Heading');
title(buf)
xlim([xmin xmax])
axis 'auto y'
xlabel('Simulated Time (s)')
ylabel('Heading (rad)')

figure(nfigure+8);
clf;
plot(pdata(1,:),pdata(8,:), 'r+', edata(1,:), edata(8,:), 'g*', combtimedata(1,:), combdata(8,:), 'b')
legend('Predicted', 'Estimated');
buf=sprintf('Radius Standard Deviations');
title(buf)
xlim([xmin xmax])
axis 'auto y'
xlabel('Simulated Time (s)')
ylabel('Radius (m)')

figure(nfigure+9);
clf;
plot(pdata(1,:),pdata(9,:), 'r+', edata(1,:), edata(9,:), 'g*', combtimedata(1,:), combdata(9,:), 'b')
legend('Predicted', 'Estimated');
buf=sprintf('Velocity Standard Deviations');
title(buf)
xlim([xmin xmax])
axis 'auto y'
xlabel('Simulated Time (s)')
ylabel('Velocity (m/s)')

figure(nfigure+10);

```

```

clf;
plot(pdata(1,:),pdata(10,:), 'r+', edata(1,:), edata(10,:), 'g*', combtimedata(1,:), combdata(10,:), 'b')
legend('Predicted', 'Estimated');
buf=sprintf('Heading Standard Deviations');
title(buf)
xlim([xmin xmax])
axis 'auto y'
xlabel('Simulated Time (s)')
ylabel('Heading (rad)')

figure(nfigure+11);
clf;
plot(edata(1,:),abs((MAX_TIME-edata(1,:))-kdata(6,:)), 'b', edata(1,:), esttimesigma(1,:), 'r')
legend('Estimated Error', 'Standard Deviation');
buf=sprintf('Error between Time Estimate and Truth');
title(buf)
xlim([min(kdata(7,1000:MAX_TIME/10))*DT xmax-1*DT])
axis 'auto y'
xlabel('Simulated Time (s)')
ylabel('Time (s)')

warning on;

```

B. BLOCK 2. CENTRALIZED DATA FUSION ARCHITECTURE USING INFORMATION FILTER

% This file executes the Monte-Carlo simulation

% Number of Monte-Carlo simulation runs

totalruns = 50;

% Initialise variables

sumfirstleftTSS(1:totalruns) = NaN;

sumfirstleftTSSntoJIHigh(1:totalruns) = NaN;

sumfirstleftTSSntoJIModerate(1:totalruns) = NaN;

sumfirstleftPL(1:totalruns) = NaN;

sumfirstleftPLntoJIHigh(1:totalruns) = NaN;

sumfirstleftPLntoJIModerate(1:totalruns) = NaN;

% Execute each run by generating new observations or using previous ones

% Produce the fused track for each run

% Then pool the tracks together for analysis

for run = 1:totalruns

 buf=sprintf('Run: =%d',run); disp(buf);

 example_sim;

 observations = datastore9(run).observations; % using previous observations

 run_ifilt;

 sumfirstleftTSS(run) = firstleftTSS;

 sumfirstleftTSSntoJIHigh(run) = firstleftTSSntoJIHigh;

 sumfirstleftTSSntoJIModerate(run) = firstleftTSSntoJIModerate;

 sumfirstleftPL(run) = firstleftPL;

 sumfirstleftPLntoJIHigh(run) = firstleftPLntoJIHigh;

 sumfirstleftPLntoJIModerate(run) = firstleftPLntoJIModerate;

 sumkdata(run,,:) = kdata;

 sumesttimesigma(run,1,:) = esttimesigma;

 lowest(run) = min(sumkdata(run,7,1000:MAX_TIME/10));

end

% Calculate Probability of Impact

Cxtemp = squeeze(sumkdata(:,8,:));

Cytemp = squeeze(sumkdata(:,9,:));

Xtemp = squeeze(sumkdata(:,10,:));

Ytemp = squeeze(sumkdata(:,11,:));

probability = zeros(1,ntime);

warning off;

for l = 1:ntime

```

Cx = diag(Cxtemp(:,l))*(180^2);
Cy = diag(Cytemp(:,l))*(180^2);
X = (Xtemp(:,l)- 302.1144444)*180;
Y = (Ytemp(:,l)- 347.7716667)*180;
W = ones(totalruns,1);
sigma_xbar_tgo = inv((W')*inv(Cx)*W);
sigma_ybar_tgo = inv((W')*inv(Cy)*W);
sigma = sqrt(sigma_xbar_tgo);
xbar_tgo = sigma_xbar_tgo*(((W')*inv(Cx)*X)); % Calculate mean impact pt
ybar_tgo = sigma_ybar_tgo*(((W')*inv(Cy)*Y)); % Calculate mean impact pt
r_not = sqrt(xbar_tgo^2 + ybar_tgo^2);
R = 300;

g = zeros(1,100);
h = zeros(1,100);
probttemp = 0;

recur = 1;
g_not = exp(-(r_not^2)/(2*sigma^2)); % g_not
h_not = 1 - exp(-(R^2)/(2*sigma^2)); % h_not
probttemp_not = g_not*h_not;

g(1) = (1/1)*((r_not^2)/(2*sigma^2))*g_not;
h(1) = -(1/factorial(1))*(((R^2)/(2*sigma^2))^(1))*exp(-(R^2)/(2*sigma^2)) + h_not;
probttemp = probttemp_not + g(1)*h(1);

while (recur <= 100)
    g(recur+1) = (1/(recur+1))*((r_not^2)/(2*sigma^2))*g(recur);
    h(recur+1) = (-1/factorial(recur+1))*(((R^2)/(2*sigma^2))^(recur+1))*exp(-(R^2)/(2*sigma^2)) + h(recur);
    probttemp = probttemp + g(recur+1)*h(recur+1);
    recur = recur + 1;
end

probability(1,l) = probttemp;
end

% Initialise variables
count1 = 0;
count2 = 0;
count3 = 0;
count4 = 0;
count5 = 0;
count6 = 0;
SsumfirstleftTSS = 0;

```

```

SsumfirstleftTSSntoJIHigh = 0;
SsumfirstleftTSSntoJIModerate = 0;
SsumfirstleftPL = 0;
SsumfirstleftPLntoJIHigh = 0;
SsumfirstleftPLntoJIModerate = 0;

% Take the average of the pooled tracks
for m = 1:totalruns
    if ~isnan(sumfirstleftTSS(m))
        count1 = count1+1;
        SsumfirstleftTSS=SsumfirstleftTSS+sumfirstleftTSS(m);
    end

    if ~isnan(sumfirstleftTSSntoJIHigh(m))
        count2 = count2+1;

SsumfirstleftTSSntoJIHigh=SsumfirstleftTSSntoJIHigh+sumfirstleftTSSntoJIHigh(m);
    end

    if ~isnan(sumfirstleftTSSntoJIModerate(m))
        count3 = count3+1;

SsumfirstleftTSSntoJIModerate=SsumfirstleftTSSntoJIModerate+sumfirstleftTSSntoJIModerate(m);
    end

    if ~isnan(sumfirstleftPL(m))
        count4 = count4+1;
        SsumfirstleftPL=SsumfirstleftPL+sumfirstleftPL(m);
    end

    if ~isnan(sumfirstleftPLntoJIHigh(m))
        count5 = count5+1;
        SsumfirstleftPLntoJIHigh=SsumfirstleftPLntoJIHigh+sumfirstleftPLntoJIHigh(m);
    end

    if ~isnan(sumfirstleftPLntoJIModerate(m))
        count6 = count6+1;

SsumfirstleftPLntoJIModerate=SsumfirstleftPLntoJIModerate+sumfirstleftPLntoJIModerate(m);
    end
end

meanfirstleftTSS = SsumfirstleftTSS/count1;

```

```

meanfirstleftTSSntoJIHigh = SsumfirstleftTSSntoJIHigh/count2;
meanfirstleftTSSntoJIModerate = SsumfirstleftTSSntoJIModerate/count3;
meanfirstleftPL = SsumfirstleftPL/count4;
meanfirstleftPLntoJIHigh = SsumfirstleftPLntoJIHigh/count5;
meanfirstleftPLntoJIModerate = SsumfirstleftPLntoJIModerate/count6;
meantruetimedata = mean(sumkdata(:,1,:));
meandistuncert = mean(sumkdata(:,2,:));
meandistapart = mean(sumkdata(:,3,:));
meanesttimedata = mean(sumkdata(:,6,:));

% Calculate the sample variance of the estimated time to impact
for p = 1:ntime
    sumtimediff=0;
    timediff=0;
    for q = 1:totalruns
        timediff = (sumkdata(q,6,p)-meanesttimedata(p))^2;
        sumtimediff=sumtimediff+timediff;
    end
    meansampletimesigma(p) = sqrt(sumtimediff /(totalruns-1));
end

figure(30);
clf;
plot(meantruetimedata(:)',meandistuncert(:)','b',meantruetimedata(:)',meandistapart(:)','g')
legend('Distance Uncertainty','Distance Apart');
buf=sprintf('Estimated End Point');
title(buf)
xlim([min(lowest)*DT MAX_TIME-1*DT])
ylim([0 1000])
xlabel('Simulated Time (s)')
ylabel('Distance (m)')

figure(31);
clf;
plot(meantruetimedata(:)',abs((MAX_TIME-meantruetimedata(:)')-
meanesttimedata(:)'),'b',meantruetimedata(:)',meansampletimesigma,'r' )
legend('Estimated Error','Standard Deviation');
buf=sprintf('Error between Time Estimate and Truth');
title(buf)
xlim([min(lowest)*DT MAX_TIME-1*DT])
ylim([0 100])
xlabel('Simulated Time (s)')
ylabel('Time (s)')

figure(32);

```

```

clf;
plot(meantruetimedata(:)',probability(:),'k')
legend('Probability of Impact');
buf=sprintf('Probability of Impact');
title(buf)
xlim([min(lowest)*DT MAX_TIME-1*DT])
ylim([0 1])
xlabel('Simulated Time (s)')
ylabel('Probability of Impact')

```



```
% This file executes the Monte-Carlo simulation and incorporates dataloss
% (i.e. observations loss) due to communication bandwidth limitations
```

```
% Number of Monte-Carlo simulation runs
```

```
totalruns = 5;
```

```
sumfirstleftTSS(1:totalruns) = NaN;
sumfirstleftTSSntoJIHigh(1:totalruns) = NaN;
sumfirstleftTSSntoJIModerate(1:totalruns) = NaN;
sumfirstleftPL(1:totalruns) = NaN;
sumfirstleftPLntoJIHigh(1:totalruns) = NaN;
sumfirstleftPLntoJIModerate(1:totalruns) = NaN;
```

```
for run = 1:totalruns
```

```
    buf=sprintf('Run: =%d',run); disp(buf);
```

```
    example_sim;
```

```
    %observations = datastoreloss(run).observations; % using previous observations
```

```
    % Since previous observations (with loss) is used, there is no need to
    % re-inject data loss. If new observations (with no loss) are made, the
    % following codes are necessary to inject data loss.
```

```
    %-----
    %   % simulate data loss due to bandwidth limitations
    %   for i = 1:10
    %       j = randi(MAX_TIME/DT-1060-3)+1060;
    %
    %       datalossduration = randi(3);
    %       for k = 1:datalossduration
    %           for n = 1:NUM_PLATFORMS
    %               observations(n,j+k).report(2:5) = 99999;
    %           end
    %       end
    %   end
    %   end
    %   datastoreloss(run).observations=observations;
    %-----
```

```
run_ifilt;
```

```
sumfirstleftTSS(run) = firstleftTSS;
sumfirstleftTSSntoJIHigh(run) = firstleftTSSntoJIHigh;
sumfirstleftTSSntoJIModerate(run) = firstleftTSSntoJIModerate;
sumfirstleftPL(run) = firstleftPL;
sumfirstleftPLntoJIHigh(run) = firstleftPLntoJIHigh;
sumfirstleftPLntoJIModerate(run) = firstleftPLntoJIModerate;
```

```

sumkdata(run,,:) = kdata;
sumesttimesigma(run,1,:) = esttimesigma;
lowest(run) = min(sumkdata(run,7,1000:MAX_TIME/10));
end

% Calculate Probability of Impact
Cxtemp = squeeze(sumkdata(:,8,:));
Cytemp = squeeze(sumkdata(:,9,:));
Xtemp = squeeze(sumkdata(:,10,:));
Ytemp = squeeze(sumkdata(:,11,:));
probability = zeros(1,ntime);

warning off;
for l = 1:ntime

    Cx = diag(Cxtemp(:,l))*(180^2);
    Cy = diag(Cytemp(:,l))*(180^2);
    X = (Xtemp(:,l)- 302.1144444)*180;
    Y = (Ytemp(:,l)- 347.7716667)*180;
    W = ones(totalruns,1);
    sigma_xbar_tgo = inv((W')*inv(Cx)*W);
    sigma_ybar_tgo = inv((W')*inv(Cy)*W);
    sigma = sqrt(sigma_xbar_tgo);
    xbar_tgo = sigma_xbar_tgo*(((W')*inv(Cx)*X)); % Calculate mean impact pt
    ybar_tgo = sigma_ybar_tgo*(((W')*inv(Cy)*Y)); % Calculate mean impact pt
    r_not = sqrt(xbar_tgo^2 + ybar_tgo^2);
    R = 300;

    g = zeros(1,100);
    h = zeros(1,100);
    probtemp = 0;

    recur = 1;
    g_not = exp(-(r_not^2)/(2*sigma^2)); % g_not
    h_not = 1 - exp(-(R^2)/(2*sigma^2)); % h_not
    probtemp_not = g_not*h_not;

    g(1) = (1/1)*((r_not^2)/(2*sigma^2))*g_not;
    h(1) = -(1/factorial(1))*(((R^2)/(2*sigma^2))^(1))*exp(-(R^2)/(2*sigma^2)) + h_not;
    probtemp = probtemp_not + g(1)*h(1);

    while (recur <= 100)
        g(recur+1) = (1/(recur+1))*((r_not^2)/(2*sigma^2))*g(recur);
        h(recur+1) = (-1/factorial(recur+1))*(((R^2)/(2*sigma^2))^(recur+1))*exp(-(R^2)/(2*sigma^2)) + h(recur);
    end
end

```

```

        probtemp = probtemp + g(recur+1)*h(recur+1);
        recur = recur + 1;
    end

    probability(1,l) = probtemp;
end

% Initialise variables
count1 = 0;
count2 = 0;
count3 = 0;
count4 = 0;
count5 = 0;
count6 = 0;
SsumfirstleftTSS = 0;
SsumfirstleftTSSntoJIHigh = 0;
SsumfirstleftTSSntoJIModerate = 0;
SsumfirstleftPL = 0;
SsumfirstleftPLntoJIHigh = 0;
SsumfirstleftPLntoJIModerate = 0;

% Take the average of the pooled tracks
for m = 1:totalruns
    if ~isnan(sumfirstleftTSS(m))
        count1 = count1+1;
        SsumfirstleftTSS=SsumfirstleftTSS+sumfirstleftTSS(m);
    end

    if ~isnan(sumfirstleftTSSntoJIHigh(m))
        count2 = count2+1;

SsumfirstleftTSSntoJIHigh=SsumfirstleftTSSntoJIHigh+sumfirstleftTSSntoJIHigh(m);
    end

    if ~isnan(sumfirstleftTSSntoJIModerate(m))
        count3 = count3+1;

SsumfirstleftTSSntoJIModerate=SsumfirstleftTSSntoJIModerate+sumfirstleftTSSntoJIM
oderate(m);
    end

    if ~isnan(sumfirstleftPL(m))
        count4 = count4+1;
        SsumfirstleftPL=SsumfirstleftPL+sumfirstleftPL(m);
    end
end

```

```

if ~isnan(sumfirstleftPLntoJIHigh(m))
    count5 = count5+1;
    SsumfirstleftPLntoJIHigh=SsumfirstleftPLntoJIHigh+sumfirstleftPLntoJIHigh(m);
end

if ~isnan(sumfirstleftPLntoJIModerate(m))
    count6 = count6+1;

SsumfirstleftPLntoJIModerate=SsumfirstleftPLntoJIModerate+sumfirstleftPLntoJIModerate(m);
end
end

meanfirstleftTSS = SsumfirstleftTSS/count1;
meanfirstleftTSSntoJIHigh = SsumfirstleftTSSntoJIHigh/count2;
meanfirstleftTSSntoJIModerate = SsumfirstleftTSSntoJIModerate/count3;
meanfirstleftPL = SsumfirstleftPL/count4;
meanfirstleftPLntoJIHigh = SsumfirstleftPLntoJIHigh/count5;
meanfirstleftPLntoJIModerate = SsumfirstleftPLntoJIModerate/count6;
meantruetimedata = mean(sumkdata(:,1,:));
meandistuncert = mean(sumkdata(:,2,:));
meandistapart = mean(sumkdata(:,3,:));
meanesttimedata = mean(sumkdata(:,6,:));

% Calculate the sample variance of the estimated time to impact
for p = 1:ntime
    sumtimediff=0;
    timediff=0;
    for q = 1:totalruns
        timediff = (sumkdata(q,6,p)-meanesttimedata(p))^2;
        sumtimediff=sumtimediff+timediff;
    end
    meansampletimesigma(p) = sqrt(sumtimediff /(totalruns-1));
end

figure(30);
clf;
plot(meantruetimedata(:)',meandistuncert(:)','b',meantruetimedata(:)',meandistapart(:)','g')
legend('Distance Uncertainty','Distance Apart');
buf=sprintf('Estimated End Point');
title(buf)
xlim([min(lowest)*DT MAX_TIME-1*DT])
ylim([0 1000])
xlabel('Simulated Time (s)')

```

```
ylabel('Distance (m)')
```

```
figure(31);  
clf;  
plot(meantruetimedata(:)',abs((MAX_TIME-meantruetimedata(:))-  
meanesttimedata(:)'),'b',meantruetimedata(:)',meansampletimesigma,'r' )  
legend('Estimated Error','Standard Deviation');  
buf=sprintf('Error between Time Estimate and Truth');  
title(buf)  
xlim([min(lowest)*DT MAX_TIME-1*DT])  
ylim([0 100])  
xlabel('Simulated Time (s)')  
ylabel('Time (s)')
```

```
figure(32);  
clf;  
plot(meantruetimedata(:)',probability(:),'k')  
legend('Probability of Impact');  
buf=sprintf('Probability of Impact');  
title(buf)  
xlim([min(lowest)*DT MAX_TIME-1*DT])  
ylim([0 1])  
xlabel('Simulated Time (s)')  
ylabel('Probability of Impact')
```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This is a script file allows for multi-target multi-sensor tracking

% Define and initialise global variables
globals;
ginit;

buf=sprintf('Generating Target Information..Please Wait\n'); disp(buf);
% Set up time base for target simulations
times=0:DT:MAX_TIME;

% Generate all target's attack destinations
destinations=generate_destinations(times);

% Generate all target trajectories
targets=generate_targets(times);

% Generate platform information
platforms=generate_platforms(times);

% Assign sensors to platforms
sensors=assign_sensors(platforms);

buf=sprintf('Generating Observations...Please Wait\n'); disp(buf);

% Generate observations
observations=generate_observations(sensors,platforms,targets,times);

buf=sprintf('Displaying Information\n'); disp(buf);

% Display track information
post_sim(targets,observations,sensors,platforms,destinations);

% This file defines the global variables

% General information
global TSS_X;      % Traffic Separation Scheme x-coordinates

```

```

global TSS_Y;      % Traffic Separation Scheme y-coordinates
global PL_X;      % Port Limits x-coordinates
global PL_Y;      % Port Limits y-coordinates
global DT;        % simulation time-step
global MAX_TIME;   % maximum simulation time

% Target definitions
global TARGET_TYPE; % defines type of target (xy or V phi)
global NUM_TARGETS; % number of targets to be simulated

% Platform definitions
global PLATFORM;    % Platform data [x0, y0, phi0, vel]
global NUM_PLATFORMS; % number of platforms
global MAX_RANGE;   % maximum observable range
global SIGMA_XDOT;
global SIGMA_YDOT;

% Destination definitions
global DESTINATION; % Destination data [x0, y0, phi0, vel]
global NUM_DESTINATIONS; % number of destinations

% State and Sensor definitions
global SIGMA_Q;      % process noise standard deviation for feature model
global SIGMA_RANGE_MPA; % Range observation noise
global SIGMA_BEARING_MPA; % Bearing observation noise
global SIGMA_RANGE_MPAC; % Range observation noise
global SIGMA_BEARING_MPAC; % Bearing observation noise
global SIGMA_RANGE_PV; % Range observation noise
global SIGMA_BEARING_PV; % Bearing observation noise
global SIGMA_RANGE_PCG; % Range observation noise
global SIGMA_BEARING_PCG; % Bearing observation noise
global Rfilter_MPA; % observation noise covariance for MPA
global Rfilter_MPAC; % observation noise covariance for MPAC
global Rfilter_PV; % observation noise covariance for PV
global Rfilter_PCG; % observation noise covariance for PCG
global F; % state transition equation
global XSIZE; % state dimension
global ZSIZE; % observation dimension

% Results "database"
global kdata;
global esttimesigma;
global startplot;
global firstleftTSS;
global firstleftTSSntoJIHigh;

```

```

global firstleftTSSntoJIModerate;
global firstleftPL;
global firstleftPLntoJIHigh;
global firstleftPLntoJIModerate;
% This file initializes the global variables

% General polygon for Traffic Separation Scheme (TSS)
X = [738.9629; 700.8405; 503.0354; 343.3085; 301.0754; 188.7756; 142.3798;
291.7565; 356.6971; 474.1785; 703.9221; 738.9538; 738.9629];
Y = [443.8223; 412.1040; 377.3690; 303.7337; 330.3691; 359.0929; 316.1193;
230.0498; 289.3964; 351.7903; 379.3511; 408.0001; 443.8223];
TSS_X = X;
TSS_Y = Y;

% General polygon for Port Limits, estimated 0.5 mile beyond the TSS
[PL_X, PL_Y] = bufferm2('xy',X,Y,5,'out');

MAX_TIME=11500;      % simulation duration
DT=10;              % simulation time-step

% Target definitions
NUM_TARGETS=1;      % number of targets to be tracked

% Platform definitions
NUM_PLATFORMS=9;

% Destination definitions
NUM_DESTINATIONS=1;

% State and Sensor definitions
SIGMA_RANGE_MPA=5;    % 0.5nm range error equivalent
SIGMA_BEARING_MPA=0.0525; % 3 degrees bearing error in radians
SIGMA_RANGE_MPAC=6;    % 0.6nm range error equivalent
SIGMA_BEARING_MPAC=0.0875; % 5 degrees bearing error in radians
SIGMA_RANGE_PV=3;      % 0.3nm range error equivalent
SIGMA_BEARING_PV=0.0525; % 3 degrees bearing error in radians
SIGMA_RANGE_PCG=4;     % 0.4nm range error equivalent
SIGMA_BEARING_PCG=0.0875; % 5 degrees bearing error in radians
Rfilter_MPA = [SIGMA_RANGE_MPA^2 0;0 SIGMA_RANGE_MPA^2];
Rfilter_MPAC = [SIGMA_RANGE_MPAC^2 0;0 SIGMA_RANGE_MPAC^2];
Rfilter_PV = [SIGMA_RANGE_PV^2 0;0 SIGMA_RANGE_PV^2];
Rfilter_PCG = [SIGMA_RANGE_PCG^2 0;0 SIGMA_RANGE_PCG^2];
XSIZE=4;              % number of state components
ZSIZE=2;              % number of sensor measurement components
SIGMA_XDOT=0.00286;   % std dev of 1knot in the x-direction

```



```

SIGMA_YDOT=0.00286;      % std dev of 1knot in the y-direction

% Results "database"
kdata = [7 MAX_TIME/DT+1];
function [latb,lonb] = bufferm2(varargin) %lat,lon,dist,direction,npts,outputformat)
%BUFFERM2 Computes buffer zone around a polygon
%
% [latb,lonb] = bufferm2(lat,lon,dist,direction)
% [latb,lonb] = bufferm2(lat,lon,dist,direction,npts)
% [latb,lonb] = bufferm2(lat,lon,dist,direction,npts,outputformat)
% [xb, yb] = bufferm2('xy',x,y,dist,direction,npts,outputformat)
%
% This function was originally designed as a replacement for the Mapping
% Toolbox function bufferm, which calculates a buffer zone around a
% polygon. The original bufferm function had some serious bugs that could
% result in incorrect buffer results and/or errors, and was also very slow.
% As of R2006b, those bugs have been fixed. However, this version still
% maintains a few advantages over the original:
%
% - Can be applied to polygons in either geographical space (as in
%   bufferm) or in cartesian coordinates.
%
% - Better treatment of polygon holes. The original function simply
%   filled in all holes; this version trims or pads holes according to the
%   buffer width given.
%
% Input and output format is identical to bufferm unless the 'xy' option is
% specified, so it can be used interchangeably.
%
% Input variables:
%
% lat:      Latitude values defining the polygon to be buffered.
%           This can be either a NaN-delimited vector, or a cell
%           array containing individual polygonal contours (each of
%           which is a vector). External contours should be listed
%           in a clockwise direction, and internal contours (holes)
%           in a counterclockwise direction.
%
% lon:      Longitude values defining the polygon to be buffered.
%           Same format as lat.
%
% dist:     Width of buffer, in degrees of arc along the surface
%           (unless 'xy' is used, in which case units correspond to
%           x-y coordinates)
%

```

```

% direction:    'in' or 'out'
%
% npts:         Number of points used to construct the circles around
%               each polygon vertex. If omitted, default is 13.
%
% outputformat: 'vector' (NaN-delimited vectors), 'cutvector'
%               (NaN-clipped vectors with cuts connecting holes to the
%               exterior of the polygon), or 'cell' (cell arrays in
%               which each element of the cell array is a separate
%               polygon), defining format of output. If omitted,
%               default is 'vector'.
%
% 'xy':         If first input is 'xy', then data will be assumed to
%               lie on a cartesian plane rather than on a sphere. Use
%               x and y coordinates as first two inputs rather than lat
%               and lon. Units of x, y, and distance should be the
%               same.
%
% Output variables:
%
% latb:         Latitude values for buffer polygon
%
% lonb:         Longitude values for buffer polygon
%
% Example:
%
% load conus
% tol = 0.1; % Tolerance for simplifying polygon outlines
% [reducedlat, reducedlon] = reducem(gtlakelat, gtlakelon, tol);
% dist = 1; % Buffer distance in degrees
% [latb, lonb] = bufferm2(reducedlat, reducedlon, dist, 'out');
% figure('Renderer','painters')
% usamap({'MN','NY'})
% geoshow(latb, lonb, 'DisplayType', 'polygon', 'FaceColor', 'yellow')
% geoshow(gtlakelat, gtlakelon,...
%         'DisplayType', 'polygon', 'FaceColor', 'blue')
% geoshow(uslat, uslon)
% geoshow(statelat, statelon)
%
% See also:
%
% bufferm, polybool
%
% Copyright 2010 Kelly Kearney

```

```

%-----
% Check input
%-----

error(nargchk(3,7,nargin));

% Determine if geographic or cartesian

if ischar(varargin{1}) && strcmp(varargin{1}, 'xy')
    geo = false;
    param = varargin(2:end);
else
    geo = true;
    param = varargin;
end

% Set defaults if not provided as input

nparam = length(param);

if geo
    [lat, lon, dist] = deal(param{1:3});
else
    [lon, lat, dist] = deal(param{1:3}); % lon = x, lat = y for mental clarity, will switch
    back at end
end

if nparam < 4
    direction = 'out';
else
    direction = param{4};
end

if nparam < 5
    npts = 13;
else
    npts = param{5};
end

if nparam < 6
    outputformat = 'vector';
else
    outputformat = param{6};
end

```

```

% Check format and dimensions of input

if ~ismember(direction, {'in', 'out'})
    error('Direction must be either "in" or "out".');
end

if ~ismember(outputformat, {'vector', 'cutvector', 'cell'})
    error('Unrecognized output format flag. ');
end

if ~isnumeric(dist) || numel(dist) > 1
    error('Distance must be a scalar. ');
end

if ~isnumeric(npts) || numel(npts) > 1
    error('Number of points must be a scalar. ');
end

if iscell(lat)
    for il = 1:numel(lat)
        if ~isvector(lat{il}) || ~isvector(lon{il}) || ~isequal(length(lat{il}), length(lon{il}))
            error('Lat (or x) and lon (or y) must be vectors or cells of vectors with identical dimensions');
        end
        lat{il} = lat{il}(:);
        lon{il} = lon{il}(:);
    end
else
    if ~isvector(lat) || ~isvector(lon) || ~isequal(length(lat), length(lon))
        error('Lat (or x) and lon (or y) must be vectors or cells of vectors with identical dimensions');
    end
    lat = lat(:);
    lon = lon(:);
end

%-----
% Split polygon(s) into
% separate faces
%-----

if iscell(lat)
    [lat, lon] = polyjoin(lat, lon); % In case multiple faces in one cell.
end

```

```

[latcells, loncells] = polysplit(lat, lon);

%-----
% Create buffer shapes
%-----

plotflag = 0;

if plotflag

    Plt.x = lon;
    Plt.y = lat;

end

latcrall = cell(0);
loncrall = cell(0);

for ipoly = 1:length(latcells)

    % Circles around each vertex

    if geo
        [latc, lonc] = calccircgeo(latcells{ipoly}, loncells{ipoly}, dist, npts);
    else
        [lonc, latc] = calccircart(loncells{ipoly}, latcells{ipoly}, dist, npts);
    end

    % Rectangles around each edge

    if geo
        [latr, lonr] = calcrecgeo(latcells{ipoly}, loncells{ipoly}, dist);
    else
        [lonr, latr] = calcrecart(loncells{ipoly}, latcells{ipoly}, dist);
    end

    % Union of circles and rectangles

    if plotflag
        Plt.rectx = lonr;
        Plt.recty = latr;
        Plt.circx = lonc;
        Plt.circy = latc;
    end
end

```

```

[latc, lonc] = multipolyunion(latc, lonc);
[latr, lonr] = multipolyunion(latr, lonr);

if plotflag
    Plt.rectcombox = lonr;
    Plt.rectcomboy = latr;
    Plt.circcombox = lonc;
    Plt.circcomboy = latc;
end

[loncr, latcr] = polybool('+', lonr, latr, lonc, latc);

% Union of new circle/rectangle combo with that from other faces

[loncrall, latcrall] = polybool('+', loncrall, latcrall, loncr, latcr);

% Plotting (for debugging only)

if plotflag

    Plt.allx = loncrall;
    Plt.ally = latcrall;

    if ipoly == 1
        figure;
        plot(Plt.x, Plt.y, 'k', 'linewidth', 2);
        hold on
    end

    plot(cat(2, Plt.rectx{:}), cat(2, Plt.recty{:}), 'b');
    plot(cat(2, Plt.circx{:}), cat(2, Plt.circy{:}), 'r');
    plot(Plt.allx{1}, Plt.ally{1}, 'g', 'linewidth', 2);

end

end

%-----
% Calculate union/difference
%-----

switch direction
case 'out'
    [lonb, latb] = polybool('+', loncells, latcells, loncrall, latcrall);
case 'in'

```

```

    [lonb, latb] = polybool('-', loncells, latcells, loncrall, latcrall);
end

if plotflag
    [Plt.yfinal, Plt.xfinal] = polyjoin(latb, lonb);
    plot(Plt.xfinal, Plt.yfinal, 'linestyle', '--', 'color', [0 .5 0], 'linewidth', 2);
end

%-----
% Reformat output
%-----

if ~geo
    y = latb; % Switch, since cartesian uses opposite order
    x = lonb;
    latb = x;
    lonb = y;
end

switch outputformat
    case 'vector'
        [latb, lonb] = polyjoin(latb, lonb);
    case 'cutvector'
        [latb, lonb] = polycut(latb, lonb);
    case 'cell'
end

%*****
%*****

function [latc, lonc] = calccircgeo(lat, lon, radius, npts)
% lat and lon: n x 1 vectors
% radius: scalar

radius = ones(length(lat),1) * radius;
[latc, lonc] = scircle1(lat, lon, radius, [], [], [], npts);
latc = num2cell(latc, 1);
lonc = num2cell(lonc, 1);

function [latr, lonr] = calcrcgeo(lat, lon, halfwidth)
% lat and lon: n x 1 vectors
% halfwidth: scalar

range = halfwidth * ones(length(lat)-1, 1);

```

```

az = azimuth(lat(1:end-1), lon(1:end-1), lat(2:end), lon(2:end));

[latbl1,lonbl1] = reckon(lat(1:end-1), lon(1:end-1), range, az-90);
[latbr1,lonbr1] = reckon(lat(1:end-1), lon(1:end-1), range, az+90);
[latbl2,lonbl2] = reckon(lat(2:end), lon(2:end), range, az-90);
[latbr2,lonbr2] = reckon(lat(2:end), lon(2:end), range, az+90);

latr = [latbl1 latbl2 latbr2 latbr1 latbl1]';
lonr = [lonbl1 lonbl2 lonbr2 lonbr1 lonbl1]';
latr = num2cell(latr, 1);
lonr = num2cell(lonr, 1);

function [latu, lonu] = multipolyunion(lat, lon)
% lat and lon are n x 1 cell arrays of vectors

latu = lat{1};
lonu = lon{1};

for ip = 2:length(lat)
    [lonu, latu] = polybool('+', lonu, latu, lon{ip}, lat{ip});
end
[latu, lonu] = polysplit(latu, lonu);

function [xc, yc] = calccirccart(x, y, radius, npts)

ang = linspace(0, 2*pi, npts+1);
ang = ang(end-1:-1:1);
xc = bsxfun(@plus, x, radius * cos(ang));
yc = bsxfun(@plus, y, radius * sin(ang));
xc = num2cell(xc', 1);
yc = num2cell(yc', 1);

% if ~ispolycw(x,y)
%   [xc,yc] = poly2ccw(xc,yc);
% end

function [xrec, yrec] = calcreccart(x, y, halfwidth)

dx = diff(x);
dy = diff(y);

is1 = dx >= 0 & dy >= 0;
is2 = dx < 0 & dy >= 0;

```



```

is3 = dx < 0 & dy < 0;
is4 = dx >= 0 & dy < 0;

ish1 = dy == 0 & dx > 0;
ish2 = dy == 0 & dx < 0;

theta = zeros(5,1);
theta(is1 | is3) = atan(dy(is1 | is3)./dx(is1 | is3));
theta(is2 | is4) = -atan(dy(is2 | is4)./dx(is2 | is4));

[xl,xr,yl,yr] = deal(zeros(size(dx)));

xl(is1) = -halfwidth * sin(theta(is1));
xr(is1) = halfwidth * sin(theta(is1));
yl(is1) = halfwidth * cos(theta(is1));
yr(is1) = -halfwidth * cos(theta(is1));

xl(is2) = -halfwidth * sin(theta(is2));
xr(is2) = halfwidth * sin(theta(is2));
yl(is2) = -halfwidth * cos(theta(is2));
yr(is2) = halfwidth * cos(theta(is2));

xl(is3) = halfwidth * sin(theta(is3));
xr(is3) = -halfwidth * sin(theta(is3));
yl(is3) = -halfwidth * cos(theta(is3));
yr(is3) = halfwidth * cos(theta(is3));

xl(is4) = halfwidth * sin(theta(is4));
xr(is4) = -halfwidth * sin(theta(is4));
yl(is4) = halfwidth * cos(theta(is4));
yr(is4) = -halfwidth * cos(theta(is4));

xrec = [xl+x(1:end-1) xl+x(2:end) xr+x(2:end) xr+x(1:end-1) xl+x(1:end-1)];
yrec = [yl+y(1:end-1) yl+y(2:end) yr+y(2:end) yr+y(1:end-1) yl+y(1:end-1)];

xrec = num2cell(xrec, 2);
yrec = num2cell(yrec, 2);

```

% This function draws a circle for a given center and radius

```
function bubble = circle(center,radius)
```

```
Resolution = 100;  
THETA=linspace(0,2*pi,Resolution);  
RHO=ones(1,Resolution)*radius;  
[X,Y] = pol2cart(THETA,RHO);  
bubble.X=X+center(1);  
bubble.Y=Y+center(2);
```

% This function generates the SAW's intended destinations
% (i.e. Areas A1, A2, and A3 of Jurong Island)

```
function destinations = generate_destinations(times)
```

% Define the global variables
globals;

% Define the destination locations

```
for i=1:NUM_DESTINATIONS
```

```
    % Area A1
```

```
    if i==1
```

```
        destinations(1,i) = make_destination;  
        destinations(1,i).time=times(1);  
        destinations(1,i).id=i;  
        destinations(1,i).x = 302.1144444;  
        destinations(1,i).y = 347.7716667;
```

```
    % Now iterate for all times
```

```
    [temp,ntimes]=size(times);
```

```
    for n=2:ntimes
```

```
        destinations(n,i)=destination_step(destinations(n-1,i),times(n));
```

```
    end
```

```
end
```

```
    % Area A2
```

```
    if i==2
```

```
        destinations(1,i) = make_destination;  
        destinations(1,i).time=times(1);  
        destinations(1,i).id=i;  
        destinations(1,i).x = 322.7311111;  
        destinations(1,i).y = 367.2116667;
```

```
    % Now iterate for all times
```

```

[temp,ntimes]=size(times);
for n=2:ntimes
    destinations(n,i)=destination_step(destinations(n-1,i),times(n));
end
end

% Area A3
if i==3
    destinations(1,i) = make_destination;
    destinations(1,i).time=times(1);
    destinations(1,i).id=i;

    destinations(1,i).x = 343.3450000;
    destinations(1,i).y = 382.5572222;

    % Now iterate for all times
    [temp,ntimes]=size(times);
    for n=2:ntimes
        destinations(n,i)=destination_step(destinations(n-1,i),times(n));
    end
end
end
% This function initialises a new destination

function new=make_destination

new=struct('id',0,'time',0,'x',0,'y',0,'phi',0,'vel',0,'gamma',0);
% This function is to step a destination

function next_destination=destination_step(destination,time)

globals;

% This is a no motion model

dt=time-destination.time;
next_destination = make_destination; % space for destination data structure
next_destination.id=destination.id;
next_destination.time = time;
next_destination.x = destination.x;
next_destination.y = destination.y;
next_destination.phi = destination.phi;
next_destination.vel = destination.vel;
next_destination.gamma = destination.gamma;

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function generates true target information for later observation
% and tracking. Dynamics of target are contained in "target_step", number
% targets is defined by the globally defined variable NUM_TARGETS. A data
% point is generated at each of a set of "times."

function targets = generate_targets(times)

% Define the global variables
globals;

% Define a SAW at pre-fixed location in the TSS
for i=1
    targets(1,i) = make_target;
    targets(1,i).time=times(1);
    targets(1,i).id=i;
    targets(1,i).x = 738.9616667;
    targets(1,i).y = 438.7050000;
    targets(1,i).gamma=0;

    % Now iterate for all times
    [temp,ntimes]=size(times);
    for n=2:ntimes
        targets(n,i)=target_step_SAW_path(targets(n-1,i),times(n));
    end
end

% Create non-SAW contacts at random locations within the TSS
% Not used since this thesis covers a single target
for i=2:NUM_TARGETS
    targets(1,i) = make_target;
    targets(1,i).time=times(1);
    targets(1,i).id=i;

    x = TSS_X(1)*rand;
    y = TSS_Y(1)*rand;

    while (inpolygon (x, y,TSS_X, TSS_Y) ~= 1)

```

```

    x = TSS_X(1)*rand;
    y = TSS_Y(1)*rand;
end

    targets(1,i).x = x;
    targets(1,i).y = y;
    targets(1,i).phi = 2*pi*rand -pi;
    targets(1,i).vel = MIN_TARGET_VEL + (MAX_TARGET_VEL-
MIN_TARGET_VEL)*rand;
    targets(1,i).gamma=0;

    % Now iterate for all times
    [temp,ntimes]=size(times);
    for n=2:ntimes
        targets(n,i)=target_step(targets(n-1,i),times(n));
    end
end

```

```
% Acknowledgement:  
% This code originates from H. Durrant-Whyte  
% Australian Centre for Field Robotics  
% The University of Sydney, Australia  
  
% This function initialises a new target  
  
function new=make_target  
  
new=struct('id',0,'time',0,'x',0,'y',0,'phi',0,'vel',0,'gamma',0);
```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This is the trajectory taken by the SAW

function next_target=target_step_SAW_path(target,time)

globals;

dt=time-target.time;
next_target = make_target;
next_target.id=target.id;
next_target.time = time;

% target.vel = 0.042870370 (in m/s which equates to 15knots)
% target.vel = 0.059160494 (in m/s which equates to 20knots)
% target.vel = 0.060018519 (in m/s which equates to 21knots)
% target.vel = 0.062876543 (in m/s which equates to 22knots)

if (next_target.time >= 0 && next_target.time < 1086)
    next_target.x = target.x + dt*0.042870370*cos(-pi+0.720);
    next_target.y = target.y + dt*0.042870370*sin(-pi+0.720);
    next_target.phi = -pi+0.720;
    next_target.vel = 0.042870370;

elseif (next_target.time >= 1086 && next_target.time < 5890)
    next_target.x = target.x + dt*0.042870370*cos(-pi+0.170);
    next_target.y = target.y + dt*0.042870370*sin(-pi+0.170);
    next_target.phi = -pi+0.170;
    next_target.vel = 0.042870370;

elseif (next_target.time >= 5890 && next_target.time < 10199)
    next_target.x = target.x + dt*0.042870370*cos(-pi+0.430);
    next_target.y = target.y + dt*0.042870370*sin(-pi+0.430);
    next_target.phi = -pi+0.430;
    next_target.vel = 0.042870370;

elseif (next_target.time >= 10199 && next_target.time < 11051)
    next_target.x = target.x + dt*0.042870370*cos(pi-0.770);
    next_target.y = target.y + dt*0.042870370*sin(pi-0.770);

```

```
next_target.phi = pi-0.770;
next_target.vel = 0.042870370;

elseif (next_target.time >= 11051 && next_target.time < 11500)
    next_target.x = target.x + dt*0.059160494*cos(pi-1.360);
    next_target.y = target.y + dt*0.059160494*sin(pi-1.360);
    next_target.phi = pi-1.360;
    next_target.vel = 0.059160494;

elseif (next_target.time >= 11500)
    next_target.x = target.x;
    next_target.y = target.y;
    next_target.phi = 0;
    next_target.vel = 0;
end
```



```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function generates trajectories for platforms

function platforms = generate_platforms(times)

% Define the global variables
globals;

% Define 5 fixed platform initial locations

% Model the MPA radar stations
for i=1:NUM_PLATFORMS
    if i==1
        platforms(1,i) = make_platform;
        platforms(1,i).time=times(1);
        platforms(1,i).id=i;
        platforms(1,i).x = 285.4700000;
        platforms(1,i).y = 360.8527778;
        platforms(1,i).phi = 2*pi*rand -pi;
        platforms(1,i).vel = 0;
        platforms(1,i).gamma=0;

        % Now iterate for all times
        [temp,ntimes]=size(times);
        for n=2:ntimes
            platforms(n,i)=platform_step(platforms(n-1,i),times(n));
        end
    end

    if i==2
        platforms(1,i) = make_platform;
        platforms(1,i).time=times(1);
        platforms(1,i).id=i;
        platforms(1,i).x = 342.7972222;
        platforms(1,i).y = 311.9850000;
        platforms(1,i).phi = 2*pi*rand -pi;
        platforms(1,i).vel = 0;
        platforms(1,i).gamma=0;
    end
end

```

```

% Now iterate for all times
[temp,ntimes]=size(times);
for n=2:ntimes
    platforms(n,i)=platform_step(platforms(n-1,i),times(n));
end
end

if i==3
    platforms(1,i) = make_platform;
    platforms(1,i).time=times(1);
    platforms(1,i).id=i;
    platforms(1,i).x = 413.1333333;
    platforms(1,i).y = 349.8194444;
    platforms(1,i).phi = 2*pi*rand -pi;
    platforms(1,i).vel = 0;
    platforms(1,i).gamma=0;

    % Now iterate for all times
    [temp,ntimes]=size(times);
    for n=2:ntimes
        platforms(n,i)=platform_step(platforms(n-1,i),times(n));
    end
end

if i==4
    platforms(1,i) = make_platform;
    platforms(1,i).time=times(1);
    platforms(1,i).id=i;
    platforms(1,i).x = 461.4761111;
    platforms(1,i).y = 403.4261111;
    platforms(1,i).phi = 2*pi*rand -pi;
    platforms(1,i).vel = 0;
    platforms(1,i).gamma=0;

    % Now iterate for all times
    [temp,ntimes]=size(times);
    for n=2:ntimes
        platforms(n,i)=platform_step(platforms(n-1,i),times(n));
    end
end

if i==5
    platforms(1,i) = make_platform;
    platforms(1,i).time=times(1);

```

```

platforms(1,i).id=i;
platforms(1,i).x = 517.0844444;
platforms(1,i).y = 421.9305556;
platforms(1,i).phi = 2*pi*rand -pi;
platforms(1,i).vel = 0;
platforms(1,i).gamma=0;

% Now iterate for all times
[temp,ntimes]=size(times);
for n=2:ntimes
    platforms(n,i)=platform_step(platforms(n-1,i),times(n));
end
end

% Define 3 mobile platform initial locations

% Model the CPC
if i==6
    platforms(1,i) = make_platform;
    platforms(1,i).time=times(1);
    platforms(1,i).id=i;
    platforms(1,i).x = 474.1785056;
    platforms(1,i).y = 351.7903000;
    platforms(1,i).phi = 2*pi*rand -pi;
    platforms(1,i).vel = 0;
    targets(1,i).gamma=0;
    % Now iterate for all times
    [temp,ntimes]=size(times);
    for n=2:ntimes
        platforms(n,i)=platform_step_PCG1_path(platforms(n-1,i),times(n));
    end
end

if i==7
    platforms(1,i) = make_platform;
    platforms(1,i).time=times(1);
    platforms(1,i).id=i;
    platforms(1,i).x = 474.1785056;
    platforms(1,i).y = 351.7903000;
    platforms(1,i).phi = 2*pi*rand -pi;
    platforms(1,i).vel = 0;
    targets(1,i).gamma=0;

    % Now iterate for all times
    [temp,ntimes]=size(times);

```

```

    for n=2:ntimes
        platforms(n,i)=platform_step_PCG2_path(platforms(n-1,i),times(n));
    end
end

% Model the PV
if i==8
    platforms(1,i) = make_platform;
    platforms(1,i).time=times(1);
    platforms(1,i).id=i;
    platforms(1,i).x = 474.1785056;
    platforms(1,i).y = 351.7903000;
    platforms(1,i).phi = 2*pi*rand -pi;
    platforms(1,i).vel = 0;
    targets(1,i).gamma=0;

    % Now iterate for all times
    [temp,ntimes]=size(times);
    for n=2:ntimes
        platforms(n,i)=platform_step_PV_path(platforms(n-1,i),times(n));
    end
end

% Given the speed and sensor coverage of the MPaA,
% its surveying of the small TSS can be modeled as a "fixed" sensor
% with complete of the TSS during its operation.
% Model the MPaA
if i==9
    platforms(1,i) = make_platform; % make the platform data structure
    platforms(1,i).time=times(1);
    platforms(1,i).id=i;
    platforms(1,i).x = 474.1785056;
    platforms(1,i).y = 351.7903000;
    platforms(1,i).phi = 2*pi*rand -pi;
    platforms(1,i).vel = 0;
    targets(1,i).gamma=0;

    % Now iterate for all times
    [temp,ntimes]=size(times);
    for n=2:ntimes
        platforms(n,i)=platform_step_MPAC(platforms(n-1,i),times(n));
    end
end
end
end

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% This function initialises a new platform

function new=make_platform

new=struct('id',0,'time',0,'x',0,'y',0,'phi',0,'vel',0,'gamma',0);
% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% This is a no motion platform model for MPA radar station

function next_platform=platform_step(platform,time)

globals;

dt=time-platform.time;
next_platform = make_platform;
next_platform.id=platform.id;
next_platform.time = time;
next_platform.x = platform.x;
next_platform.y = platform.y;
next_platform.phi = platform.phi;
next_platform.vel = platform.vel;
next_platform.gamma = platform.gamma;
% This is the platform motion model for one of the two CPCs for a day

function next_platform=platform_step_PCG1_path(platform,time)

globals;

dt=time-platform.time;
next_platform = make_platform;
next_platform.id=platform.id;
next_platform.time = time;

if (next_platform.time >= 0)
    next_platform.x = 474.1785056;
    next_platform.y = 351.7903000;

```

```

    next_platform.phi = platform.phi;
    next_platform.vel = platform.vel;
    next_platform.gamma = platform.gamma;
end

%-----
% This models the platform's patrol path. However, by making the assumption
% that the sensors have extended range to simplify sensor management, this
% modeling is not used here. This will be useful for future work with the
% inclusion of sensor management.
%
% if (next_platform.time >= 0 && next_platform.time < 18000)
%     next_platform.x = platform.x + dt*0.015061617*cos(1.519574188-0.5*pi);
%     next_platform.y = platform.y + dt*0.015061617*sin(1.519574188-0.5*pi);
%     next_platform.phi = 1.519574188-0.5*pi;
%     next_platform.vel = 0.015061617;
%     next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 18000 && next_platform.time < 32400)
%     next_platform.x = platform.x + dt*0.025102695*cos(1.519574188+0.5*pi);
%     next_platform.y = platform.y + dt*0.025102695*sin(1.519574188+0.5*pi);
%     next_platform.phi = 1.519574188+0.5*pi;
%     next_platform.vel = 0.025102695;
%     next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 32400 && next_platform.time < 46800)
%     next_platform.x = platform.x + dt*0.015061617*cos(1.519574188-0.5*pi);
%     next_platform.y = platform.y + dt*0.015061617*sin(1.519574188-0.5*pi);
%     %next_platform.phi = 1.519574188-0.5*pi;
%     %next_platform.vel = 0.015061617;
%     %next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 46800 && next_platform.time < 61200)
%     next_platform.x = platform.x + dt*0.025102695*cos(1.519574188+0.5*pi);
%     next_platform.y = platform.y + dt*0.025102695*sin(1.519574188+0.5*pi);
%     next_platform.phi = 1.519574188+0.5*pi;
%     next_platform.vel = 0.025102695;
%     next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 61200 && next_platform.time < 75600)
%     next_platform.x = platform.x + dt*0.015061617*cos(1.519574188-0.5*pi);
%     next_platform.y = platform.y + dt*0.015061617*sin(1.519574188-0.5*pi);
%     %next_platform.phi = 1.519574188-0.5*pi;
%     %next_platform.vel = 0.015061617;
%     %next_platform.gamma = platform.gamma;

```

```

%
% elseif (next_platform.time >= 75600 && next_platform.time < 86400)
%   next_platform.x = platform.x + dt*0.025102695*cos(1.519574188+0.5*pi);
%   next_platform.y = platform.y + dt*0.025102695*sin(1.519574188+0.5*pi);
%   next_platform.phi = 1.519574188+0.5*pi;
%   next_platform.vel = 0.025102695;
%   next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 86400)
%   next_platform.x = platform.x;
%   next_platform.y = platform.y;
%   next_platform.phi = 0;
%   next_platform.vel = 0;
%   next_platform.gamma = 0;
% end
%-----

```

```

% This is the platform motion model for one of the two CPCs for a day

function next_platform=platform_step_PCG2_path(platform,time)

globals;

dt=time-platform.time;
next_platform = make_platform;
next_platform.id=platform.id;
next_platform.time = time;

if (next_platform.time >= 0)
    next_platform.x = 474.1785056;
    next_platform.y = 351.7903000;
    next_platform.phi = platform.phi;
    next_platform.vel = platform.vel;
    next_platform.gamma = platform.gamma;
end

%-----
% This models the platform's patrol path. However, by making the assumption
% that the sensors have extended range to simplify sensor management, this
% modeling is not used here. This will be useful for future work with the
% inclusion of sensor management.
%
% if (next_platform.time >= 0 && next_platform.time < 18000)
%     next_platform.x = platform.x + dt*0.013140537*cos(-1.330818664+0.5*pi);
%     next_platform.y = platform.y + dt*0.013140537*sin(-1.330818664+0.5*pi);
%     next_platform.phi = -1.330818664+0.5*pi;
%     next_platform.vel = 0.013140537;
%     next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 18000 && next_platform.time < 32400)
%     next_platform.x = platform.x + dt*0.021900894*cos(-1.330818664-0.5*pi);
%     next_platform.y = platform.y + dt*0.021900894*sin(-1.330818664-0.5*pi);
%     next_platform.phi = -1.330818664-0.5*pi;
%     next_platform.vel = 0.021900894;
%     next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 32400 && next_platform.time < 46800)
%     next_platform.x = platform.x + dt*0.013140537*cos(-1.330818664+0.5*pi);
%     next_platform.y = platform.y + dt*0.013140537*sin(-1.330818664+0.5*pi);
%     next_platform.phi = -1.330818664+0.5*pi;
%     next_platform.vel = 0.013140537;

```



```

%   next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 46800 && next_platform.time < 61200)
%   next_platform.x = platform.x + dt*0.021900894*cos(-1.330818664-0.5*pi);
%   next_platform.y = platform.y + dt*0.021900894*sin(-1.330818664-0.5*pi);
%   next_platform.phi = -1.330818664-0.5*pi;
%   next_platform.vel = 0.021900894;
%   next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 61200 && next_platform.time < 75600)
%   next_platform.x = platform.x + dt*0.013140537*cos(-1.330818664+0.5*pi);
%   next_platform.y = platform.y + dt*0.013140537*sin(-1.330818664+0.5*pi);
%   next_platform.phi = -1.330818664+0.5*pi;
%   next_platform.vel = 0.013140537;
%   next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 75600 && next_platform.time < 86400)
%   next_platform.x = platform.x + dt*0.021900894*cos(-1.330818664-0.5*pi);
%   next_platform.y = platform.y + dt*0.021900894*sin(-1.330818664-0.5*pi);
%   next_platform.phi = -1.330818664-0.5*pi;
%   next_platform.vel = 0.021900894;
%   next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 86400)
%   next_platform.x = platform.x;
%   next_platform.y = platform.y;
%   next_platform.phi = 0;
%   next_platform.vel = 0;
%   next_platform.gamma = 0;
% end
%-----

```

```

% This is the platform motion model for PV for a day

function next_platform=platform_step_PV_path(platform,time)

globals;

dt=time-platform.time;
next_platform = make_platform;
next_platform.id=platform.id;
next_platform.time = time;

if (next_platform.time >= 0)
    next_platform.x = 474.1785056;
    next_platform.y = 351.7903000;
    next_platform.phi = platform.phi;
    next_platform.vel = platform.vel;
    next_platform.gamma = platform.gamma;
end

%-----
% This models the platform's patrol path. However, by making the assumption
% that the sensors have extended range to simplify sensor management, this
% modeling is not used here. This will be useful for future work with the
% inclusion of sensor management.
%
% if (next_platform.time >= 0 && next_platform.time < 43200)
%     next_platform.x = platform.x + dt*0.013972407*cos(0.152813367);
%     next_platform.y = platform.y + dt*0.013972407*sin(0.152813367);
%     next_platform.phi = 0.152813367;
%     next_platform.vel = 0.013972407;
%     next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 43200 && next_platform.time < 86400)
%     next_platform.x = platform.x + dt*0.013972407*cos(0.152813367+pi);
%     next_platform.y = platform.y + dt*0.013972407*sin(0.152813367+pi);
%     next_platform.phi = 0.152813367+pi;
%     next_platform.vel = 0.013972407;
%     next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 86400)
%     next_platform.x = platform.x;
%     next_platform.y = platform.y;
%     next_platform.phi = 0;
%     next_platform.vel = 0;

```

```
% next_platform.gamma = 0;  
% end  
%-----
```

```

% This is the platform motion model for MPaA

function next_platform=platform_step_MPAC(platform,time)

globals;

next_platform = make_platform;
next_platform.id=platform.id;
next_platform.time = time;

if (next_platform.time >= 0)
    next_platform.x = 474.1785056;
    next_platform.y = 351.7903000;
    next_platform.phi = platform.phi;
    next_platform.vel = platform.vel;
    next_platform.gamma = platform.gamma;
end

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function creates and assigns sensors to platforms
% It assigns one sensor to one platform

function sensors=assign_sensors(platforms)

globals;

% Define sensors for the 5 fixed platforms

for i=1:NUM_PLATFORMS
    if i==1
        sensors(i)=make_sensor;
        sensors(i).id=i;
        sensors(i).platform=i;
        sensors(i).r_err=SIGMA_RANGE_MPA;
        sensors(i).b_err=SIGMA_BEARING_MPA;
        sensors(i).point=0;      % zero point angle
        sensors(i).beam_view=2*pi;
        sensors(i).max_range=154; % an equivalent of 15nm

    elseif i==2
        sensors(i)=make_sensor;
        sensors(i).id=i;
        sensors(i).platform=i;
        sensors(i).r_err=SIGMA_RANGE_MPA;
        sensors(i).b_err=SIGMA_BEARING_MPA;
        sensors(i).point=0;
        sensors(i).beam_view=2*pi;
        sensors(i).max_range=154;

    elseif i==3
        sensors(i)=make_sensor;
        sensors(i).id=i;
        sensors(i).platform=i;
        sensors(i).r_err=SIGMA_RANGE_MPA;
        sensors(i).b_err=SIGMA_BEARING_MPA;
        sensors(i).point=0;

```

```

sensors(i).beam_view=2*pi;
sensors(i).max_range=154;

elseif i==4
    sensors(i)=make_sensor;
    sensors(i).id=i;
    sensors(i).platform=i;
    sensors(i).r_err=SIGMA_RANGE_MPA;
    sensors(i).b_err=SIGMA_BEARING_MPA;
    sensors(i).point=0;
    sensors(i).beam_view=2*pi;
    sensors(i).max_range=154;

elseif i==5
    sensors(i)=make_sensor;
    sensors(i).id=i;
    sensors(i).platform=i;
    sensors(i).r_err=SIGMA_RANGE_MPA;
    sensors(i).b_err=SIGMA_BEARING_MPA;
    sensors(i).point=0;
    sensors(i).beam_view=2*pi;
    sensors(i).max_range=154;

% Define sensors for the 4 mobile platforms

% Define the CPC sensor
elseif i==6
    sensors(i)=make_sensor;
    sensors(i).id=i;
    sensors(i).platform=i;
    sensors(i).r_err=SIGMA_RANGE_PCG;
    sensors(i).b_err=SIGMA_BEARING_PCG;
    sensors(i).point=0;
    sensors(i).beam_view=2*pi;
    sensors(i).max_range=1000; % range extended to simulate
                              % presence at Jurong Island

% Define the CPC sensor
elseif i==7
    sensors(i)=make_sensor;
    sensors(i).id=i;
    sensors(i).platform=i;
    sensors(i).r_err=SIGMA_RANGE_PCG;
    sensors(i).b_err=SIGMA_BEARING_PCG;
    sensors(i).point=0;

```

```

sensors(i).beam_view=2*pi;
sensors(i).max_range=1000;

% Define the PV sensor
elseif i==8
    sensors(i)=make_sensor;
    sensors(i).id=i;
    sensors(i).platform=i;
    sensors(i).r_err=SIGMA_RANGE_PV;
    sensors(i).b_err=SIGMA_BEARING_PV;
    sensors(i).point=0;
    sensors(i).beam_view=2*pi;
    sensors(i).max_range=1000;

% Define the MPaA sensor
elseif i==9
    sensors(i)=make_sensor;
    sensors(i).id=i;
    sensors(i).platform=i;
    sensors(i).r_err=SIGMA_RANGE_MPAC;
    sensors(i).b_err=SIGMA_BEARING_MPAC;
    sensors(i).point=0;
    sensors(i).beam_view=2*pi;
    sensors(i).max_range=1000;
end
end

```

```
% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% This function initialises a new sensor

function new=make_sensor

new=struct('id',0,'platform',0,'r_err',0,'b_err',0,'point',0,...
          'beam_view',0,'max_range',0);
```



```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function generates observations from sensors located on
% platforms observing targets for the stipulated time-steps

function observations=generate_observations(sensors,platforms,tracks,times)

[temp,nsensors]=size(sensors);
for s=1:nsensors
    ntimes=1;
    for t=times
        observations(s,ntimes).sensor=s;
        observations(s,ntimes).time=t;
        observations(s,ntimes).report=sensor_report(sensors(s),platforms,tracks,t);
        ntimes=ntimes+1;
    end
end

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function retrieves the location of the platform

function [px,py]=get_platform_loc(pnum,time,platforms)

[nsamps,nplats]=size(platforms);
if (pnum> nplats) | (pnum <1)
    error('Incorrect platform number specified in get_platform_loc');
end
if (time<platforms(1,pnum).time)|(time>platforms(nsamps,pnum).time)
    error('Time out of range in get_platform_loc');
end

it=1;
while platforms(it,pnum).time < time
    it=it+1;
end
px=platforms(it,pnum).x;
py=platforms(it,pnum).y;

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function retrieves the location of the target

function [tx,ty]=get_target_loc(tnum,time,tracks)

[nsamps,ntracks]=size(tracks);
if (tnum> ntracks) | (tnum <1)
    error('Incorrect track number specified in get_target_loc');
end
if (time<tracks(1,tnum).time)|(time>tracks(nsamps,tnum).time)
    error('Time out of range in get_target_loc');
end

it=1;
while tracks(it,tnum).time < time
    it=it+1;
end
tx=tracks(it,tnum).x;
ty=tracks(it,tnum).y;

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function constructs a report for a sensor at a particular time

function report=sensor_report(sensor,platforms,tracks,time)

globals;

report=zeros(NUM_TARGETS,5);

% Find platform location at this time
[px,py]=get_platform_loc(sensor.platform,time,platforms);

for tnum=1:NUM_TARGETS
    % Find target location
    [tx,ty]=get_target_loc(tnum,time,tracks);
    % Compute range and bearing
    dx=tx-px;
    dy=ty-py;
    range=sqrt(dx*dx + dy*dy);
    bearing=atan2(dy,dx);
    % Determine if this is actually visible
    if range < sensor.max_range
        % Add error from sensor model
        report(tnum,1)=tnum;
        report(tnum,2)=range + sensor.r_err*(-1+2*rand);
        report(tnum,3)=bearing + sensor.b_err*(-1+2*rand);
        report(tnum,4)=px;
        report(tnum,5)=py;
    else
        report(tnum,1)=tnum;
        report(tnum,2)=NaN;
        report(tnum,3)=NaN;
        report(tnum,4)=NaN;
        report(tnum,5)=NaN;
    end
end
end

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function displays true target trajectories, platform trajectories,
% target destinations, and sensor observations following a simulation run

function
[true,plat,obs,dest]=post_sim(targets,observations,sensors,platforms,destinations)

globals;

% Set plotting size
[nsamps,nplatforms]=size(platforms);
[nsamps,ndestinations]=size(destinations);
[nsamps,ntargets]=size(targets);
[nsensors,nsamps]=size(observations);

figure(1)
clf;
hold on
data=zeros(2,nsamps);
title('True Target Motions')
xlabel('x-position (m)')
ylabel('y-position (m)')

% Plot true target trajectories
for n=1:ntargets
    for i=1:nsamps
        data(1,i)=targets(i,n).x;
        data(2,i)=targets(i,n).y;
    end
    true=plot(data(1,:),data(2,:));
end

% Plot true platform trajectories
for n=1:nplatforms
    for i=1:nsamps
        data(1,i)=platforms(i,n).x;
        data(2,i)=platforms(i,n).y;
    end
end

```

```

if n == 1
    plat=plot(data(1,:),data(2:),'ko');
elseif n == 2
    plat=plot(data(1,:),data(2:),'ko');
elseif n == 3
    plat=plot(data(1,:),data(2:),'ko');
elseif n == 4
    plat=plot(data(1,:),data(2:),'ko');
elseif n == 5
    plat=plot(data(1,:),data(2:),'ko');
elseif n == 6
    plat=plot(data(1,:),data(2:),'r*');
elseif n == 7
    plat=plot(data(1,:),data(2:),'r*');
elseif n == 8
    plat=plot(data(1,:),data(2:),'gh');
elseif n == 9
    plat=plot(data(1,:),data(2:),'bo');
end
end

% Plot true destination locations
for n=1:ndestinations
    for i=1:nsamps
        data(1,i)=destinations(i,n).x;
        data(2,i)=destinations(i,n).y;
    end
    dest=plot(data(1,:),data(2:),'rh');
end

% Plot sensor observations
for n=1:nsensors % for all sensors
    for i=1:nsamps % for all time
        report=observations(n,i).report;
        for m=1:ntargets % for all targets
            if report(m,1) > 0 % if they are seen
                [zx,zy,R]=xy_obs(report,m,n); % convert observation to global xy coordinates
                odata(1,m)=zx;
                odata(2,m)=zy;
            end
        end
    end
    if n == 1
        obs=plot(odata(1,:),odata(2:),'k. ');
    elseif n == 2
        obs=plot(odata(1,:),odata(2:),'k. ');
    end
end

```

```

elseif n == 3
    obs=plot(odata(1,:),odata(2:),'k.');
```

```
elseif n == 4
    obs=plot(odata(1,:),odata(2:),'k.');
```

```
elseif n == 5
    obs=plot(odata(1,:),odata(2:),'k.');
```

```
elseif n == 6
    obs=plot(odata(1,:),odata(2:),'r.');
```

```
elseif n == 7
    obs=plot(odata(1,:),odata(2:),'r.');
```

```
elseif n == 8
    obs=plot(odata(1,:),odata(2:),'g.');
```

```
elseif n == 9
    obs=plot(odata(1,:),odata(2:),'b.');
```

```
end
end
end

legend([true,plat,obs,dest],'Target      True      Position','Tracking      Stations','Target
Observations','Intended Target Destination')
hold off

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function extract an xy observation from a range-bearing report

function [zx,zy,R]=xy_obs(rep,n,sensor_id)

globals;

sr=sin(rep(n,3));
cr=cos(rep(n,3));
zx=rep(n,4)+rep(n,2)*cr;
zy=rep(n,5)+rep(n,2)*sr;
ROT=[cr -sr; sr cr];
range2=rep(n,2)*rep(n,2);
if sensor_id==1
    sigma=[SIGMA_RANGE_MPA^2 0;0 range2*SIGMA_BEARING_MPA^2];
elseif sensor_id==2
    sigma=[SIGMA_RANGE_MPA^2 0;0 range2*SIGMA_BEARING_MPA^2];
elseif sensor_id==3
    sigma=[SIGMA_RANGE_MPA^2 0;0 range2*SIGMA_BEARING_MPA^2];
elseif sensor_id==4
    sigma=[SIGMA_RANGE_MPA^2 0;0 range2*SIGMA_BEARING_MPA^2];
elseif sensor_id==5
    sigma=[SIGMA_RANGE_MPA^2 0;0 range2*SIGMA_BEARING_MPA^2];
elseif sensor_id==6
    sigma=[SIGMA_RANGE_PCG^2 0;0 range2*SIGMA_BEARING_PCG^2];
elseif sensor_id==7
    sigma=[SIGMA_RANGE_PCG^2 0;0 range2*SIGMA_BEARING_PCG^2];
elseif sensor_id==8
    sigma=[SIGMA_RANGE_PV^2 0;0 range2*SIGMA_BEARING_PV^2];
elseif sensor_id==9
    sigma=[SIGMA_RANGE_MPAC^2 0;0 range2*SIGMA_BEARING_MPAC^2];
end
R=ROT*sigma*ROT';

```



```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This file executes the filter algorithm

globals;
% Set up filter parameters
filter.Q= [DT^2*SIGMA_XDOT^2 0 0 0;
           0 SIGMA_XDOT^2 0 0;
           0 0 DT^2*SIGMA_YDOT^2 0;
           0 0 0 SIGMA_YDOT^2];
filter.H=[1 0 0 0; 0 0 1 0];
[nsensors,ntime]=size(observations);

% Use sensors
for i=1:nsensors
    filter.use_sensors(i)=1;
end

% Run filter
tracks=itracker(observations,sensors,platforms,filter,times);

% Plot the track trajectory
post_itracks;

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function generates tracks from a sequence of observations and
% assesses its threat to Jurong Island "real-time"

function tracks=itracker(observations,sensors,platforms,filter,times)

globals;

% Number of sensors and number of reports
[nsensors,ntimes]=size(observations);

% Search for first target first observation
target = 1;
for k = 1:nsensors
    for j =1:ntimes;
        if (~isnan(observations(k,j).report(target,2:5)))
            temp(j)=j;
        else
            temp(j)=99999;
        end
    end
    starttime(k) = min(temp);
    if (starttime(k) <= starttime(1))
        startstep = starttime(k);
    end
end

% Initialise the target
for j = 1
    tracks(startstep,j)=init_itracks(j,observations,filter,startstep);
end

% Initialise variables
for n = 1:ntimes
    kdata(1,n) = 0;
    kdata(2,n) = 0;
    kdata(3,n) = 0;
    kdata(4,n) = 0;

```

```

kdata(5,n) = 0;
kdata(6,n) = 0;
kdata(7,n) = 99999;
kdata(8,n) = 0;
kdata(9,n) = 0;
kdata(10,n) = 0;
kdata(11,n) = 0;
leftTSS(n) = NaN;
leftTSSntoJIHigh(n) = NaN;
leftTSSntoJIModerate(n) = NaN;
leftPL(n) = NaN;
leftPLntoJIHigh(n) = NaN;
leftPLntoJIModerate(n) = NaN;
end

% Initialise variables
distapart = 0;
distuncert = 0;
disttoeach = 0;
speedtoeach = 0;
timetoeach = 0;

for i = (startstep+1):ntimes
    startplot = 99999;
    buf = sprintf('Step: time=%d',i-1); disp(buf);
    % Do state prediction for the target
    for j=1
        tracks(i,j) = info_pred(tracks(i-1,j),times(i),filter);
    end

    % Do data association, returning an obs*track array of associations
    da_array = data_association(tracks,observations,i);

    % Finally, do update for the target
    for j=1
        tracks(i,j) = info_update(tracks(i,j),observations,filter,da_array,i,j);
    end

    warning off;
    Y = tracks(i,1).Yest;
    P = inv(Y);
    % Check if SAW has entered Port Limits
    if P ~= inf
        % Calculate the position uncertainty of the target at present location
        x = P*tracks(i,1).yest;
    end
end

```

```

ysigma = sqrt(P);
xuncert = ysigma(1,1);
yuncert = ysigma(3,3);
distuncert = sqrt(xuncert^2 + yuncert^2);
b = circle([x(1), x(3)],distuncert);

% Project the position uncertainty of the target
% at final destination
FF = [1 (ntimes-i)*DT 0 0;
      0 1 0 0;
      0 0 1 (ntimes-i)*DT;
      0 0 0 1];

GG = [1 0 0 0;
      0 1 0 0;
      0 0 1 0;
      0 0 0 1];

QQ = [((ntimes-i)*DT)^2*SIGMA_XDOT^2 0 0 0;
      0 SIGMA_XDOT^2 0 0;
      0 0 ((ntimes-i)*DT)^2*SIGMA_YDOT^2 0;
      0 0 0 SIGMA_YDOT^2];

MM = (inv(FF))*(tracks(i,1).Yest)*(inv(FF));
SSigma = GG'*MM*GG+inv(QQ);
OOmega = MM*GG*inv(SSigma);
projYpred = MM-(OOmega*SSigma*OOmega');
projYpred = force_sym(projYpred);
projP = inv(projYpred);
projPsigma = sqrt(projP);
projxuncert = projPsigma(1,1);
projyuncert = projPsigma(3,3);
projdistuncert = sqrt(projxuncert^2 + projyuncert^2);
projypred =(eye(4,4)-(OOmega*GG'))*(inv(FF))*tracks(i,1).yest;
projstate = projP*projypred;
warning on;

if min(inpolygon (b.X, b.Y, TSS_X, TSS_Y)) == 0 &&...
    min(inpolygon (b.X, b.Y, PL_X, PL_Y)) == 0
    buf=sprintf('Warning: Entered Port Limits at Time=%d',i-1);disp(buf);

% Check if SAW is turning towards Jurong Island
distoreach = sqrt(abs(x(1)-302.1144444)^2 +...
    abs(x(3)-347.7716667)^2);
speedtoreach = sqrt(x(2)^2+x(4)^2);

```

```

timetoreach = disttoreach/speedtoreach;
% Calculate the time target left Port Limits
leftPL(i) = MAX_TIME-timetoreach;

% Calculate the distance between the projected target end point
% and Jurong Island
distapart = sqrt((projstate(1) - 302.1144444)^2 + ...
    (projstate(3) - 347.7716667)^2);

% Threat level assessment
% Send warning if target is within 0.5nm to Jurong Island
if distapart < 5 && MAX_TIME - (i-1)*DT >= 0
    buf=sprintf('Warning: Heading Towards Jurong Island. Time to Impact = %.2f
sec',timetoreach); disp(buf);
    if projdistuncert > distapart
        buf=sprintf('Warning: High Confidence of Impact');disp(buf);
        startplot = i-1;
        leftPLntoJIHigh(i) = MAX_TIME-timetoreach;
    else
        buf=sprintf('Warning: Moderate Confidence of Impact');disp(buf);
        startplot = i-1;
        leftPLntoJIModerate(i) = MAX_TIME-timetoreach;
    end
end

% Check if SAW has left Traffic Separation Scheme
elseif min(inpolygon (b.X, b.Y, TSS_X, TSS_Y))== 0 &&...
    min(inpolygon (b.X, b.Y, PL_X, PL_Y)) == 1
    buf=sprintf('Warning: Left Traffic Separation Scheme at Time=%d',i-1);
disp(buf);

% Check if SAW is turning towards Jurong Island
disttoreach = sqrt(abs(x(1)-302.1144444)^2+...
    abs(x(3)-347.7716667)^2);
speedtoreach = sqrt(x(2)^2+x(4)^2);
timetoreach = disttoreach/speedtoreach;
% Calculate the time target left TSS
leftTSS(i) = MAX_TIME-timetoreach;

% Calculate the distance between the projected target
% end point and Jurong Island
distapart = sqrt((projstate(1) - 302.1144444)^2 + ...
    (projstate(3) - 347.7716667)^2);

% Threat level assessment

```

```

% Send warning if target is within 0.5nm to Jurong Island
if distapart < 5 && MAX_TIME - (i-1)*DT >= 0
    buf=sprintf('Warning: Heading Towards Jurong Island. Time to Impact = %.2f
sec',timetoreach); disp(buf);
    if projdistuncert > distapart
        buf=sprintf('Warning: High Confidence of Impact');disp(buf);
        startplot = i-1;
        leftTSSntoJIHigh(i) = MAX_TIME-timetoreach;
    else
        buf=sprintf('Warning: Moderate Confidence of Impact');disp(buf);
        startplot = i-1;
        leftTSSntoJIModerate(i) = MAX_TIME-timetoreach;
    end
end
end
end

```

```

kdata(1,i) = tracks(i,1).time;
kdata(2,i) = projdistuncert*180;
kdata(3,i) = distapart*180;
kdata(4,i) = disttoreach*180;
kdata(5,i) = speedtoreach*180;
kdata(6,i) = timetoreach;
kdata(7,i) = startplot;
kdata(8,i) = projP(1,1);
kdata(9,i) = projP(3,3);
kdata(10,i) = projstate(1);
kdata(11,i) = projstate(3);

```

```

end

```

```

% Future work for multi-targets
% Search for first observation for other targets
if NUM_TARGETS ~=1
    for target=2:NUM_TARGETS
        for k = 1:nsensors
            for j =1:ntimes;
                if (~isnan(observations(k,j).report(target,2:5)))
                    temp(j)=j;
                else
                    temp(j)=99999;
                end
            end
        end
        starttime(k) = min(temp);
        if (starttime(k) <= starttime(1))
            startstep = starttime(k);
        end
    end
end

```

```

        end
    end
end
end

% Initialisation
for j=2:NUM_TARGETS
    tracks(startstep,j)=init_itracks(j,observations,filter,startstep);
end

% Track
for i=(startstep+1):ntimes
    % Prediction for each target
    for j=2:NUM_TARGETS
        tracks(i,j)=info_pred(tracks(i-1,j),times(i),filter);
    end

    % Data association, returning an obs*track array of associations
    da_array=data_association(tracks,observations,i);

    % Update for each target
    for j=2:NUM_TARGETS
        tracks(i,j)=info_update(tracks(i,j),observations,filter,da_array,i,j);
    end
end
end

figure(20);
clf;
plot(kdata(1,:),(kdata(2,:)),'b',kdata(1,:),kdata(3,:),'g')
legend('Distance Uncertainty','Distance Apart');
buf=sprintf('Estimated End Point');
title(buf)
xlim([min(kdata(7,1000:MAX_TIME/DT))*DT MAX_TIME-1*DT])
axis 'auto y'
xlabel('Simulated Time (s)')
ylabel('Distance (m)')

figure(21);
clf;
plot(kdata(1,:),kdata(4,:),'b',kdata(1,:),kdata(5,:),'g',kdata(1,:),kdata(6,:),'r')
legend('Distance to Impact','Speed to Impact', 'Time to Impact');
buf=sprintf('Projected Distance, Speed and Time to Impact');
title(buf)
xlim([min(kdata(7,1000:MAX_TIME/DT))*DT MAX_TIME-1*DT])

```

```
axis 'auto y'  
xlabel('Simulated Time (s)')
```

```
% Record the times the target crossed TSS, PL, & corresponding threat level  
firstleftTSS = min(leftTSS(1000:MAX_TIME/DT));  
firstleftTSSntoJIHigh = min(leftTSSntoJIHigh(1000:MAX_TIME/DT));  
firstleftTSSntoJIModerate = min(leftTSSntoJIModerate(1000:MAX_TIME/DT));  
firstleftPL = min(leftPL(1000:MAX_TIME/DT));  
firstleftPLntoJIHigh = min(leftPLntoJIHigh(1000:MAX_TIME/DT));  
firstleftPLntoJIModerate = min(leftPLntoJIModerate(1000:MAX_TIME/DT));
```



```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function initialises a set of tracks from the first few observations

function track=init_itracks(ntarget,observations,filter,startstep)

globals;

% Straight-forward in information form, now initialise the track
ypred=zeros(XSIZE,1);    %initial condition
yest=ypred;
Ypred=zeros(XSIZE,XSIZE); %initial condition
Yest=Ypred;

% put together data structure
track=make_itrack(ntarget,1,startstep,ypred,yest,Ypred,Yest);
function track=make_track(id,type,time,ypred,yest,Ypred,Yest)

track=struct('id',id,'type',type,'time',time,'ypred',ypred,'yest',yest,'Ypred',Ypred,'Yest',Yest
);

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function performs prediction for a CVM

function track_pred=info_pred(track,time,filter)

dt = time-track.time;
if dt < 0
    error('Negative prediction step in state_pred')
end

% Compute transition matrices
F=[1 dt 0 0;
   0 1 0 0;
   0 0 1 dt;
   0 0 0 1];
G=[1 0 0 0;
   0 1 0 0;
   0 0 1 0;
   0 0 0 1];
Q=filter.Q;

% Do prediction
M=(inv(F))*(track.Yest)*(inv(F));
Sigma=G'*M*G+inv(Q);
Omega=M*G*inv(Sigma); % here is the problem
Ypred=M-(Omega*Sigma*Omega');
ypred=(eye(4,4)-(Omega*G'))*(inv(F))*(track.yest);
Ypred=force_sym(Ypred); % seems to require for stability in Matlab

% Make space for later
Yest=Ypred;
yest=ypred;

% Construct the new track
track_pred=make_itrack(track.id,1,time,ypred,yest,Ypred,Yest);
function sigma=force_sym(sigma)
%
% a function that forces symmetry on a matrix

```

```
%  
  
[nx,ny]=size(sigma);  
if nx ~= ny  
    error('Matrix must be square');  
end  
  
for i=1:nx  
    for j=i+1:nx  
        temp=(sigma(i,j)+sigma(j,i))/2;  
        sigma(i,j)=temp;  
        sigma(j,i)=temp;  
    end  
end
```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function sets up the data association array

function da=data_association(tracks,observations,time)

% This is an array of size ntargets to nsensors
% For each sensor i, the entry states which sensor report
% is associated with a target j
[nsensors,ntimes]=size(observations);
[ntimes,ntargets]=size(tracks);
da=zeros(ntargets,nsensors);

% For each sensor
for i=1:nsensors
    % Associate a track with a report
    for j=1:ntargets
        da(j,i)=j;
    end
end
end

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function performs information filter update

function track=info_update(track,observations,filter,da_array,ntime,ntarget)

globals;

% Initialise to prediction
yest=track.ypred;
Yest=track.Ypred;

% Observation model
H=filter.H;

% Get new information
[nsensors,temp]=size(observations);
for i=1:nsensors
    if filter.use_sensors(i) == 1
        % Check for valid observations first
        if (~isnan(observations(i,ntime).report(ntarget,2:5)))
            if (observations(i,ntime).report(ntarget,2:5)~=999999)
                % Extract observation
                rep=observations(i,ntime).report;
                na=da_array(ntarget,i);
                [zx,zy,R]=xy_obs(rep,na,i); % R is unused
                z=[zx;zy];
                if i==1
                    Ri=inv(Rfilter_MPA);
                elseif i==2
                    Ri=inv(Rfilter_MPA);
                elseif i==3
                    Ri=inv(Rfilter_MPA);
                elseif i==4
                    Ri=inv(Rfilter_MPA);
                elseif i==5
                    Ri=inv(Rfilter_MPA);
                elseif i==6
                    Ri=inv(Rfilter_PCG);

```

```

elseif i==7
    Ri=inv(Rfilter_PCG);
elseif i==8
    Ri=inv(Rfilter_PV);
elseif i==9
    Ri=inv(Rfilter_MPAC);
end
% Construct information terms
info=H'*Ri*z;
Info=H'*Ri*H;
% Add the information to prediction
yest=yest+info;
Yest=Yest+Info;
% Seems to require for stability in Matlab
Yest=force_sym(Yest);
elseif (observations(i,ntime).report(ntarget,2:5)== 99999)
    yest=yest;
    Yest=Yest;
    % Seems to require for stability in Matlab
    Yest=force_sym(Yest);
end
end
end
end

% Do assignment
track.yest=yest;
track.Yest=Yest;

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This file displays the fused target track following a simulation run

globals;

% Set plotting size
[nsamps,ntargets]=size(tracks);
figure(1)
xlim([150 750])
ylim([280 440])
hold on
warning off;
data=zeros(2,nsamps);

j =1;
while (isempty(tracks(j,1).ypred(:)))
    j = j+1;
end
trackstarttime = j;

for n=1:ntargets
    for i=trackstarttime:nsamps
        Y=tracks(i,n).Yest;
        P=inv(Y);
        x=P*tracks(i,n).yest;
        data(1,i)=x(1);
        data(2,i)=x(3);
    end
    estip=plot(data(1,trackstarttime:nsamps),data(2,trackstarttime:nsamps),'g-');
end

legend([estip], 'Estimated Track Position')
hold off

plot_errors(tracks, ntargets, 2, trackstarttime, targets)

warning on;

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function produces various plots with regards to the SAW parameters

function plot_errors(tracks,ntarget,nfigure,trackstarttime,targets)

globals;

% Initialise
[nsamps,ntargets]=size(tracks);
ydata=zeros(10,nsamps);
pdata=zeros(10,nsamps); %predicted data
edata=zeros(10,nsamps); %estimated data
tdata=zeros(10,nsamps); %true data
esttimesigma=zeros(1,nsamps);
xmin = 4500;
xmax = MAX_TIME;

warning off;
% Calculating and storing the data for each time-step
for i=trackstarttime:nsamps

    Y=tracks(i,ntarget).Yest;
    P=inv(Y);
    x=P*tracks(i,ntarget).yest;

    Yp=tracks(i,ntarget).Ypred;
    Pp=inv(Yp);
    xp=Pp*tracks(i,ntarget).ypred;

    ydata(1,i)=tracks(i,ntarget).time;
    ydata(2,i)=(x(1)-targets(i,ntarget).x)*180;
    ydata(3,i)=(x(3)-targets(i,ntarget).y)*180;
    ysigma=real(sqrt(P));
    ydata(4,i)=ysigma(1,1)*180;
    ydata(5,i)=ysigma(3,3)*180;
    ydata(6,i)=(real(sqrt(x(2)*x(2)+x(4)*x(4)))-targets(i,ntarget).vel)*180;
    ydata(7,i)=atan2(x(4),x(2))-targets(i,ntarget).phi;
    if abs(ydata(7,i)) > pi

```



```

        ydata(7,i) = 2*pi-abs(ydata(7,i));
    end
    ydata(8,i)=sqrt(ysigma(1,1)*ysigma(1,1)+ysigma(3,3)*ysigma(3,3))*180;
    ydata(9,i)=sqrt(ysigma(2,2)*ysigma(2,2)+ysigma(4,4)*ysigma(4,4))*180;
    ydata(10,i)=real(sqrt(((x(2)/(x(2)^2 + x(4)^2))^2)*(ysigma(4,4)^2)...
        +((x(4)/(x(2)^2 + x(4)^2))^2)*(ysigma(2,2)^2)));

    tdata(1,i)=tracks(i,ntarget).time;
    tdata(2,i)=targets(i,ntarget).x*180;
    tdata(3,i)=targets(i,ntarget).y*180;
    tdata(6,i)=targets(i,ntarget).vel*180;
    tdata(7,i)=targets(i,ntarget).phi;

    pdata(1,i)=tracks(i,ntarget).time;
    pdata(2,i)=xp(1)*180;
    pdata(3,i)=xp(3)*180;
    psigma=real(sqrt(Pp));
    pdata(4,i)=psigma(1,1)*180;
    pdata(5,i)=psigma(3,3)*180;
    pdata(6,i)=(real(sqrt(xp(2)*xp(2)+xp(4)*xp(4))))*180;
    pdata(7,i)=atan2(xp(4),xp(2));
    if abs(pdata(7,i)) > pi
        pdata(7,i) = 2*pi-abs(pdata(7,i));
    end
    pdata(8,i)=sqrt(psigma(1,1)*psigma(1,1)+psigma(3,3)*psigma(3,3))*180;
    pdata(9,i)=sqrt(psigma(2,2)*psigma(2,2)+psigma(4,4)*psigma(4,4))*180;
    pdata(10,i)=real(sqrt(((xp(2)/(xp(2)^2 + xp(4)^2))^2)*(psigma(4,4)^2)...
        +((xp(4)/(xp(2)^2 + xp(4)^2))^2)*(psigma(2,2)^2)));

    edata(1,i)=tracks(i,ntarget).time;
    edata(2,i)=x(1)*180;
    edata(3,i)=x(3)*180;
    esigma=real(sqrt(P));
    edata(4,i)=esigma(1,1)*180;
    edata(5,i)=esigma(3,3)*180;
    edata(6,i)=(real(sqrt(x(2)*x(2)+x(4)*x(4))))*180;
    edata(7,i)=atan2(x(4),x(2));
    if abs(edata(7,i)) > pi
        edata(7,i) = 2*pi-abs(edata(7,i));
    end
    edata(8,i)=sqrt(esigma(1,1)*esigma(1,1)+esigma(3,3)*esigma(3,3))*180;
    edata(9,i)=sqrt(esigma(2,2)*esigma(2,2)+ysigma(4,4)*esigma(4,4))*180;
    edata(10,i)=real(sqrt(((x(2)/(x(2)^2 + x(4)^2))^2)*(esigma(4,4)^2)...
        +((x(4)/(x(2)^2 + x(4)^2))^2)*(esigma(2,2)^2)));

```

```

    esttimesigma(1,i)=sqrt(edata(8,i)^2+...
        (kdata(6,i)^2)*(edata(9,i)^2))/...
        edata(6,i);
end

% merging the heading standard deviations for plotting
p=trackstarttime;q=trackstarttime;i=trackstarttime;
for j=1:(nsamps-trackstarttime+1)*2
    if j/2-round(j/2)==0
        combtimedata(1,j)= tracks(i,ntarget).time;
        combddata(8,j)=edata(8,p);
        combddata(9,j)=edata(9,p);
        combddata(10,j)=edata(10,p);
        p=p+1;
        i=i+1;
    else
        combtimedata(1,j)= tracks(i,ntarget).time;
        combddata(8,j)=pdata(8,q);
        combddata(9,j)=pdata(9,q);
        combddata(10,j)=pdata(10,q);
        q=q+1;
    end
end

% Plot
figure(nfigure);
clf;
plot(ydata(1,:),abs(ydata(2,:)),'b',ydata(1,:),ydata(4,:),'r')
legend('Estimated Error','Standard Deviation');
buf=sprintf('Error between X-component Estimate and Truth');
title(buf)
xlim([xmin xmax])
axis 'auto y'
xlabel('Simulated Time (s)')
ylabel('X error (m)')

figure(nfigure+1);
clf;
plot(ydata(1,:),abs(ydata(3,:)),'b',ydata(1,:),ydata(5,:),'r')
legend('Estimated Error','Standard Deviation');
buf=sprintf('Error between Y-component Estimate and Truth');
title(buf)
xlim([xmin xmax])
axis 'auto y'
xlabel('Simulated Time (s)')

```

```

ylabel('Y error (m)')

figure(nfigure+2);
clf;
plot(ydata(1,:),abs(ydata(6,:)), 'b', ydata(1,:), ydata(9,:), 'r')
legend('Estimated Error', 'Standard Deviation');
buf=sprintf('Error between Velocity Estimate and Truth');
title(buf)
xlim([xmin xmax])
axis 'auto y'
xlabel('Simulated Time (s)')
ylabel('Velocity error (m/s)')

figure(nfigure+3);
clf;
plot(ydata(1,:),abs(ydata(7,:)), 'b', ydata(1,:), ydata(10,:), 'r')
legend('Estimated Error', 'Standard Deviation');
buf=sprintf('Error between Heading Estimate and Truth');
title(buf)
xlim([xmin xmax])
axis 'auto y'
xlabel('Simulated Time (s)')
ylabel('Heading error (rads)')

figure(nfigure+4);
clf;
plot(pdata(1,:),pdata(2,:), 'r', edata(1,:), edata(2,:), 'g', tdata(1,:), tdata(2,:), 'b')
legend('Predicted X', 'Estimated X', 'True X');
buf=sprintf('X position');
title(buf)
xlim([xmin xmax])
axis 'auto y'
xlabel('Simulated Time (s)')
ylabel('X Position (m)')

figure(nfigure+5);
clf;
plot(pdata(1,:),pdata(3,:), 'r', edata(1,:), edata(3,:), 'g', tdata(1,:), tdata(3,:), 'b')
legend('Predicted Y', 'Estimated Y', 'True Y');
buf=sprintf('Y position');
title(buf)
xlim([xmin xmax])
axis 'auto y'
xlabel('Simulated Time (s)')
ylabel('Y Position (m)')

```

```

figure(nfigure+6);
clf;
plot(pdata(1,:),pdata(6,:), 'r', edata(1,:), edata(6,:), 'g', tdata(1,:), tdata(6,:), 'b')
legend('Predicted Velocity', 'Estimated Velocity', 'True Velocity');
buf=sprintf('Velocity');
title(buf)
xlim([xmin xmax])
axis 'auto y'
xlabel('Simulated Time (s)')
ylabel('Velocity (m/s)')

figure(nfigure+7);
clf;
plot(pdata(1,:),pdata(7,:), 'r', edata(1,:), edata(7,:), 'g', tdata(1,:), tdata(7,:), 'b')
legend('Predicted Heading', 'Estimated Heading', 'True Heading');
buf=sprintf('Heading');
title(buf)
xlim([xmin xmax])
axis 'auto y'
xlabel('Simulated Time (s)')
ylabel('Heading (rad)')

figure(nfigure+8);
clf;
plot(pdata(1,:),pdata(8,:), 'r+', edata(1,:), edata(8,:), 'g*', combtimedata(1,:), combdata(8,:), 'b')
legend('Predicted', 'Estimated');
buf=sprintf('Radius Standard Deviations');
title(buf)
xlim([xmin xmax])
axis 'auto y'
xlabel('Simulated Time (s)')
ylabel('Radius (m)')

figure(nfigure+9);
clf;
plot(pdata(1,:),pdata(9,:), 'r+', edata(1,:), edata(9,:), 'g*', combtimedata(1,:), combdata(9,:), 'b')
legend('Predicted', 'Estimated');
buf=sprintf('Velocity Standard Deviations');
title(buf)
xlim([xmin xmax])
axis 'auto y'
xlabel('Simulated Time (s)')
ylabel('Velocity (m/s)')

```

```

figure(nfigure+10);
clf;
plot(pdata(1,:),pdata(10,:), 'r+', edata(1,:), edata(10,:), 'g*', combtimedata(1,:), combdata(10,:), 'b')
legend('Predicted', 'Estimated');
buf=sprintf('Heading Standard Deviations');
title(buf)
xlim([xmin xmax])
axis 'auto y'
xlabel('Simulated Time (s)')
ylabel('Heading (rad)')

figure(nfigure+11);
clf;
plot(edata(1,:), abs((MAX_TIME-edata(1,:))-kdata(6,:)), 'b', edata(1,:), esttimesigma(1,:), 'r')
legend('Estimated Error', 'Standard Deviation');
buf=sprintf('Error between Time Estimate and Truth');
title(buf)
xlim([min(kdata(7,1000:MAX_TIME/10))*DT xmax-1*DT])
axis 'auto y'
xlabel('Simulated Time (s)')
ylabel('Time (s)')

warning on;

```

C. BLOCK 3. DECENTRALIZED DATA FUSION ARCHITECTURE USING INFORMATION FILTER

% This file executes the Monte-Carlo simulation

% Number of Monte-Carlo simulation runs

totalruns = 50;

% Initialise variables

sumfirstleftTSS(1:totalruns) = NaN;

sumfirstleftTSSntoJIHigh(1:totalruns) = NaN;

sumfirstleftTSSntoJIModerate(1:totalruns) = NaN;

sumfirstleftPL(1:totalruns) = NaN;

sumfirstleftPLntoJIHigh(1:totalruns) = NaN;

sumfirstleftPLntoJIModerate(1:totalruns) = NaN;

% Execute each run by generating new observations or using previous ones

% Produce the fused track for each run

% Then pool the tracks together for analysis

for run = 1:totalruns

buf=sprintf('Run: =%d',run); disp(buf);

example_sim; %generate new observations

observations = datastore5(run).observations; % using previous observations

NEW_RUN = 0;

run_net;

sumfirstleftTSS(run) = firstleftTSS;

sumfirstleftTSSntoJIHigh(run) = firstleftTSSntoJIHigh;

sumfirstleftTSSntoJIModerate(run) = firstleftTSSntoJIModerate;

sumfirstleftPL(run) = firstleftPL;

sumfirstleftPLntoJIHigh(run) = firstleftPLntoJIHigh;

sumfirstleftPLntoJIModerate(run) = firstleftPLntoJIModerate;

sumkdata(run,,:) = kdata;

sumesttimesigma(run,1,:) = esttimesigma;

lowest(run) = min(sumkdata(run,7,1000:MAX_TIME/10));

end

% Calculate Probability of Impact

Cxtemp = squeeze(sumkdata(:,8,:));

Cytemp = squeeze(sumkdata(:,9,:));

Xtemp = squeeze(sumkdata(:,10,:));

Ytemp = squeeze(sumkdata(:,11,:));

probability = zeros(1,nt);

warning off;

for l = 1:nt

```

Cx = diag(Cxtemp(:,l))*(180^2);
Cy = diag(Cytemp(:,l))*(180^2);
X = (Xtemp(:,l)- 302.1144444)*180;
Y = (Ytemp(:,l)- 347.7716667)*180;
W = ones(totalruns,1);
sigma_xbar_tgo = inv((W')*inv(Cx)*W);
sigma_ybar_tgo = inv((W')*inv(Cy)*W);
sigma = sqrt(sigma_xbar_tgo);
xbar_tgo = sigma_xbar_tgo*(((W')*inv(Cx)*X)); % Calculate mean impact pt
ybar_tgo = sigma_ybar_tgo*(((W')*inv(Cy)*Y)); % Calculate mean impact pt
r_not = sqrt(xbar_tgo^2 + ybar_tgo^2);
R = 300;

g = zeros(1,100);
h = zeros(1,100);
probttemp = 0;

recur = 1;
g_not = exp(-(r_not^2)/(2*sigma^2)); % g_not
h_not = 1 - exp(-(R^2)/(2*sigma^2)); % h_not
probttemp_not = g_not*h_not;

g(1) = (1/1)*((r_not^2)/(2*sigma^2))*g_not;
h(1) = -(1/factorial(1))*(((R^2)/(2*sigma^2))^(1))*exp(-(R^2)/(2*sigma^2)) + h_not;
probttemp = probttemp_not + g(1)*h(1);

while (recur <= 100)
    g(recur+1) = (1/(recur+1))*((r_not^2)/(2*sigma^2))*g(recur);
    h(recur+1) = (-1/factorial(recur+1))*(((R^2)/(2*sigma^2))^(recur+1))*exp(-(
(R^2)/(2*sigma^2)) + h(recur);
    probttemp = probttemp + g(recur+1)*h(recur+1);
    recur = recur + 1;
end

probability(1,1) = probttemp;
end

% Initialise variables
count1 = 0;
count2 = 0;
count3 = 0;
count4 = 0;
count5 = 0;
count6 = 0;

```

```

SsumfirstleftTSS = 0;
SsumfirstleftTSSntoJIHigh = 0;
SsumfirstleftTSSntoJIModerate = 0;
SsumfirstleftPL = 0;
SsumfirstleftPLntoJIHigh = 0;
SsumfirstleftPLntoJIModerate = 0;

% Take the average of the pooled tracks
for m = 1:totalruns
    if ~isnan(sumfirstleftTSS(m))
        count1 = count1+1;
        SsumfirstleftTSS=SsumfirstleftTSS+sumfirstleftTSS(m);
    end

    if ~isnan(sumfirstleftTSSntoJIHigh(m))
        count2 = count2+1;

SsumfirstleftTSSntoJIHigh=SsumfirstleftTSSntoJIHigh+sumfirstleftTSSntoJIHigh(m);
    end

    if ~isnan(sumfirstleftTSSntoJIModerate(m))
        count3 = count3+1;

SsumfirstleftTSSntoJIModerate=SsumfirstleftTSSntoJIModerate+sumfirstleftTSSntoJIModerate(m);
    end

    if ~isnan(sumfirstleftPL(m))
        count4 = count4+1;
        SsumfirstleftPL=SsumfirstleftPL+sumfirstleftPL(m);
    end

    if ~isnan(sumfirstleftPLntoJIHigh(m))
        count5 = count5+1;
        SsumfirstleftPLntoJIHigh=SsumfirstleftPLntoJIHigh+sumfirstleftPLntoJIHigh(m);
    end

    if ~isnan(sumfirstleftPLntoJIModerate(m))
        count6 = count6+1;

SsumfirstleftPLntoJIModerate=SsumfirstleftPLntoJIModerate+sumfirstleftPLntoJIModerate(m);
    end
end

```



```

meanfirstleftTSS = SsumfirstleftTSS/count1;
meanfirstleftTSSntoJIHigh = SsumfirstleftTSSntoJIHigh/count2;
meanfirstleftTSSntoJIModerate = SsumfirstleftTSSntoJIModerate/count3;
meanfirstleftPL = SsumfirstleftPL/count4;
meanfirstleftPLntoJIHigh = SsumfirstleftPLntoJIHigh/count5;
meanfirstleftPLntoJIModerate = SsumfirstleftPLntoJIModerate/count6;
meantruetimedata = mean(sumkdata(:,1,:));
meandistuncert = mean(sumkdata(:,2,:));
meandistapart = mean(sumkdata(:,3,:));
meanesttimedata = mean(sumkdata(:,6,:));

```

% Calculate the sample variance of the estimated time to impact

```

for p = 1:nt
    sumtimediff=0;
    timediff=0;
    for q = 1:totalruns
        timediff = (sumkdata(q,6,p)-meanesttimedata(p))^2;
        sumtimediff=sumtimediff+timediff;
    end
    meansampletimesigma(p) = sqrt(sumtimediff /(totalruns-1));
end

```

```

figure(30);
clf;
plot(meantruetimedata(:)',meandistuncert(:)','b',meantruetimedata(:)',meandistapart(:)','g')
legend('Distance Uncertainty','Distance Apart');
buf=sprintf('Estimated End Point');
title(buf)
xlim([min(lowest)*DT MAX_TIME-1*DT])
ylim([0 1000])
xlabel('Simulated Time (s)')
ylabel('Distance (m)')

```

```

figure(31);
clf;
plot(meantruetimedata(:)',abs((MAX_TIME-meantruetimedata(:))-
meanesttimedata(:))','b',meantruetimedata(:)',meansampletimesigma,'r' )
legend('Estimated Error','Standard Deviation');
buf=sprintf('Error between Time Estimate and Truth');
title(buf)
xlim([min(lowest)*DT MAX_TIME-1*DT])
ylim([0 100])
xlabel('Simulated Time (s)')
ylabel('Time (s)')

```

```

figure(32);
clf;
plot(meantruetimedata(:)',probability(:),'k')
legend('Probability of Impact');
buf=sprintf('Probability of Impact');
title(buf)
xlim([min(lowest)*DT MAX_TIME-1*DT])
ylim([0 1])
xlabel('Simulated Time (s)')
ylabel('Probability of Impact')

```

This file executes the Monte-Carlo simulation and incorporates dataloss
(i.e. observations loss) due to communication bandwidth limitations

% Number of Monte-Carlo simulation runs

totalruns = 50;

% Initialise variables

sumfirstleftTSS(1:totalruns) = NaN;

sumfirstleftTSSntoJIHigh(1:totalruns) = NaN;

sumfirstleftTSSntoJIModerate(1:totalruns) = NaN;

sumfirstleftPL(1:totalruns) = NaN;

sumfirstleftPLntoJIHigh(1:totalruns) = NaN;

sumfirstleftPLntoJIModerate(1:totalruns) = NaN;

% Execute each run by generating new observations or using previous ones

% Inject data loss

% Produce the fused track for each run

% Then pool the tracks together for analysis

for run = 1:totalruns

 buf=sprintf('Run: =%d',run); disp(buf);

 example_sim; %generate new observations

 %observations = datastoreloss(run).observations; % using previous observations

% Since previous observations (with loss) is used, there is no need to

% re-inject data loss. If new observations (with no loss) are made, the

% following codes are necessary to inject data loss.

%-----

 % simulate data loss due to bandwidth limitations

 for i = 1:10

 j = randi(MAX_TIME/DT-1060-3)+1060;

 datalossduration = randi(3);

 for k = 1:datalossduration

 for n = 1:NUM_PLATFORMS

 observations(n,j+k).report(2:5) = 99999;

 end

 end

 end

 datastoreloss(run).observations=observations;

%-----

NEW_RUN = 0;

run_net;

sumfirstleftTSS(run) = firstleftTSS;

```

sumfirstleftTSSntoJIHigh(run) = firstleftTSSntoJIHigh;
sumfirstleftTSSntoJIModerate(run) = firstleftTSSntoJIModerate;
sumfirstleftPL(run) = firstleftPL;
sumfirstleftPLntoJIHigh(run) = firstleftPLntoJIHigh;
sumfirstleftPLntoJIModerate(run) = firstleftPLntoJIModerate;
sumkdata(run, :, :) = kdata;
sumesttimesigma(run, 1, :) = esttimesigma;
lowest(run) = min(sumkdata(run, 7, 1000:MAX_TIME/10));
end

% Calculate Probability of Impact
Cxtemp = squeeze(sumkdata(:, 8, :));
Cytemp = squeeze(sumkdata(:, 9, :));
Xtemp = squeeze(sumkdata(:, 10, :));
Ytemp = squeeze(sumkdata(:, 11, :));
probability = zeros(1, nt);

warning off;
for l = 1:nt

    Cx = diag(Cxtemp(:, l)) * (180^2);
    Cy = diag(Cytemp(:, l)) * (180^2);
    X = (Xtemp(:, l) - 302.1144444) * 180;
    Y = (Ytemp(:, l) - 347.7716667) * 180;
    W = ones(totalruns, 1);
    sigma_xbar_tgo = inv((W') * inv(Cx) * W);
    sigma_ybar_tgo = inv((W') * inv(Cy) * W);
    sigma = sqrt(sigma_xbar_tgo);
    xbar_tgo = sigma_xbar_tgo * (((W') * inv(Cx) * X)); % Calculate mean impact pt
    ybar_tgo = sigma_ybar_tgo * (((W') * inv(Cy) * Y)); % Calculate mean impact pt
    r_not = sqrt(xbar_tgo^2 + ybar_tgo^2);
    R = 300;

    g = zeros(1, 100);
    h = zeros(1, 100);
    probtemp = 0;

    recur = 1;
    g_not = exp(-(r_not^2)/(2*sigma^2)); % g_not
    h_not = 1 - exp(-(R^2)/(2*sigma^2)); % h_not
    probtemp_not = g_not * h_not;

    g(1) = (1/1) * ((r_not^2)/(2*sigma^2)) * g_not;
    h(1) = -(1/factorial(1)) * (((R^2)/(2*sigma^2))^(1)) * exp(-(R^2)/(2*sigma^2)) + h_not;
    probtemp = probtemp_not + g(1) * h(1);

```

```

while (recur <= 100)
    g(recur+1) = (1/(recur+1))*((r_not^2)/(2*sigma^2))*g(recur);
    h(recur+1) = (-1/factorial(recur+1))*(((R^2)/(2*sigma^2))^(recur+1))*exp(-(R^2)/(2*sigma^2)) + h(recur);
    probtemp = probtemp + g(recur+1)*h(recur+1);
    recur = recur + 1;
end

probability(1,l) = probtemp;
end

% Initialise variables
count1 = 0;
count2 = 0;
count3 = 0;
count4 = 0;
count5 = 0;
count6 = 0;
SsumfirstleftTSS = 0;
SsumfirstleftTSSntoJIHigh = 0;
SsumfirstleftTSSntoJIModerate = 0;
SsumfirstleftPL = 0;
SsumfirstleftPLntoJIHigh = 0;
SsumfirstleftPLntoJIModerate = 0;

% Take the average of the pooled tracks
for m = 1:totalruns
    if ~isnan(sumfirstleftTSS(m))
        count1 = count1+1;
        SsumfirstleftTSS=SsumfirstleftTSS+sumfirstleftTSS(m);
    end

    if ~isnan(sumfirstleftTSSntoJIHigh(m))
        count2 = count2+1;

SsumfirstleftTSSntoJIHigh=SsumfirstleftTSSntoJIHigh+sumfirstleftTSSntoJIHigh(m);
    end

    if ~isnan(sumfirstleftTSSntoJIModerate(m))
        count3 = count3+1;

SsumfirstleftTSSntoJIModerate=SsumfirstleftTSSntoJIModerate+sumfirstleftTSSntoJIModerate(m);
    end
end

```

```

if ~isnan(sumfirstleftPL(m))
    count4 = count4+1;
    SsumfirstleftPL=SsumfirstleftPL+sumfirstleftPL(m);
end

if ~isnan(sumfirstleftPLntoJIHigh(m))
    count5 = count5+1;
    SsumfirstleftPLntoJIHigh=SsumfirstleftPLntoJIHigh+sumfirstleftPLntoJIHigh(m);
end

if ~isnan(sumfirstleftPLntoJIModerate(m))
    count6 = count6+1;

SsumfirstleftPLntoJIModerate=SsumfirstleftPLntoJIModerate+sumfirstleftPLntoJIModerate(m);
end
end

meanfirstleftTSS = SsumfirstleftTSS/count1;
meanfirstleftTSSntoJIHigh = SsumfirstleftTSSntoJIHigh/count2;
meanfirstleftTSSntoJIModerate = SsumfirstleftTSSntoJIModerate/count3;
meanfirstleftPL = SsumfirstleftPL/count4;
meanfirstleftPLntoJIHigh = SsumfirstleftPLntoJIHigh/count5;
meanfirstleftPLntoJIModerate = SsumfirstleftPLntoJIModerate/count6;
meantruetimedata = mean(sumkdata(:,1,:));
meandistuncert = mean(sumkdata(:,2,:));
meandistapart = mean(sumkdata(:,3,:));
meanesttimedata = mean(sumkdata(:,6,:));

% Calculate the sample variance of the estimated time to impact
for p = 1:nt
    sumtimediff=0;
    timediff=0;
    for q = 1:totalruns
        timediff = (sumkdata(q,6,p)-meanesttimedata(p))^2;
        sumtimediff=sumtimediff+timediff;
    end
    meansampletimesigma(p) = sqrt(sumtimediff /(totalruns-1));
end

figure(30);
clf;
plot(meantruetimedata(:)',meandistuncert(:)','b',meantruetimedata(:)',meandistapart(:)','g')
legend('Distance Uncertainty','Distance Apart');

```

```

buf=sprintf('Estimated End Point');
title(buf)
xlim([min(lowest)*DT MAX_TIME-1*DT])
ylim([0 1000])
xlabel('Simulated Time (s)')
ylabel('Distance (m)')

figure(31);
clf;
plot(meantruetimedata(:)',abs((MAX_TIME-meantruetimedata(:))-
meanesttimedata(:)'),'b',meantruetimedata(:)',meansampletimesigma,'r' )
legend('Estimated Error','Standard Deviation');
buf=sprintf('Error between Time Estimate and Truth');
title(buf)
xlim([min(lowest)*DT MAX_TIME-1*DT])
ylim([0 100])
xlabel('Simulated Time (s)')
ylabel('Time (s)')

figure(32);
clf;
plot(meantruetimedata(:)',probability(:),'k')
legend('Probability of Impact');
buf=sprintf('Probability of Impact');
title(buf)
xlim([min(lowest)*DT MAX_TIME-1*DT])
ylim([0 1])
xlabel('Simulated Time (s)')
ylabel('Probability of Impact')

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This file runs the decentralised tracker. The default condition is to
% do a new run every time. There is also the option to use previous
% observation runs so that different filters can be compared.

globals;
ginit;

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

% If we choose to use previous data,
% then the existence of various variables needs to be checked
if ~NEW_RUN
    if ~exist('observations','var') | ~exist('targets','var') |...
        ~exist('platforms','var') | ~exist('times','var')
        error('No existing Data to perform simulation');
    end
    [num_nodes,num_times]=size(observations);
    nt=length(times);
    if (num_nodes ~= NUM_NODES) | (num_times ~= nt)
        error('Observation Record is of incorrect dimension');
    end
    if nt~= length(targets)
        error('Target set is of incorrect dimension');
    end
    [num_times,num_plat]=size(platforms);
    if (num_nodes ~= num_plat) | (num_times ~= nt)
        error('Platform set is of incorrect dimension');
    end
    buf=sprintf('Old Run Commencing with %d nodes and %d time
steps\n',num_nodes,num_times);
    disp(buf);
end

% If we are doing a new run then simulate the target and platform motions
if NEW_RUN

```



```

clear all;
globals;
ginit;
NEW_RUN=1;
times=0:DT:MAX_TIME;
num_times=length(times);
buf=sprintf('New Run Commencing with %d nodes and %d time
steps\n',NUM_NODES,num_times-1);
disp(buf);

% This code currently apply to only one target
% One target considerably simplifies computation
buf=sprintf('Generating Target Set\n'); disp(buf);
targets=generate__targets(times);

% Simulate platforms motion, assuming they are known
buf=sprintf('Generating Platform Trajectories\n'); disp(buf);
platforms=generate__platforms(times);

% Generate all target's attack destinations
destinations=generate__destinations(times);
else
% If we are using previous data set
% then there is no need to simulate targets or platforms
ginit;
clear Nodes;
end

% In every case, the sensors which sit on the platforms are defined
% Also, this allows the filters to be changed between runs
buf=sprintf('Defining Sensors\n'); disp(buf);
filter=get_filter_params;

for i=1:NUM_NODES
    Nodes(i)=make__sensor(i,filter);
end

% Establish initial topology
com_net=init_net(platforms,Nodes);

% Establish the time base
ntimes=length(times);

% Define space to record results
Results_y=zeros(ntimes,NUM_NODES,filter.XDIM);

```

```
Results_Y=zeros(ntimes,NUM_NODES,filter.XDIM,filter.XDIM);
```

```
%-----  
%  
% START OF FILTER LOOPS HERE  
%  
%-----
```

```
% Run loop for all sensors at each time.
```

```
% This is as close as we can get to running this in parallel
```

```
for n=1:ntimes  
    kdata(1,n)=0;  
    kdata(2,n)=0;  
    kdata(3,n)=0;  
    kdata(4,n)=0;  
    kdata(5,n)=0;  
    kdata(6,n)=0;  
    kdata(7,n)=99999;  
    kdata(8,n) = 0;  
    kdata(9,n) = 0;  
    kdata(10,n) = 0;  
    kdata(11,n) = 0;  
    leftTSS(n)=NaN;  
    leftTSSntoJIHigh(n)=NaN;  
    leftTSSntoJIModerate(n)=NaN;  
    leftPL(n)=NaN;  
    leftPLntoJIHigh(n)=NaN;  
    leftPLntoJIModerate(n)=NaN;  
end
```

```
distapart = 0;  
distuncert = 0;  
disttoeach = 0;  
speedtoeach = 0;  
timetoeach = 0;
```

```
for i=1:ntimes  
    startplot=99999;  
    buf=sprintf('Step: time=%d',i-1); disp(buf);  
    % Time in this loop  
    t=times(i);  
  
    % Generate an observation for each node at this time  
    for s=1:NUM_NODES
```

```

    if NEW_RUN % new observation simulation required
        obs(s).report=sensor__report(Nodes(s),platforms,targets,s,i);
        observations(s,i)=obs(s); % record observations for plotting
    else % use old observation data
        obs(s)=observations(s,i);
    end
end

% First, do a local prediction
for s=1:NUM_NODES
    Nodes(s)=local_predict(Nodes(s),t);
end

% Second, a local update generating y tilde at each site
for s=1:NUM_NODES
    Nodes(s)=local_update(Nodes(s),obs(s),s);
end

% Next, establish network connection
com_net=set_net_topology(com_net,Nodes,t,'BROADCAST');

% Then communicate
% This sets up and exchanges data in channel filters
for s=1:NUM_NODES
    Nodes(s)=communicate(Nodes,com_net,s);
end

% Finally do global update by adding
for s=1:NUM_NODES
    Nodes(s)=assimilate(Nodes(s),t);
end

% Record results
for s=1:NUM_NODES
    Results_y(i,s,:)=Nodes(s).est.y;
    Results_Y(i,s,:)=Nodes(s).est.Y;
    PredictedResults_y(i,s,:)=Nodes(s).pred.y;
    PredictedResults_Y(i,s,:)=Nodes(s).pred.Y;
end

s=1;
warning off;
P=inv(squeeze(Results_Y(i,s,:))); %choose any sensor since all of them have the
same track information
if P ~= inf

```

```

% Calculate the position uncertainty of the target
% at present location
x=P*squeeze(Results_y(i,s,:));
sigma=sqrt(P);
xuncert = sigma(1,1);
yuncert = sigma(3,3);
distuncert = sqrt(xuncert^2 + yuncert^2);
b = circle([x(1), x(3)],distuncert);

% Project the position uncertainty of the target
% at final destination
FF=[1 (ntimes-i)*DT    0    0;
    0 1    0    0;
    0 0    1 (ntimes-i)*DT;
    0 0    0    1];

GG=[1 0 0 0;
    0 1 0 0;
    0 0 1 0;
    0 0 0 1];

QQ= [((ntimes-i)*DT)^2*SIGMA_XDOT^2 0 0 0;
     0 SIGMA_XDOT^2 0 0;
     0 0 ((ntimes-i)*DT)^2*SIGMA_YDOT^2 0;
     0 0 0 SIGMA_YDOT^2];

MM=(inv(FF))*squeeze(Results_Y(i,s,:,:))*(inv(FF));
SSigma=GG'*MM*GG+inv(QQ);
OOmega=MM*GG*inv(SSigma);
projYpred=MM-(OOmega*SSigma*OOmega');
projYpred=force_sym(projYpred);
projP = inv(projYpred);
projPsigma = sqrt(projP);
projxuncert = projPsigma(1,1);
projyuncert = projPsigma(3,3);
projdistuncert = sqrt(projxuncert^2 + projyuncert^2);
projypred=(eye(4,4)-(OOmega*GG'))*(inv(FF))*squeeze(Results_y(i,s,:));
projstate = projP*projypred;

% Check if SAW has entered Port Limits
if min(inpolygon (b.X, b.Y, TSS_X, TSS_Y))== 0 &&...
    min(inpolygon (b.X, b.Y, PL_X, PL_Y)) == 0
    buf=sprintf('Warning: Entered Port Limits at Timestep=%d',i-1); disp(buf);

% Check if SAW is turning towards Jurong Island

```

```

disttoreach = sqrt( abs(x(1)-302.1144444)^2+...
                    abs(x(3)-347.7716667)^2);
speedtoreach = sqrt(x(2)^2+x(4)^2);
timetoreach = disttoreach/speedtoreach;
% Calculate the time target left Port Limits
leftPL(i) = MAX_TIME-timetoreach;
% Calculate the distance between the projected target end point
% and Jurong Island
distapart = sqrt((projstate(1) - 302.1144444)^2+(projstate(3) - 347.7716667)^2);

% Threat level assessment
% Send warning if target is within 0.5nm to Jurong Island
if distapart < 5 && MAX_TIME - (i-1)*DT >= 0
    buf=sprintf('Warning: Heading Towards Jurong Island. Time to Impact = %.2f
sec',timetoreach); disp(buf);
    if projdistuncert > distapart
        buf=sprintf('Warning: High Confidence of Impact');disp(buf);
        startplot=i-1;
        leftPLntoJIHigh(i) = MAX_TIME-timetoreach;
    else
        buf=sprintf('Warning: Moderate Confidence of Impact');disp(buf);
        startplot=i-1;
        leftPLntoJIModerate(i) = MAX_TIME-timetoreach;
    end
end

% Check if SAW has left Traffic Separation Scheme
elseif min(inpolygon (b.X, b.Y, TSS_X, TSS_Y))== 0 &&...
    min(inpolygon (b.X, b.Y, PL_X, PL_Y)) == 1
    buf=sprintf('Warning: Left Traffic Separation Scheme at Timestep=%d',(i-1));
disp(buf);

% check if SAW is turning towards Jurong Island
disttoreach = sqrt( abs(x(1)-302.1144444)^2+...
                    abs(x(3)-347.7716667)^2);
speedtoreach = sqrt(x(2)^2+x(4)^2);
timetoreach = disttoreach/speedtoreach;
% Calculate the time target left TSS
leftTSS(i) = MAX_TIME-timetoreach;

% Calculate the distance between the projected target
% end point and Jurong Island
distapart = sqrt((projstate(1) - 302.1144444)^2 + ...
                (projstate(3) - 347.7716667)^2);

```

```

    % Threat level assessment
    % Send warning if target is within 0.5nm to Jurong Island
    if distapart < 5 && MAX_TIME - (i-1)*DT >= 0
        buf=sprintf('Warning: Heading Towards Jurong Island. Time to Impact = %.2f
sec',timetoreach); disp(buf);
        if projdistuncert > distapart
            buf=sprintf('Warning: High Confidence of Impact');disp(buf);
            startplot=i-1;
            leftTSSntoJIHigh(i) = MAX_TIME-timetoreach;
        else
            buf=sprintf('Warning: Moderate Confidence of Impact');disp(buf);
            startplot=i-1;
            leftTSSntoJIModerate(i) = MAX_TIME-timetoreach;
        end
    end
end

kdata(1,i)=times(i);
kdata(2,i)=projdistuncert*180;
kdata(3,i)=distapart*180;
kdata(4,i)=disttoreach*180;
kdata(5,i)=speedtoreach*180;
kdata(6,i)=timetoreach;
kdata(7,i)=startplot;
kdata(8,i)=projP(1,1);
kdata(9,i)=projP(3,3);
kdata(10,i)=projstate(1);
kdata(11,i)=projstate(3);

end
end

disp('Plotting Data...');

figure(20);
clf;
plot(kdata(1,:), (kdata(2,:), 'b', kdata(1,:), kdata(3,:), 'g')
legend('Distance Uncertainty', 'Distance Apart');
buf=sprintf('Estimated End Point');
title(buf)
xlim([min(kdata(7,1000:MAX_TIME/DT))*DT MAX_TIME-1*DT])
axis 'auto y'
xlabel('Simulated Time (s)')
ylabel('Distance (m)')

```

```

figure(21);
clf;
plot(kdata(1,:),kdata(4,:), 'b',kdata(1,:),kdata(5,:), 'g',kdata(1,:),kdata(6,:), 'r')
legend('Distance to Impact', 'Speed to Impact', 'Time to Impact');
buf=sprintf('Projected Distance, Speed and Time to Impact');
title(buf)
xlim([min(kdata(7,1000:MAX_TIME/DT))*DT MAX_TIME-1*DT])
axis 'auto y'
xlabel('Simulated Time (s)')

% Record the times the target crossed TSS, PL, & corresponding threat level
firstleftTSS = min(leftTSS(1000:MAX_TIME/DT));
firstleftTSSntoJIHigh = min(leftTSSntoJIHigh(1000:MAX_TIME/DT));
firstleftTSSntoJIModerate = min(leftTSSntoJIModerate(1000:MAX_TIME/DT));
firstleftPL = min(leftPL(1000:MAX_TIME/DT));
firstleftPLntoJIHigh = min(leftPLntoJIHigh(1000:MAX_TIME/DT));
firstleftPLntoJIModerate = min(leftPLntoJIModerate(1000:MAX_TIME/DT));

time = 0;
for i=1:ntimes
    P=inv(squeeze(Results_Y(i,s,:)));
    if P == inf
        time = i;
    end
end
warning on;

post__sim(targets,observations,platforms,destinations);
plot__tracks(targets,Results_y,Results_Y,times)
plot__errors(targets,Results_y,Results_Y,PredictedResults_y,...
    PredictedResults_Y, times,1,2,time)
plot__coms(platforms,com_net)
% This file defines the global variables

% General information
global TSS_X;      % Traffic Separation Scheme x-coordinates
global TSS_Y;      % Traffic Separation Scheme y-coordinates
global PL_X;       % Port Limits x-coordinates
global PL_Y;       % Port Limits y-coordinates
global DT;         % simulation time-step
global MAX_TIME;   % maximum simulation time

% Target definitions
global TARGET_TYPE; % defines type of target (xy or V phi)
global NUM_TARGETS; % number of targets to be simulated

```

```

% Platform definitions
global PLATFORM;    % Platform data [x0, y0, phi0, vel]
global NUM_PLATFORMS; % number of platforms
global MAX_RANGE;    % maximum observable range
global SIGMA_XDOT;
global SIGMA_YDOT;

% Destination definitions
global DESTINATION; % Destination data [x0, y0, phi0, vel]
global NUM_DESTINATIONS; % number of destinations

% Communications definitions
global NUM_NODES;    % number of sensor nodes
global NUM_CHANS;    % number of channels per node
global FULL_CON;     % connectivity

% State and Sensor definitions
global SIGMA_Q;       % process noise standard deviation for feature model
global SIGMA_RANGE_MPA; % Range observation noise
global SIGMA_BEARING_MPA; % Bearing observation noise
global SIGMA_RANGE_MPAC; % Range observation noise
global SIGMA_BEARING_MPAC; % Bearing observation noise
global SIGMA_RANGE_PV; % Range observation noise
global SIGMA_BEARING_PV; % Bearing observation noise
global SIGMA_RANGE_PCG; % Range observation noise
global SIGMA_BEARING_PCG; % Bearing observation noise
global Rfilter_MPA;    % observation noise covariance for MPA
global Rfilter_MPAC;   % observation noise covariance for MPAC
global Rfilter_PV;     % observation noise covariance for PV
global Rfilter_PCG;    % observation noise covariance for PCG
global F;              % state transition equation
global XSIZE;          % state dimension
global ZSIZE;          % observation dimension

% Results "database"
global kdata;
global esttimesigma;
global startplot;
global firstleftTSS;
global firstleftTSSntoJIHigh;
global firstleftTSSntoJIModerate;
global firstleftPL;
global firstleftPLntoJIHigh;
global firstleftPLntoJIModerate;

```



```

% This file initializes the global variables

% General polygon for Traffic Separation Scheme (TSS)
X = [738.9629; 700.8405; 503.0354; 343.3085; 301.0754; 188.7756; 142.3798;
291.7565; 356.6971; 474.1785; 703.9221; 738.9538; 738.9629];
Y = [443.8223; 412.1040; 377.3690; 303.7337; 330.3691; 359.0929; 316.1193;
230.0498; 289.3964; 351.7903; 379.3511; 408.0001; 443.8223];
TSS_X = X;
TSS_Y = Y;

% General polygon for Port Limits, estimated 0.5 mile beyond the TSS
[PL_X, PL_Y] = bufferm2('xy',X,Y,5,'out');

MAX_TIME=11500;      % simulation duration
DT=10;               % simulation time-step

% Target definitions
NUM_TARGETS=1;       % number of targets to be tracked

% Platform definitions
NUM_PLATFORMS=9;

% Destination definitions
NUM_DESTINATIONS=1;

% Number of sensor nodes
NUM_NODES=NUM_PLATFORMS;
NUM_CHANS=9;         % number of channels caters up to 9 nodes
FULL_CON=1;          % set to full connectivity between all nodes

% State and Sensor definitions
SIGMA_RANGE_MPA=5;    % 0.5nm range error equivalent
SIGMA_BEARING_MPA=0.0525; % 3 degrees bearing error in radians
SIGMA_RANGE_MPAC=6;   % 0.6nm range error equivalent
SIGMA_BEARING_MPAC=0.0875; % 5 degrees bearing error in radians
SIGMA_RANGE_PV=3;     % 0.3nm range error equivalent
SIGMA_BEARING_PV=0.0525; % 3 degrees bearing error in radians
SIGMA_RANGE_PCG=4;    % 0.4nm range error equivalent
SIGMA_BEARING_PCG=0.0875; % 5 degrees bearing error in radians
Rfilter_MPA = [SIGMA_RANGE_MPA^2 0;0 SIGMA_RANGE_MPA^2];
Rfilter_MPAC = [SIGMA_RANGE_MPAC^2 0;0 SIGMA_RANGE_MPAC^2];
Rfilter_PV = [SIGMA_RANGE_PV^2 0;0 SIGMA_RANGE_PV^2];
Rfilter_PCG = [SIGMA_RANGE_PCG^2 0;0 SIGMA_RANGE_PCG^2];
XSIZE=4;              % number of state components
ZSIZE=2;              % number of sensor measurement components

```

```

SIGMA_XDOT=0.00286;    % std dev of 1knot in the x-direction
SIGMA_YDOT=0.00286;    % std dev of 1knot in the y-direction

% Results "database"
kdata = [7 MAX_TIME/DT+1];
function [latb,lonb] = bufferm2(varargin) %lat,lon,dist,direction,npts,outputformat)
%BUFFERM2 Computes buffer zone around a polygon
%
% [latb,lonb] = bufferm2(lat,lon,dist,direction)
% [latb,lonb] = bufferm2(lat,lon,dist,direction,npts)
% [latb,lonb] = bufferm2(lat,lon,dist,direction,npts,outputformat)
% [xb, yb] = bufferm2('xy',x,y,dist,direction,npts,outputformat)
%
% This function was originally designed as a replacement for the Mapping
% Toolbox function bufferm, which calculates a buffer zone around a
% polygon. The original bufferm function had some serious bugs that could
% result in incorrect buffer results and/or errors, and was also very slow.
% As of R2006b, those bugs have been fixed. However, this version still
% maintains a few advantages over the original:
%
% - Can be applied to polygons in either geographical space (as in
%   bufferm) or in cartesian coordinates.
%
% - Better treatment of polygon holes. The original function simply
%   filled in all holes; this version trims or pads holes according to the
%   buffer width given.
%
% Input and output format is identical to bufferm unless the 'xy' option is
% specified, so it can be used interchangeably.
%
% Input variables:
%
% lat:      Latitude values defining the polygon to be buffered.
%           This can be either a NaN-delimited vector, or a cell
%           array containing individual polygonal contours (each of
%           which is a vector). External contours should be listed
%           in a clockwise direction, and internal contours (holes)
%           in a counterclockwise direction.
%
% lon:      Longitude values defining the polygon to be buffered.
%           Same format as lat.
%
% dist:     Width of buffer, in degrees of arc along the surface
%           (unless 'xy' is used, in which case units correspond to
%           x-y coordinates)

```

```

%
% direction:    'in' or 'out'
%
% npts:        Number of points used to construct the circles around
%              each polygon vertex. If omitted, default is 13.
%
% outputformat: 'vector' (NaN-delimited vectors), 'cutvector'
%              (NaN-clipped vectors with cuts connecting holes to the
%              exterior of the polygon), or 'cell' (cell arrays in
%              which each element of the cell array is a separate
%              polygon), defining format of output. If omitted,
%              default is 'vector'.
%
% 'xy':        If first input is 'xy', then data will be assumed to
%              lie on a cartesian plane rather than on a sphere. Use
%              x and y coordinates as first two inputs rather than lat
%              and lon. Units of x, y, and distance should be the
%              same.
%
% Output variables:
%
% latb:        Latitude values for buffer polygon
%
% lonb:        Longitude values for buffer polygon
%
% Example:
%
% load conus
% tol = 0.1; % Tolerance for simplifying polygon outlines
% [reducedlat, reducedlon] = reducem(gtlakelat, gtlakelon, tol);
% dist = 1; % Buffer distance in degrees
% [latb, lonb] = bufferm2(reducedlat, reducedlon, dist, 'out');
% figure('Renderer','painters')
% usamap({'MN','NY'})
% geoshow(latb, lonb, 'DisplayType', 'polygon', 'FaceColor', 'yellow')
% geoshow(gtlakelat, gtlakelon,...
%         'DisplayType', 'polygon', 'FaceColor', 'blue')
% geoshow(uslat, uslon)
% geoshow(statelat, statelon)
%
% See also:
%
% bufferm, polybool
%
% Copyright 2010 Kelly Kearney

```

```

%-----
% Check input
%-----

error(nargchk(3,7,nargin));

% Determine if geographic or cartesian

if ischar(varargin{1}) && strcmp(varargin{1}, 'xy')
    geo = false;
    param = varargin(2:end);
else
    geo = true;
    param = varargin;
end

% Set defaults if not provided as input

nparam = length(param);

if geo
    [lat, lon, dist] = deal(param{1:3});
else
    [lon, lat, dist] = deal(param{1:3}); % lon = x, lat = y for mental clarity, will switch
back at end
end

if nparam < 4
    direction = 'out';
else
    direction = param{4};
end

if nparam < 5
    npts = 13;
else
    npts = param{5};
end

if nparam < 6
    outputformat = 'vector';
else
    outputformat = param{6};
end

```

```

% Check format and dimensions of input

if ~ismember(direction, {'in', 'out'})
    error('Direction must be either "in" or "out".');
end

if ~ismember(outputformat, {'vector', 'cutvector', 'cell'})
    error('Unrecognized output format flag.');
```

```

end

if ~isnumeric(dist) || numel(dist) > 1
    error('Distance must be a scalar.');
```

```

end

if ~isnumeric(npts) || numel(npts) > 1
    error('Number of points must be a scalar.');
```

```

end

if iscell(lat)
    for il = 1:numel(lat)
        if ~isvector(lat{il}) || ~isvector(lon{il}) || ~isequal(length(lat{il}), length(lon{il}))
            error('Lat (or x) and lon (or y) must be vectors or cells of vectors with identical
dimensions');
```

```

        end
        lat{il} = lat{il}(:);
        lon{il} = lon{il}(:);
    end
else
    if ~isvector(lat) || ~isvector(lon) || ~isequal(length(lat), length(lon))
        error('Lat (or x) and lon (or y) must be vectors or cells of vectors with identical
dimensions');
```

```

    end
    lat = lat(:);
    lon = lon(:);
end

%-----
% Split polygon(s) into
% separate faces
%-----

if iscell(lat)
    [lat, lon] = polyjoin(lat, lon); % In case multiple faces in one cell.
end

```

```

[latcells, loncells] = polysplit(lat, lon);

%-----
% Create buffer shapes
%-----

plotflag = 0;

if plotflag

    Plt.x = lon;
    Plt.y = lat;

end

latcrall = cell(0);
loncrall = cell(0);

for ipoly = 1:length(latcells)

    % Circles around each vertex

    if geo
        [latc, lonc] = calccircgeo(latcells{ipoly}, loncells{ipoly}, dist, npts);
    else
        [lonc, latc] = calccirccart(loncells{ipoly}, latcells{ipoly}, dist, npts);
    end

    % Rectangles around each edge

    if geo
        [latr, lonr] = calcrecgeo(latcells{ipoly}, loncells{ipoly}, dist);
    else
        [lonr, latr] = calcreccart(loncells{ipoly}, latcells{ipoly}, dist);
    end

    % Union of circles and rectangles

    if plotflag
        Plt.rectx = lonr;
        Plt.recty = latr;
        Plt.circx = lonc;
        Plt.circy = latc;
    end
end

```

```

[latc, lonc] = multipolyunion(latc, lonc);
[latr, lonr] = multipolyunion(latr, lonr);

if plotflag
    Plt.rectcombox = lonr;
    Plt.rectcomboy = latr;
    Plt.circcombox = lonc;
    Plt.circcomboy = latc;
end

[loncr, latcr] = polybool('+', lonr, latr, lonc, latc);

% Union of new circle/rectangle combo with that from other faces

[loncrall, latcrall] = polybool('+', loncrall, latcrall, loncr, latcr);

% Plotting (for debugging only)

if plotflag

    Plt.allx = loncrall;
    Plt.ally = latcrall;

    if ipoly == 1
        figure;
        plot(Plt.x, Plt.y, 'k', 'linewidth', 2);
        hold on
    end

    plot(cat(2, Plt.rectx{:}), cat(2, Plt.recty{:}), 'b');
    plot(cat(2, Plt.circx{:}), cat(2, Plt.circy{:}), 'r');
    plot(Plt.allx{1}, Plt.ally{1}, 'g', 'linewidth', 2);

end

end

%-----
% Calculate union/difference
%-----

switch direction
case 'out'
    [lonb, latb] = polybool('+', loncells, latcells, loncrall, latcrall);

```

```

    case 'in'
        [lonb, latb] = polybool('-', loncells, latcells, loncrall, latcrall);
    end

    if plotflag
        [Plt.yfinal, Plt.xfinal] = polyjoin(latb, lonb);
        plot(Plt.xfinal, Plt.yfinal, 'linestyle', '--', 'color', [0 .5 0], 'linewidth', 2);
    end

    %-----
    % Reformat output
    %-----

    if ~geo
        y = latb; % Switch, since cartesian uses opposite order
        x = lonb;
        latb = x;
        lonb = y;
    end

    switch outputformat
        case 'vector'
            [latb, lonb] = polyjoin(latb, lonb);
        case 'cutvector'
            [latb, lonb] = polycut(latb, lonb);
        case 'cell'
    end

    %*****
    %*****

    function [latc, lonc] = calccircgeo(lat, lon, radius, npts)
    % lat and lon: n x 1 vectors
    % radius: scalar

    radius = ones(length(lat),1) * radius;
    [latc, lonc] = scircle1(lat, lon, radius, [], [], [], npts);
    latc = num2cell(latc, 1);
    lonc = num2cell(lonc, 1);

    function [latr, lonr] = calcrecgeo(lat, lon, halfwidth)
    % lat and lon: n x 1 vectors
    % halfwidth: scalar

```



```

range = halfwidth * ones(length(lat)-1, 1);

az = azimuth(lat(1:end-1), lon(1:end-1), lat(2:end), lon(2:end));

[latbl1,lonbl1] = reckon(lat(1:end-1), lon(1:end-1), range, az-90);
[latbr1,lonbr1] = reckon(lat(1:end-1), lon(1:end-1), range, az+90);
[latbl2,lonbl2] = reckon(lat(2:end), lon(2:end), range, az-90);
[latbr2,lonbr2] = reckon(lat(2:end), lon(2:end), range, az+90);

latr = [latbl1 latbl2 latbr2 latbr1 latbl1]';
lonr = [lonbl1 lonbl2 lonbr2 lonbr1 lonbl1]';
latr = num2cell(latr, 1);
lonr = num2cell(lonr, 1);

function [latu, lonu] = multipolyunion(lat, lon)
% lat and lon are n x 1 cell arrays of vectors

latu = lat{1};
lonu = lon{1};

for ip = 2:length(lat)
    [lonu, latu] = polybool('+', lonu, latu, lon{ip}, lat{ip});
end
[latu, lonu] = polysplit(latu, lonu);

function [xc, yc] = calccirccart(x, y, radius, npts)

ang = linspace(0, 2*pi, npts+1);
ang = ang(end-1:-1:1);
xc = bsxfun(@plus, x, radius * cos(ang));
yc = bsxfun(@plus, y, radius * sin(ang));
xc = num2cell(xc, 1);
yc = num2cell(yc, 1);

% if ~ispolycw(x,y)
%   [xc,yc] = poly2ccw(xc,yc);
% end

function [xrec, yrec] = calcreccart(x, y, halfwidth)

dx = diff(x);
dy = diff(y);

is1 = dx >= 0 & dy >= 0;

```

```

is2 = dx < 0 & dy >= 0;
is3 = dx < 0 & dy < 0;
is4 = dx >= 0 & dy < 0;

ish1 = dy == 0 & dx > 0;
ish2 = dy == 0 & dx < 0;

theta = zeros(5,1);
theta(is1 | is3) = atan(dy(is1 | is3)./dx(is1 | is3));
theta(is2 | is4) = -atan(dy(is2 | is4)./dx(is2 | is4));

[xl,xr,yl,yr] = deal(zeros(size(dx)));

xl(is1) = -halfwidth * sin(theta(is1));
xr(is1) = halfwidth * sin(theta(is1));
yl(is1) = halfwidth * cos(theta(is1));
yr(is1) = -halfwidth * cos(theta(is1));

xl(is2) = -halfwidth * sin(theta(is2));
xr(is2) = halfwidth * sin(theta(is2));
yl(is2) = -halfwidth * cos(theta(is2));
yr(is2) = halfwidth * cos(theta(is2));

xl(is3) = halfwidth * sin(theta(is3));
xr(is3) = -halfwidth * sin(theta(is3));
yl(is3) = -halfwidth * cos(theta(is3));
yr(is3) = halfwidth * cos(theta(is3));

xl(is4) = halfwidth * sin(theta(is4));
xr(is4) = -halfwidth * sin(theta(is4));
yl(is4) = halfwidth * cos(theta(is4));
yr(is4) = -halfwidth * cos(theta(is4));

xrec = [xl+x(1:end-1) xl+x(2:end) xr+x(2:end) xr+x(1:end-1) xl+x(1:end-1)];
yrec = [yl+y(1:end-1) yl+y(2:end) yr+y(2:end) yr+y(1:end-1) yl+y(1:end-1)];

xrec = num2cell(xrec, 2);
yrec = num2cell(yrec, 2);

```


% This function draws a circle for a given center and radius

function bubble = circle(center,radius)

Resolution = 100;

THETA=linspace(0,2*pi,Resolution);

RHO=ones(1,Resolution)*radius;

[X,Y] = pol2cart(THETA,RHO);

bubble.X=X+center(1);

bubble.Y=Y+center(2);

```
% This function generates the SAW's intended destinations  
% (i.e. Areas A1, A2, and A3 of Jurong Island)
```

```
function destinations = generate__destinations(times)
```

```
% Define the global variables  
globals;
```

```
% Define the destination locations
```

```
for i=1:NUM_DESTINATIONS
```

```
    % Area A1
```

```
    if i==1
```

```
        destinations(1,i) = make_destination;
```

```
        destinations(1,i).time=times(1);
```

```
        destinations(1,i).id=i;
```

```
        destinations(1,i).x = 302.1144444;
```

```
        destinations(1,i).y = 347.7716667;
```

```
    % Now iterate for all times
```

```
    [temp,ntimes]=size(times);
```

```
    for n=2:ntimes
```

```
        destinations(n,i)=destination_step(destinations(n-1,i),times(n));
```

```
    end
```

```
end
```

```
    % Area A2
```

```
    if i==2
```

```
        destinations(1,i) = make_destination;
```

```
        destinations(1,i).time=times(1);
```

```
        destinations(1,i).id=i;
```

```
        destinations(1,i).x = 322.7311111;
```

```
        destinations(1,i).y = 367.2116667;
```

```
    % Now iterate for all times
```

```
    [temp,ntimes]=size(times);
```

```
    for n=2:ntimes
```

```
        destinations(n,i)=destination_step(destinations(n-1,i),times(n));
```

```
    end
```

```
end
```

```
    % Area A3
```

```
    if i==3
```

```
        destinations(1,i) = make_destination;
```

```
        destinations(1,i).time=times(1);
```

```
destinations(1,i).id=i;

destinations(1,i).x = 343.3450000;
destinations(1,i).y = 382.5572222;

% Now iterate for all times
[temp,ntimes]=size(times);
for n=2:ntimes
    destinations(n,i)=destination_step(destinations(n-1,i),times(n));
end
end
end
```

```
% This function initialises a new destination  
  
function new=make_destination  
  
new=struct('id',0,'time',0,'x',0,'y',0,'phi',0,'vel',0,'gamma',0);
```

```

% This function is to step a destination

function next_destination=destination_step(destination,time)

globals;

% This is a no motion model

dt=time-destination.time;
next_destination = make_destination; % space for destination data structure
next_destination.id=destination.id;
next_destination.time = time;
next_destination.x = destination.x;
next_destination.y = destination.y;
next_destination.phi = destination.phi;
next_destination.vel = destination.vel;
next_destination.gamma = destination.gamma;

```



```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function generates true target information for later observation
% and tracking. Dynamics of target are contained in "target_step", number
% targets is defined by the globally defined variable NUM_TARGETS. A data
% point is generated at each of a set of "times."

function targets = generate__targets(times)

% Define the global variables
globals;

% Define a SAW at pre-fixed location in the TSS
for i=1
    targets(1,i) = make_target;
    targets(1,i).time=times(1);
    targets(1,i).id=i;
    targets(1,i).x = 738.9616667;
    targets(1,i).y = 438.7050000;
    targets(1,i).gamma=0;

    % Now iterate for all times
    [temp,ntimes]=size(times);
    for n=2:ntimes
        targets(n,i)=target_step_SAW_path(targets(n-1,i),times(n));
    end
end

% Create non-SAW contacts at random locations within the TSS
% Not used since this thesis covers a single target
for i=2:NUM_TARGETS
    targets(1,i) = make_target;
    targets(1,i).time=times(1);
    targets(1,i).id=i;

    x = TSS_X(1)*rand;
    y = TSS_Y(1)*rand;

    while (inpolygon (x, y,TSS_X, TSS_Y) ~= 1)

```

```

    x = TSS_X(1)*rand;
    y = TSS_Y(1)*rand;
end

    targets(1,i).x = x;
    targets(1,i).y = y;
    targets(1,i).phi = 2*pi*rand -pi;
    targets(1,i).vel = MIN_TARGET_VEL + (MAX_TARGET_VEL-
MIN_TARGET_VEL)*rand;
    targets(1,i).gamma=0;

    % Now iterate for all time
    [temp,ntimes]=size(times);
    for n=2:ntimes
        targets(n,i)=target_step(targets(n-1,i),times(n));
    end
end

```

```
% Acknowledgement:  
% This code originates from H. Durrant-Whyte  
% Australian Centre for Field Robotics  
% The University of Sydney, Australia  
  
% This function initialises a new target  
  
function new=make_target  
  
new=struct('id',0,'time',0,'x',0,'y',0,'phi',0,'vel',0,'gamma',0);
```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This is the trajectory taken by the SAW

function next_target=target_step_SAW_path(target,time)

globals;

dt=time-target.time;
next_target = make_target;
next_target.id=target.id;
next_target.time = time;

% target.vel = 0.042870370 (in m/s which equates to 15knots)
% target.vel = 0.059160494 (in m/s which equates to 20knots)
% target.vel = 0.060018519 (in m/s which equates to 21knots)
% target.vel = 0.062876543 (in m/s which equates to 22knots)

if (next_target.time >= 0 && next_target.time < 1086)
    next_target.x = target.x + dt*0.042870370*cos(-pi+0.720);
    next_target.y = target.y + dt*0.042870370*sin(-pi+0.720);
    next_target.phi = -pi+0.720;
    next_target.vel = 0.042870370;

elseif (next_target.time >= 1086 && next_target.time < 5890)
    next_target.x = target.x + dt*0.042870370*cos(-pi+0.170);
    next_target.y = target.y + dt*0.042870370*sin(-pi+0.170);
    next_target.phi = -pi+0.170;
    next_target.vel = 0.042870370;

elseif (next_target.time >= 5890 && next_target.time < 10199)
    next_target.x = target.x + dt*0.042870370*cos(-pi+0.430);
    next_target.y = target.y + dt*0.042870370*sin(-pi+0.430);
    next_target.phi = -pi+0.430;
    next_target.vel = 0.042870370;

elseif (next_target.time >= 10199 && next_target.time < 11051)
    next_target.x = target.x + dt*0.042870370*cos(pi-0.770);
    next_target.y = target.y + dt*0.042870370*sin(pi-0.770);

```

```
next_target.phi = pi-0.770;
next_target.vel = 0.042870370;

elseif (next_target.time >= 11051 && next_target.time < 11500)
    next_target.x = target.x + dt*0.059160494*cos(pi-1.360);
    next_target.y = target.y + dt*0.059160494*sin(pi-1.360);
    next_target.phi = pi-1.360;
    next_target.vel = 0.059160494;

elseif (next_target.time >= 11500)
    next_target.x = target.x;
    next_target.y = target.y;
    next_target.phi = 0;
    next_target.vel = 0;
end
```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function generates trajectories for platforms

function platforms = generate__platforms(times)

% Define the global variables
globals;

% Define 5 fixed platform initial locations

% Model the MPA radar stations
for i=1:NUM_PLATFORMS
    if i==1
        platforms(1,i) = make_platform;
        platforms(1,i).time=times(1);
        platforms(1,i).id=i;
        platforms(1,i).x = 285.4700000;
        platforms(1,i).y = 360.8527778;
        platforms(1,i).phi = 2*pi*rand -pi;
        platforms(1,i).vel = 0;
        platforms(1,i).gamma=0;

        % Now iterate for all time
        [temp,ntimes]=size(times);
        for n=2:ntimes
            platforms(n,i)=platform_step(platforms(n-1,i),times(n));
        end
    end

    if i==2
        platforms(1,i) = make_platform;
        platforms(1,i).time=times(1);
        platforms(1,i).id=i;
        platforms(1,i).x = 342.7972222;
        platforms(1,i).y = 311.9850000;
        platforms(1,i).phi = 2*pi*rand -pi;
        platforms(1,i).vel = 0;
        platforms(1,i).gamma=0;
    end
end

```

```

% Now iterate for all time
[temp,ntimes]=size(times);
for n=2:ntimes
    platforms(n,i)=platform_step(platforms(n-1,i),times(n));
end
end

if i==3
    platforms(1,i) = make_platform;
    platforms(1,i).time=times(1);
    platforms(1,i).id=i;
    platforms(1,i).x = 413.1333333;
    platforms(1,i).y = 349.8194444;
    platforms(1,i).phi = 2*pi*rand -pi;
    platforms(1,i).vel = 0;
    platforms(1,i).gamma=0;

    % Now iterate for all time
    [temp,ntimes]=size(times);
    for n=2:ntimes
        platforms(n,i)=platform_step(platforms(n-1,i),times(n));
    end
end

if i==4
    platforms(1,i) = make_platform;
    platforms(1,i).time=times(1);
    platforms(1,i).id=i;
    platforms(1,i).x = 461.4761111;
    platforms(1,i).y = 403.4261111;
    platforms(1,i).phi = 2*pi*rand -pi;
    platforms(1,i).vel = 0;
    platforms(1,i).gamma=0;

    % Now iterate for all time
    [temp,ntimes]=size(times);
    for n=2:ntimes
        platforms(n,i)=platform_step(platforms(n-1,i),times(n));
    end
end

if i==5
    platforms(1,i) = make_platform;
    platforms(1,i).time=times(1);

```

```

platforms(1,i).id=i;
platforms(1,i).x = 517.0844444;
platforms(1,i).y = 421.9305556;
platforms(1,i).phi = 2*pi*rand -pi;
platforms(1,i).vel = 0;
platforms(1,i).gamma=0;

% Now iterate for all time
[temp,ntimes]=size(times);
for n=2:ntimes
    platforms(n,i)=platform_step(platforms(n-1,i),times(n));
end
end

% Define 3 mobile platform initial locations

% Model the CPC
if i==6
    platforms(1,i) = make_platform;
    platforms(1,i).time=times(1);
    platforms(1,i).id=i;
    platforms(1,i).x = 474.1785056;
    platforms(1,i).y = 351.7903000;
    platforms(1,i).phi = 2*pi*rand -pi;
    platforms(1,i).vel = 0;
    targets(1,i).gamma=0;

    % Now iterate for all time
    [temp,ntimes]=size(times);
    for n=2:ntimes
        platforms(n,i)=platform_step_PCG1_path(platforms(n-1,i),times(n));
    end
end

if i==7
    platforms(1,i) = make_platform;
    platforms(1,i).time=times(1);
    platforms(1,i).id=i;
    platforms(1,i).x = 474.1785056;
    platforms(1,i).y = 351.7903000;
    platforms(1,i).phi = 2*pi*rand -pi;
    platforms(1,i).vel = 0;
    targets(1,i).gamma=0;

    % Now iterate for all time

```



```

[temp,ntimes]=size(times);
for n=2:ntimes
    platforms(n,i)=platform_step_PCG2_path(platforms(n-1,i),times(n));
end
end

% Model the PV
if i==8
    platforms(1,i) = make_platform;
    platforms(1,i).time=times(1);
    platforms(1,i).id=i;
    platforms(1,i).x = 474.1785056;
    platforms(1,i).y = 351.7903000;
    platforms(1,i).phi = 2*pi*rand -pi;
    platforms(1,i).vel = 0;
    targets(1,i).gamma=0;

    % Now iterate for all time
    [temp,ntimes]=size(times);
    for n=2:ntimes
        platforms(n,i)=platform_step_PV_path(platforms(n-1,i),times(n));
    end
end

% Given the speed and sensor coverage of the MPaA,
% its surveying of the small TSS can be modeled as a "fixed" sensor
% with complete of the TSS during its operation.
% Model the MPaA
if i==9
    platforms(1,i) = make_platform;
    platforms(1,i).time=times(1);
    platforms(1,i).id=i;
    platforms(1,i).x = 474.1785056;
    platforms(1,i).y = 351.7903000;
    platforms(1,i).phi = 2*pi*rand -pi;
    platforms(1,i).vel = 0;
    targets(1,i).gamma=0;

    % Now iterate for all time
    [temp,ntimes]=size(times);
    for n=2:ntimes
        platforms(n,i)=platform_step_MPAC(platforms(n-1,i),times(n));
    end
end
end
end

```

```
% Acknowledgement:  
% This code originates from H. Durrant-Whyte  
% Australian Centre for Field Robotics  
% The University of Sydney, Australia  
  
% This function initialises a new platform  
  
function new=make_platform  
  
new=struct('id',0,'time',0,'x',0,'y',0,'phi',0,'vel',0,'gamma',0);
```

```
% Acknowledgement:  
% This code originates from H. Durrant-Whyte  
% Australian Centre for Field Robotics  
% The University of Sydney, Australia
```

% This is a no motion platform model for MPA radar station

function next_platform=platform_step(platform,time)

globals;

```
dt=time-platform.time;
next_platform = make_platform;
next_platform.id=platform.id;
next_platform.time = time;
next_platform.x = platform.x;
next_platform.y = platform.y;
next_platform.phi = platform.phi;
next_platform.vel = platform.vel;
next_platform.gamma = platform.gamma;
```

```

% This is the platform motion model for one of the two CPCs for a day

function next_platform=platform_step_PCG1_path(platform,time)

globals;

dt=time-platform.time;
next_platform = make_platform;
next_platform.id=platform.id;
next_platform.time = time;

if (next_platform.time >= 0)
    next_platform.x = 474.1785056;
    next_platform.y = 351.7903000;
    next_platform.phi = platform.phi;
    next_platform.vel = platform.vel;
    next_platform.gamma = platform.gamma;
end

%-----
% This models the platform's patrol path. However, by making the assumption
% that the sensors have extended range to simplify sensor management, this
% modeling is not used here. This will be useful for future work with the
% inclusion of sensor management.
%
% if (next_platform.time >= 0 && next_platform.time < 18000)
%     next_platform.x = platform.x + dt*0.015061617*cos(1.519574188-0.5*pi);
%     next_platform.y = platform.y + dt*0.015061617*sin(1.519574188-0.5*pi);
%     next_platform.phi = 1.519574188-0.5*pi;
%     next_platform.vel = 0.015061617;
%     next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 18000 && next_platform.time < 32400)
%     next_platform.x = platform.x + dt*0.025102695*cos(1.519574188+0.5*pi);
%     next_platform.y = platform.y + dt*0.025102695*sin(1.519574188+0.5*pi);
%     next_platform.phi = 1.519574188+0.5*pi;
%     next_platform.vel = 0.025102695;
%     next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 32400 && next_platform.time < 46800)
%     next_platform.x = platform.x + dt*0.015061617*cos(1.519574188-0.5*pi);
%     next_platform.y = platform.y + dt*0.015061617*sin(1.519574188-0.5*pi);
%     %next_platform.phi = 1.519574188-0.5*pi;
%     %next_platform.vel = 0.015061617;

```

```

%   %next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 46800 && next_platform.time < 61200)
%   next_platform.x = platform.x + dt*0.025102695*cos(1.519574188+0.5*pi);
%   next_platform.y = platform.y + dt*0.025102695*sin(1.519574188+0.5*pi);
%   next_platform.phi = 1.519574188+0.5*pi;
%   next_platform.vel = 0.025102695;
%   next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 61200 && next_platform.time < 75600)
%   next_platform.x = platform.x + dt*0.015061617*cos(1.519574188-0.5*pi);
%   next_platform.y = platform.y + dt*0.015061617*sin(1.519574188-0.5*pi);
%   %next_platform.phi = 1.519574188-0.5*pi;
%   %next_platform.vel = 0.015061617;
%   %next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 75600 && next_platform.time < 86400)
%   next_platform.x = platform.x + dt*0.025102695*cos(1.519574188+0.5*pi);
%   next_platform.y = platform.y + dt*0.025102695*sin(1.519574188+0.5*pi);
%   next_platform.phi = 1.519574188+0.5*pi;
%   next_platform.vel = 0.025102695;
%   next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 86400)
%   next_platform.x = platform.x;
%   next_platform.y = platform.y;
%   next_platform.phi = 0;
%   next_platform.vel = 0;
%   next_platform.gamma = 0;
% end
%-----

```

```

% This is the platform motion model for one of the two CPCs for a day

function next_platform=platform_step_PCG2_path(platform,time)

globals;

dt=time-platform.time;
next_platform = make_platform;
next_platform.id=platform.id;
next_platform.time = time;

if (next_platform.time >= 0)
    next_platform.x = 474.1785056;
    next_platform.y = 351.7903000;
    next_platform.phi = platform.phi;
    next_platform.vel = platform.vel;
    next_platform.gamma = platform.gamma;
end

%-----
% This models the platform's patrol path. However, by making the assumption
% that the sensors have extended range to simplify sensor management, this
% modeling is not used here. This will be useful for future work with the
% inclusion of sensor management.
%
% if (next_platform.time >= 0 && next_platform.time < 18000)
%     next_platform.x = platform.x + dt*0.013140537*cos(-1.330818664+0.5*pi);
%     next_platform.y = platform.y + dt*0.013140537*sin(-1.330818664+0.5*pi);
%     next_platform.phi = -1.330818664+0.5*pi;
%     next_platform.vel = 0.013140537;
%     next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 18000 && next_platform.time < 32400)
%     next_platform.x = platform.x + dt*0.021900894*cos(-1.330818664-0.5*pi);
%     next_platform.y = platform.y + dt*0.021900894*sin(-1.330818664-0.5*pi);
%     next_platform.phi = -1.330818664-0.5*pi;
%     next_platform.vel = 0.021900894;
%     next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 32400 && next_platform.time < 46800)
%     next_platform.x = platform.x + dt*0.013140537*cos(-1.330818664+0.5*pi);
%     next_platform.y = platform.y + dt*0.013140537*sin(-1.330818664+0.5*pi);
%     next_platform.phi = -1.330818664+0.5*pi;
%     next_platform.vel = 0.013140537;

```

```

%   next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 46800 && next_platform.time < 61200)
%   next_platform.x = platform.x + dt*0.021900894*cos(-1.330818664-0.5*pi);
%   next_platform.y = platform.y + dt*0.021900894*sin(-1.330818664-0.5*pi);
%   next_platform.phi = -1.330818664-0.5*pi;
%   next_platform.vel = 0.021900894;
%   next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 61200 && next_platform.time < 75600)
%   next_platform.x = platform.x + dt*0.013140537*cos(-1.330818664+0.5*pi);
%   next_platform.y = platform.y + dt*0.013140537*sin(-1.330818664+0.5*pi);
%   next_platform.phi = -1.330818664+0.5*pi;
%   next_platform.vel = 0.013140537;
%   next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 75600 && next_platform.time < 86400)
%   next_platform.x = platform.x + dt*0.021900894*cos(-1.330818664-0.5*pi);
%   next_platform.y = platform.y + dt*0.021900894*sin(-1.330818664-0.5*pi);
%   next_platform.phi = -1.330818664-0.5*pi;
%   next_platform.vel = 0.021900894;
%   next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 86400)
%   next_platform.x = platform.x;
%   next_platform.y = platform.y;
%   next_platform.phi = 0;
%   next_platform.vel = 0;
%   next_platform.gamma = 0;
% end
%-----

```



```

% This is the platform motion model for PV for a day

function next_platform=platform_step_PV_path(platform,time)

globals;

dt=time-platform.time;
next_platform = make_platform;
next_platform.id=platform.id;
next_platform.time = time;

if (next_platform.time >= 0)
    next_platform.x = 474.1785056;
    next_platform.y = 351.7903000;
    next_platform.phi = platform.phi;
    next_platform.vel = platform.vel;
    next_platform.gamma = platform.gamma;
end

%-----
% This models the platform's patrol path. However, by making the assumption
% that the sensors have extended range to simplify sensor management, this
% modeling is not used here. This will be useful for future work with the
% inclusion of sensor management.
%
% if (next_platform.time >= 0 && next_platform.time < 43200)
%     next_platform.x = platform.x + dt*0.013972407*cos(0.152813367);
%     next_platform.y = platform.y + dt*0.013972407*sin(0.152813367);
%     next_platform.phi = 0.152813367;
%     next_platform.vel = 0.013972407;
%     next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 43200 && next_platform.time < 86400)
%     next_platform.x = platform.x + dt*0.013972407*cos(0.152813367+pi);
%     next_platform.y = platform.y + dt*0.013972407*sin(0.152813367+pi);
%     next_platform.phi = 0.152813367+pi;
%     next_platform.vel = 0.013972407;
%     next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 86400)
%     next_platform.x = platform.x;
%     next_platform.y = platform.y;
%     next_platform.phi = 0;
%     next_platform.vel = 0;

```

```
% next_platform.gamma = 0;  
% end  
%-----
```

```

% This is the platform motion model for MPaA

function next_platform=platform_step_MPAC(platform,time)

globals;

next_platform = make_platform;
next_platform.id=platform.id;
next_platform.time = time;

if (next_platform.time >= 0)
    next_platform.x = 474.1785056;
    next_platform.y = 351.7903000;
    next_platform.phi = platform.phi;
    next_platform.vel = platform.vel;
    next_platform.gamma = platform.gamma;
end

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function is to get a fixed filter parameter block

function filter=get_filter_params

globals;

% Assume a constant velocity model;
filter.XDIM=4;
filter.ZDIM=2;
filter.Q= [DT^2*SIGMA_XDOT^2 0 0 0;
           0 SIGMA_XDOT^2 0 0;
           0 0 DT^2*SIGMA_YDOT^2 0;
           0 0 0 SIGMA_YDOT^2];
filter.H= [1 0 0 0; 0 0 1 0];

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% Returns a sensor data structure which contains all
% necessary information for a decentralised sensor.

function sensor=make__sensor(id,filter)

globals;

% Define sensors for the 5 fixed platforms
if id==1
    sensor.id=id;
    sensor.time=0.0;
    sensor.r_err=SIGMA_RANGE_MPA;
    sensor.b_err=SIGMA_BEARING_MPA;
    sensor.point=0;      % zero point angle
    sensor.beam_view=2*pi;
    sensor.max_range=154; % an equivalent of 15nm
    sensor.filter=filter;
    sensor.est=make_track(id,filter.XDIM,0.0);
    sensor.pred=make_track(id,filter.XDIM,0.0);
    sensor.num_chans=NUM_CHANS;
    for i=1:NUM_CHANS
        sensor.chan_in(i)=make_track(i,filter.XDIM,0.0);
        sensor.chan_out(i)=make_track(i,filter.XDIM,0.0);
    end

elseif id==2
    sensor.id=id;
    sensor.time=0.0;
    sensor.r_err=SIGMA_RANGE_MPA;
    sensor.b_err=SIGMA_BEARING_MPA;
    sensor.point=0;
    sensor.beam_view=2*pi;
    sensor.max_range=154;
    sensor.filter=filter;
    sensor.est=make_track(id,filter.XDIM,0.0);
    sensor.pred=make_track(id,filter.XDIM,0.0);
    sensor.num_chans=NUM_CHANS;

```

```

for i=1:NUM_CHANS
    sensor.chan_in(i)=make_track(i,filter.XDIM,0.0);
    sensor.chan_out(i)=make_track(i,filter.XDIM,0.0);
end

elseif id==3
    sensor.id=id;
    sensor.time=0.0;
    sensor.r_err=SIGMA_RANGE_MPA;
    sensor.b_err=SIGMA_BEARING_MPA;
    sensor.point=0;
    sensor.beam_view=2*pi;
    sensor.max_range=154;
    sensor.filter=filter;
    sensor.est=make_track(id,filter.XDIM,0.0);
    sensor.pred=make_track(id,filter.XDIM,0.0);
    sensor.num_chans=NUM_CHANS;
    for i=1:NUM_CHANS
        sensor.chan_in(i)=make_track(i,filter.XDIM,0.0);
        sensor.chan_out(i)=make_track(i,filter.XDIM,0.0);
    end

elseif id==4
    sensor.id=id;
    sensor.time=0.0;
    sensor.r_err=SIGMA_RANGE_MPA;
    sensor.b_err=SIGMA_BEARING_MPA;
    sensor.point=0;
    sensor.beam_view=2*pi;
    sensor.max_range=154;
    sensor.filter=filter;
    sensor.est=make_track(id,filter.XDIM,0.0);
    sensor.pred=make_track(id,filter.XDIM,0.0);
    sensor.num_chans=NUM_CHANS;
    for i=1:NUM_CHANS
        sensor.chan_in(i)=make_track(i,filter.XDIM,0.0);
        sensor.chan_out(i)=make_track(i,filter.XDIM,0.0);
    end

elseif id==5
    sensor.id=id;
    sensor.time=0.0;
    sensor.r_err=SIGMA_RANGE_MPA;
    sensor.b_err=SIGMA_BEARING_MPA;
    sensor.point=0;

```

```

sensor.beam_view=2*pi;
sensor.max_range=154;
sensor.filter=filter;
sensor.est=make_track(id,filter.XDIM,0.0);
sensor.pred=make_track(id,filter.XDIM,0.0);
sensor.num_chans=NUM_CHANS;
for i=1:NUM_CHANS
sensor.chan_in(i)=make_track(i,filter.XDIM,0.0);
sensor.chan_out(i)=make_track(i,filter.XDIM,0.0);
end

```

% Define sensors for the 4 mobile platforms

% Define the CPC sensor

```

elseif id==6
sensor.id=id;
sensor.time=0.0;
sensor.r_err=SIGMA_RANGE_PCG;
sensor.b_err=SIGMA_BEARING_PCG;
sensor.point=0;      % zero point angle
sensor.beam_view=2*pi;
sensor.max_range=1000; % range extended to simulate
                      % presence at Jurong Island
sensor.filter=filter;
sensor.est=make_track(id,filter.XDIM,0.0);
sensor.pred=make_track(id,filter.XDIM,0.0);
sensor.num_chans=NUM_CHANS;
for i=1:NUM_CHANS
sensor.chan_in(i)=make_track(i,filter.XDIM,0.0);
sensor.chan_out(i)=make_track(i,filter.XDIM,0.0);
end

```

% Define the CPC sensor

```

elseif id==7
sensor.id=id;
sensor.time=0.0;
sensor.r_err=SIGMA_RANGE_PCG;
sensor.b_err=SIGMA_BEARING_PCG;
sensor.point=0;
sensor.beam_view=2*pi;
sensor.max_range=1000;
sensor.filter=filter;
sensor.est=make_track(id,filter.XDIM,0.0);
sensor.pred=make_track(id,filter.XDIM,0.0);
sensor.num_chans=NUM_CHANS;

```

```

    for i=1:NUM_CHANS
        sensor.chan_in(i)=make_track(i,filter.XDIM,0.0);
        sensor.chan_out(i)=make_track(i,filter.XDIM,0.0);
    end

% Define the PV sensor
elseif id==8
    sensor.id=id;
    sensor.time=0.0;
    sensor.r_err=SIGMA_RANGE_PV;
    sensor.b_err=SIGMA_BEARING_PV;
    sensor.point=0;
    sensor.beam_view=2*pi;
    sensor.max_range=1000;
    sensor.filter=filter;
    sensor.est=make_track(id,filter.XDIM,0.0);
    sensor.pred=make_track(id,filter.XDIM,0.0);
    sensor.num_chans=NUM_CHANS;
    for i=1:NUM_CHANS
        sensor.chan_in(i)=make_track(i,filter.XDIM,0.0);
        sensor.chan_out(i)=make_track(i,filter.XDIM,0.0);
    end

% Define the MPaA sensor
elseif id==9
    sensor.id=id;
    sensor.time=0.0;
    sensor.r_err=SIGMA_RANGE_MPAC;
    sensor.b_err=SIGMA_BEARING_MPAC;
    sensor.point=0;
    sensor.beam_view=2*pi;
    sensor.max_range=1000;
    sensor.filter=filter;
    sensor.est=make_track(id,filter.XDIM,0.0);
    sensor.pred=make_track(id,filter.XDIM,0.0);
    sensor.num_chans=NUM_CHANS;
    for i=1:NUM_CHANS
        sensor.chan_in(i)=make_track(i,filter.XDIM,0.0);
        sensor.chan_out(i)=make_track(i,filter.XDIM,0.0);
    end
end
end

```



```
% Acknowledgement:  
% This code originates from H. Durrant-Whyte  
% Australian Centre for Field Robotics  
% The University of Sydney, Australia
```

```
% This function makes space for a track
```

```
function track=make_track(id,dim,time)
```

```
track.id=id;  
track.XDIM=dim;  
track.time=time;  
track.y=zeros(dim,1);  
track.Y=zeros(dim,dim);
```

% This function initialises a network structure

function network=init_net(platforms,nodes)

 nsize=length(nodes);

 network=ones(nsize,nsize);

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function constructs a report for a sensor at a particular time

function report=sensor__report(sensor,platforms,targets,nsensor,ntime)

globals;

report=zeros(NUM_TARGETS,5);

% Find platform location at this time
px=platforms(ntime,nsensor).x;
py=platforms(ntime,nsensor).y;

for tnum=1:NUM_TARGETS
    % Find target location
    tx=targets(ntime,tnum).x;
    ty=targets(ntime,tnum).y;
    % Compute range and bearing
    dx=tx-px;
    dy=ty-py;
    range=sqrt(dx*dx + dy*dy);
    bearing=atan2(dy,dx);
    % Determine if this is actually visible
    if range < sensor.max_range
        % Add error from sensor model
        report(tnum,1)=tnum;
        report(tnum,2)=range + sensor.r_err*(-1+2*rand);
        report(tnum,3)=bearing + sensor.b_err*(-1+2*rand);
        report(tnum,4)=px;
        report(tnum,5)=py;
    else
        report(tnum,1)=tnum;
        report(tnum,2)=NaN;
        report(tnum,3)=NaN;
        report(tnum,4)=NaN;
        report(tnum,5)=NaN;
    end
end
end

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function performs local prediction for each node using the CVM

function node=local_predict(node,time)

globals;

dt=time-node.time;
if dt < 0
    error('Negative prediction step in state_pred')
end

% Compute matrices
F=[1 dt 0 0;
    0 1 0 0;
    0 0 1 dt;
    0 0 0 1];
G=[1 0 0 0;
    0 1 0 0;
    0 0 1 0;
    0 0 0 1];
Q=node.filter.Q;

% Do prediction
M=(inv(F))'*(node.est.Y)*(inv(F));
Sigma=G'*M*G+inv(Q);
Omega=M*G*inv(Sigma);
node.pred.Y=M-(Omega*Sigma*Omega');
node.pred.y=(eye(4,4)-(Omega*G'))*(inv(F))'*(node.est.y);
node.time=time;
% Seems to require for stability in Matlab
node.pred.Y=force_sym(node.pred.Y);

```

```
% Acknowledgement:  
% This code originates from H. Durrant-Whyte  
% Australian Centre for Field Robotics  
% The University of Sydney, Australia
```

```
% Modified by J. Ng on 30 May 2011.
```

```
% This function forces symmetry on a matrix
```

```
function sigma=force_sym(sigma)  
  
[nx,ny]=size(sigma);  
if nx ~= ny  
    error('Matrix must be square');  
end  
  
for i=1:nx  
    for j=i+1:nx  
        temp=(sigma(i,j)+sigma(j,i))/2;  
        sigma(i,j)=temp;  
        sigma(j,i)=temp;  
    end  
end
```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function performs information filter local update for each node

function node=local_update(node,obs,i)

globals;

% Observation model
H=node.filter.H;

if isnan(obs.report(2:5)) % Wait for observation
    % Construct information terms
    info=zeros(XSIZE,1);
    Info=zeros(XSIZE, XSIZE);
    % Add the information to prediction
    node.est.y=node.pred.y+info;
    node.est.Y=node.pred.Y+Info;
    % Seems to require the next step for stability in Matlab
    node.est.Y=force_sym(node.est.Y);

elseif (obs.report(2:5) ~= 99999) % Update with observation
    % Extract observation
    [zx,zy,R]=xy_obs(obs.report,1,i); % R is not used
    z=[zx;zy];
    if i==1
        Ri=inv(Rfilter_MPA);
    elseif i==2
        Ri=inv(Rfilter_MPA);
    elseif i==3
        Ri=inv(Rfilter_MPA);
    elseif i==4
        Ri=inv(Rfilter_MPA);
    elseif i==5
        Ri=inv(Rfilter_MPA);
    elseif i==6
        Ri=inv(Rfilter_PCG);
    elseif i==7
        Ri=inv(Rfilter_PCG);

```

```

elseif i==8
    Ri=inv(Rfilter_PV);
elseif i==9
    Ri=inv(Rfilter_MPAC);
end
% Construct information terms
info=H'*Ri*z;
Info=H'*Ri*H;
% Add the information to prediction
node.est.y=node.pred.y+info;
node.est.Y=node.pred.Y+Info;
% Seems to require the next step for stability in Matlab
node.est.Y=force_sym(node.est.Y);

elseif (obs.report(2:5) == 99999) % Propagate prediction if no observation
    node.est.y=node.pred.y;
    node.est.Y=node.pred.Y;
end

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function constructs the sensor network connectivity topology

function net=set_net_topology(net,nodes,time,METHOD);

globals;

switch upper(METHOD)
case 'FIXED'      % Fixed net
    return;
case 'BROADCAST'
    net=ones(NUM_NODES,NUM_NODES);
    for i=1:NUM_NODES
        net(i,i)=0; % Dynamic net
    end
otherwise
    disp('Warning: Unknown network topology type');
    net=net      % Default is fixed
end

```



```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% A function to "communicate" information between sensor nodes
% In general the philosophy is first to identify the nodes
% in the communication neighbourhood, then to compute a common
% information term locally. Finally the complete estimate is
% communicated and subtraction of common terms is done at the
% receiving node. This approach makes the fusion process robust to
% changes in communication topology.

function node=communicate(Nodes,com_net,node_num)

globals;

% This node
node=Nodes(node_num);

% Find the current communication neighbourhood
num_chan=1;
for i=1:NUM_NODES
    if com_net(node_num,i)==1
        % For connected nodes, compute common prior and estimate
        % The difference is the new information
        node.chan_out(num_chan)=Nodes(i).pred;
        % chan in contains the new observations
        node.chan_in(num_chan).y=Nodes(i).est.y-Nodes(i).pred.y;
        node.chan_in(num_chan).Y=Nodes(i).est.Y-Nodes(i).pred.Y;
        num_chan=num_chan+1;
    end
end

% Note the number of channels for later
nodes.num_chan=num_chan-1;

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function assimilates all the information from each sensor channel.
% The new information is simply the incoming minus the outgoing
% information, or the update minus the common information.

function node=assimilate(node,time);

globals;

% Initialisation
locali=node.est.y-node.pred.y;
localI=node.est.Y-node.pred.Y;
yc=node.pred.y;
Yc=node.pred.Y;

% Generate common prior using covariance intersection
if ~FULL_CON
    for i=1:node.num_chans
        [yc,Yc,omega]=ci_common(node.chan_out(i).y,node.chan_out(i).Y,yc,Yc);
    end
end
% Or through fully connected assumption
node.est.y=yc+locali;
node.est.Y=Yc+localI;

for i=1:node.num_chans
    node.est.y=node.est.y+node.chan_in(i).y;
    node.est.Y=node.est.Y+node.chan_in(i).Y;
end
% Seems to require the next step for stability in Matlab
node.est.Y=force_sym(node.est.Y);
% This function performs the covariance intersect update of two estimates
% (a with covariance A; b with covariance B) and an observation matrix H.
% The output is the output estimate (c with covariance C) and the value
% of omega which minimizes the determinant of C.

function [yc,Yc,omega] = ci_common(ya,Ya,yb,Yb)

```

```

globals;

H = [1 0 0 0;
     0 0 1 0;
     1 0 0 0;
     0 0 1 0];

if Ya(1)~=0 || Yb(1)~=0
    f=inline('1/det(omega*Yai+(1-omega)*Ybi)','omega', 'Yai', 'Ybi');
    Yai=inv(Ya);
    Ybi=inv(Yb);
    YbiH=H'*Ybi;
    YbiHH=YbiH*H;
    omega = fminsearch(f,0.5,[],Yai,Ybi);
    Yci=Yai*omega+YbiHH*(1-omega);
    Yc=inv(Yci);           % New covariance
    yc=Yc*(Yai*ya+YbiH*yb); % New mean
else
    Yc=zeros(XSIZE,XSIZE);
    yc=zeros(XSIZE,1);
    omega = 0;
end

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function displays true target trajectories, platform trajectories,
% target destinations, and sensor observations following a simulation run

function [true,plat,obs,dest]=post_sim(targets,observations,platforms,destinations)

globals;

% Set plotting size
[nsamps,nplatforms]=size(platforms);
[nsamps,ndestinations]=size(destinations);
[nsamps,ntargets]=size(targets);
[nsensors,nsamps]=size(observations);

figure(1)
clf;
hold on
data=zeros(2,nsamps);
title('True Target Motions')
xlabel('x-position (m)')
ylabel('y-position (m)')

% Plot true target trajectories
for n=1:ntargets
    for i=1:nsamps
        data(1,i)=targets(i,n).x;
        data(2,i)=targets(i,n).y;
    end
    true=plot(data(1,:),data(2,:), 'b-');
end

% Plot true platform trajectories
for n=1:nplatforms
    for i=1:nsamps
        data(1,i)=platforms(i,n).x;
        data(2,i)=platforms(i,n).y;
    end
    if n == 1

```

```

    plat=plot(data(1,:),data(2:),'ko');
elseif n == 2
    plat=plot(data(1,:),data(2:),'ko');
elseif n == 3
    plat=plot(data(1,:),data(2:),'ko');
elseif n == 4
    plat=plot(data(1,:),data(2:),'ko');
elseif n == 5
    plat=plot(data(1,:),data(2:),'ko');
elseif n == 6
    plat=plot(data(1,:),data(2:),'r*');
elseif n == 7
    plat=plot(data(1,:),data(2:),'r*');
elseif n == 8
    plat=plot(data(1,:),data(2:),'gh');
elseif n == 9
    plat=plot(data(1,:),data(2:),'bo');
end
end

```

% Plot true destination locations

```

for n=1:ndestinations
    for i=1:nsamps
        data(1,i)=destinations(i,n).x;
        data(2,i)=destinations(i,n).y;
    end
    dest=plot(data(1,:),data(2:),'rh');
end

```

% Plot sensor observations

```

for n=1:nsensors % for all sensors
    for i=1:nsamps % for all time
        report=observations(n,i).report;
        for m=1:ntargets % for all targets
            if report(m,1) > 0 % if they are seen
                [zx,zy,R]=xy_obs(report,m,n); % convert observation to global xy coordinates
                odata(1,m)=zx;
                odata(2,m)=zy;
            end
        end
    end
    if n == 1
        obs=plot(odata(1,:),odata(2:),'k. ');
    elseif n == 2
        obs=plot(odata(1,:),odata(2:),'k. ');
    elseif n == 3

```

```

        obs=plot(odata(1,:),odata(2:),'k.');
```

```

elseif n == 4
    obs=plot(odata(1,:),odata(2:),'k.');
```

```

elseif n == 5
    obs=plot(odata(1,:),odata(2:),'k.');
```

```

elseif n == 6
    obs=plot(odata(1,:),odata(2:),'r.');
```

```

elseif n == 7
    obs=plot(odata(1,:),odata(2:),'r.');
```

```

elseif n == 8
    obs=plot(odata(1,:),odata(2:),'g.');
```

```

elseif n == 9
    obs=plot(odata(1,:),odata(2:),'b.');
```

```

end
end
end

legend([true,plat,obs,dest],'Target      True      Position','Tracking      Stations','Target
Observations','Intended Target Destination')
hold off

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function extract an xy observation from a range-bearing report

function [zx,zy,R]=xy_obs(rep,n,sensor_id)

globals;

sr=sin(rep(n,3));
cr=cos(rep(n,3));
zx=rep(n,4)+rep(n,2)*cr;
zy=rep(n,5)+rep(n,2)*sr;
ROT=[cr -sr; sr cr];
range2=rep(n,2)*rep(n,2);
if sensor_id==1
    sigma=[SIGMA_RANGE_MPA^2 0;0 range2*SIGMA_BEARING_MPA^2];
elseif sensor_id==2
    sigma=[SIGMA_RANGE_MPA^2 0;0 range2*SIGMA_BEARING_MPA^2];
elseif sensor_id==3
    sigma=[SIGMA_RANGE_MPA^2 0;0 range2*SIGMA_BEARING_MPA^2];
elseif sensor_id==4
    sigma=[SIGMA_RANGE_MPA^2 0;0 range2*SIGMA_BEARING_MPA^2];
elseif sensor_id==5
    sigma=[SIGMA_RANGE_MPA^2 0;0 range2*SIGMA_BEARING_MPA^2];
elseif sensor_id==6
    sigma=[SIGMA_RANGE_PCG^2 0;0 range2*SIGMA_BEARING_PCG^2];
elseif sensor_id==7
    sigma=[SIGMA_RANGE_PCG^2 0;0 range2*SIGMA_BEARING_PCG^2];
elseif sensor_id==8
    sigma=[SIGMA_RANGE_PV^2 0;0 range2*SIGMA_BEARING_PV^2];
elseif sensor_id==9
    sigma=[SIGMA_RANGE_MPAC^2 0;0 range2*SIGMA_BEARING_MPAC^2];
end
R=ROT*sigma*ROT';

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function displays the fused target track following a simulation run

function plot_tracks(targets,y,Y,times)

globals;

% Set plotting size
[ntimes,nsensors,xdim1]=size(y);
[ntimes,nsensors,xdim2,xdim3]=size(Y);
odata=zeros(2,ntimes);
title('Target Motions')
xlabel('x-position (m)')
ylabel('y-position (m)')
xlim([150 750])
ylim([280 440])
hold on
warning off;

s=1;
for i=1:ntimes
    P=inv(squeeze(Y(i,s,:,:)));
    x=P*squeeze(y(i,s,:));
    odata(1,i,s)=x(1);
    odata(2,i,s)=x(3);
end

plot(odata(1,:,1),odata(2,:,1),'g-')

hold off

warning on;

```



```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function produces various plots with regards to the SAW parameters

function plot_errors(targets,y,Y,yp,Yp,times,node_num,nfigure,trackstarttime)

globals;

% Initialise
[ntimes,nsensors,xdim1]=size(y);
[ntimes,nsensors,xdim2,xdim3]=size(Y);
data=zeros(10,ntimes);
pdata=zeros(10,ntimes); %predicted data
edata=zeros(10,ntimes); %estimated data
tdata=zeros(10,ntimes); %true data
esttimesigma=zeros(1,ntimes);
ntarget = NUM_TARGETS;
xmin = 4500;
xmax = MAX_TIME;

warning off;
% Calculating and storing the data for each time-step
for i=1:ntimes
    P=inv(squeeze(Y(i,node_num,:,:)));
    x=P*squeeze(y(i,node_num,:));

    Pp=inv(squeeze(Yp(i,node_num,:,:)));
    xp=Pp*squeeze(yp(i,node_num,:));

    data(1,i)=times(i);
    data(2,i)=(x(1)-targets(i,ntarget).x)*180;
    data(3,i)=(x(3)-targets(i,ntarget).y)*180;
    sigma=real(sqrt(P));
    data(4,i)=sigma(1,1)*180;
    data(5,i)=sigma(3,3)*180;
    data(6,i)=(real(sqrt(x(2)*x(2)+x(4)*x(4)))-targets(i,ntarget).vel)*180;
    data(7,i)=atan2(x(4),x(2))-targets(i,ntarget).phi;
    if abs(data(7,i)) > pi
        data(7,i) = 2*pi-abs(data(7,i));
    end
end

```

```

end
data(8,i)=sqrt(sigma(1,1)*sigma(1,1)+sigma(3,3)*sigma(3,3))*180;
data(9,i)=sqrt(sigma(2,2)*sigma(2,2)+sigma(4,4)*sigma(4,4))*180;
data(10,i)=real(sqrt(((x(2)/(x(2)^2 + x(4)^2))^2)*(sigma(4,4)^2)...
+((x(4)/(x(2)^2 + x(4)^2))^2)*(sigma(2,2)^2)));

tdata(1,i)=times(i);
tdata(2,i)=targets(i,ntarget).x*180;
tdata(3,i)=targets(i,ntarget).y*180;
tdata(6,i)=targets(i,ntarget).vel*180;
tdata(7,i)=targets(i,ntarget).phi;

pdata(1,i)=times(i);
pdata(2,i)=xp(1)*180;
pdata(3,i)=xp(3)*180;
psigma=real(sqrt(Pp));
pdata(4,i)=psigma(1,1)*180;
pdata(5,i)=psigma(3,3)*180;
pdata(6,i)=(real(sqrt(xp(2)*xp(2)+xp(4)*xp(4))))*180;
pdata(7,i)=atan2(xp(4),xp(2));
if abs(pdata(7,i)) > pi
    pdata(7,i) = 2*pi-abs(pdata(7,i));
end
pdata(8,i)=sqrt(psigma(1,1)*psigma(1,1)+psigma(3,3)*psigma(3,3))*180;
pdata(9,i)=sqrt(psigma(2,2)*psigma(2,2)+psigma(4,4)*psigma(4,4))*180;
pdata(10,i)=real(sqrt(((xp(2)/(xp(2)^2 + xp(4)^2))^2)*(psigma(4,4)^2)...
+((xp(4)/(xp(2)^2 + xp(4)^2))^2)*(psigma(2,2)^2)));

edata(1,i)=times(i);
edata(2,i)=x(1)*180;
edata(3,i)=x(3)*180;
esigma=real(sqrt(P));
edata(4,i)=esigma(1,1)*180;
edata(5,i)=esigma(3,3)*180;
edata(6,i)=(real(sqrt(x(2)*x(2)+x(4)*x(4))))*180;
edata(7,i)=atan2(x(4),x(2));
if abs(edata(7,i)) > pi
    edata(7,i) = 2*pi-abs(edata(7,i));
end
edata(8,i)=sqrt(esigma(1,1)*esigma(1,1)+esigma(3,3)*esigma(3,3))*180;
edata(9,i)=sqrt(esigma(2,2)*esigma(2,2)+esigma(4,4)*esigma(4,4))*180;
edata(10,i)=real(sqrt(((x(2)/(x(2)^2 + x(4)^2))^2)*(esigma(4,4)^2)...
+((x(4)/(x(2)^2 + x(4)^2))^2)*(esigma(2,2)^2)));

esttimesigma(1,i)=sqrt(edata(8,i)^2+...

```

```

        (kdata(6,i)^2)*(edata(9,i)^2))/...
        edata(6,i);
end

% merging the heading standard deviations for plotting
p=1;q=1;i=1;
for j=1:(ntimes)*2
    if j/2-round(j/2)==0
        combtimedata(1,j)= times(i);
        combddata(8,j)=edata(8,p);
        combddata(9,j)=edata(9,p);
        combddata(10,j)=edata(10,p);
        p=p+1;
        i=i+1;
    else
        combtimedata(1,j)= times(i);
        combddata(8,j)=pdata(8,q);
        combddata(9,j)=pdata(9,q);
        combddata(10,j)=pdata(10,q);
        q=q+1;
    end
end

% Plot
figure(nfigure);
clf;
plot(data(1,:),abs(data(2,:)),'b',data(1,:),data(4,:),'r')
legend('Estimated Error','Standard Deviation');
buf=sprintf('Error between X-component Estimate and Truth');
title(buf)
xlim([xmin xmax])
axis 'auto y'
xlabel('Simulated Time (s)')
ylabel('X error (m)')

figure(nfigure+1);
clf;
plot(data(1,:),abs(data(3,:)),'b',data(1,:),data(5,:),'r')
legend('Estimated Error','Standard Deviation');
buf=sprintf('Error between Y-component Estimate and Truth');
title(buf)
xlim([xmin xmax])
axis 'auto y'
xlabel('Simulated Time (s)')
ylabel('Y error (m)')

```

```

figure(nfigure+2);
clf;
plot(data(1,:),abs(data(6,:)), 'b', data(1,:), data(9,:), 'r')
legend('Estimated Error', 'Standard Deviation');
buf=sprintf('Error between Velocity Estimate and Truth');
title(buf)
xlim([xmin xmax])
axis 'auto y'
xlabel('Simulated Time (s)')
ylabel('Velocity error (m/s)')

figure(nfigure+3);
clf;
plot(data(1,:),abs(data(7,:)), 'b', data(1,:), data(10,:), 'r')
legend('Estimated Error', 'Standard Deviation');
buf=sprintf('Error between Heading Estimate and Truth');
title(buf)
xlim([xmin xmax])
axis 'auto y'
xlabel('Simulated Time (s)')
ylabel('Heading error (rads)')

figure(nfigure+4);
clf;
plot(pdata(1,:),pdata(2,:), 'r', edata(1,:), edata(2,:), 'g', tdata(1,:), tdata(2,:), 'b')
legend('Predicted X', 'Estimated X', 'True X');
buf=sprintf('X position');
title(buf)
xlim([xmin xmax])
axis 'auto y'
xlabel('Simulated Time (s)')
ylabel('X Position (m)')

figure(nfigure+5);
clf;
plot(pdata(1,:),pdata(3,:), 'r', edata(1,:), edata(3,:), 'g', tdata(1,:), tdata(3,:), 'b')
legend('Predicted Y', 'Estimated Y', 'True Y');
buf=sprintf('Y position');
title(buf)
xlim([xmin xmax])
axis 'auto y'
xlabel('Simulated Time (s)')
ylabel('Y Position (m)')

```

```

figure(nfigure+6);
clf;
plot(pdata(1,:),pdata(6,:), 'r', edata(1,:), edata(6,:), 'g', tdata(1,:), tdata(6,:), 'b')
legend('Predicted Velocity', 'Estimated Velocity', 'True Velocity');
buf=sprintf('Velocity');
title(buf)
xlim([xmin xmax])
axis 'auto y'
xlabel('Simulated Time (s)')
ylabel('Velocity (m/s)')

figure(nfigure+7);
clf;
plot(pdata(1,:),pdata(7,:), 'r', edata(1,:), edata(7,:), 'g', tdata(1,:), tdata(7,:), 'b')
legend('Predicted Heading', 'Estimated Heading', 'True Heading');
buf=sprintf('Heading');
title(buf)
xlim([xmin xmax])
axis 'auto y'
xlabel('Simulated Time (s)')
ylabel('Heading (rad)')

figure(nfigure+8);
clf;
plot(pdata(1,:),pdata(8,:), 'r+', edata(1,:), edata(8,:), 'g*', combtimedata(1,:), combdata(8,:), 'b')
legend('Predicted', 'Estimated');
buf=sprintf('Radius Standard Deviations');
title(buf)
xlim([xmin xmax])
axis 'auto y'
xlabel('Simulated Time (s)')
ylabel('Radius (m)')

figure(nfigure+9);
clf;
plot(pdata(1,:),pdata(9,:), 'r+', edata(1,:), edata(9,:), 'g*', combtimedata(1,:), combdata(9,:), 'b')
legend('Predicted', 'Estimated');
buf=sprintf('Velocity Standard Deviations');
title(buf)
xlim([xmin xmax])
axis 'auto y'
xlabel('Simulated Time (s)')
ylabel('Velocity (m/s)')

figure(nfigure+10);

```

```

clf;
plot(pdata(1,:),pdata(10,:),'r+',edata(1,:),edata(10,:),'g*',combtimedata(1,:),combddata(10,:),'b')
legend('Predicted','Estimated');
buf=sprintf('Heading Standard Deviations');
title(buf)
xlim([xmin xmax])
axis 'auto y'
xlabel('Simulated Time (s)')
ylabel('Heading (rad)')

figure(nfigure+11);
clf;
plot(edata(1,:),abs((MAX_TIME-edata(1,:))-kdata(6,:)),'b',edata(1,:),esttimesigma(1,:),'r')
legend('Estimated Error','Standard Deviation');
buf=sprintf('Error between Time Estimate and Truth');
title(buf)
xlim([min(kdata(7,1000:MAX_TIME/10))*DT xmax-1*DT])
axis 'auto y'
xlabel('Simulated Time (s)')
ylabel('Time (s)')

warning on;

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% This function plots the network nodes and communication links

function plot_coms(platforms,com_net)

globals;

[nsamps,nplatforms]=size(platforms);

figure(14)
clf;
hold on
data=zeros(2,nplatforms);
ploc=zeros(2,2);
title('Platforms and Communication Links')
xlabel('x-position (m)')
ylabel('y-position (m)')

for n=1:nplatforms
    data(1,n)=platforms(1,n).x;
    data(2,n)=platforms(1,n).y;
end
plat=plot(data(1,:),data(2,:),'ko');

for i=1:nplatforms
    for j=i:nplatforms
        if com_net(i,j)
            ploc(1,1)=data(1,i);
            ploc(1,2)=data(1,j);
            ploc(2,1)=data(2,i);
            ploc(2,2)=data(2,j);
        end
        links=plot(ploc(1,:),ploc(2,:),'r');
    end
end

legend([plat,links],'Sensors','Communication Links');
hold off

```

D. BLOCK 4. HYBRID DATA & TRACK FUSION ARCHITECTURE USING INFORMATION FILTER & TRACK-TO-TRACK ALGORITHM

% This file executes the Monte-Carlo simulation

% Number of Monte-Carlo simulation runs

totalruns = 50;

% Initialise variables

sumfirstleftTSS(1:totalruns) = NaN;

sumfirstleftTSSntoJIHigh(1:totalruns) = NaN;

sumfirstleftTSSntoJIModerate(1:totalruns) = NaN;

sumfirstleftPL(1:totalruns) = NaN;

sumfirstleftPLntoJIHigh(1:totalruns) = NaN;

sumfirstleftPLntoJIModerate(1:totalruns) = NaN;

% Execute each run by generating new observations or using previous ones

% Produce the fused track for each run

% Then pool the tracks together for analysis

for run =1:totalruns

 buf=sprintf('Run: =%d',run); disp(buf);

 example_sim; %generate new observations

 datastore9(run).observations=observations; % store observations to be executed by other filters

 observations = datastore9(run).observations;

 NEW_RUN = 0;

 run_indep;

 sumfirstleftTSS(run) = firstleftTSS;

 sumfirstleftTSSntoJIHigh(run) = firstleftTSSntoJIHigh;

 sumfirstleftTSSntoJIModerate(run) = firstleftTSSntoJIModerate;

 sumfirstleftPL(run) = firstleftPL;

 sumfirstleftPLntoJIHigh(run) = firstleftPLntoJIHigh;

 sumfirstleftPLntoJIModerate(run) = firstleftPLntoJIModerate;

 sumkdata(run,,:) = kdata;

 sumesttimesigma(run,1,:) = esttimesigma;

 lowest(run) = min(sumkdata(run,7,1000:MAX_TIME/10));

end

% Calculate Probability of Impact

Cxtemp = squeeze(sumkdata(:,8,:));

Cytemp = squeeze(sumkdata(:,9,:));

Xtemp = squeeze(sumkdata(:,10,:));

Ytemp = squeeze(sumkdata(:,11,:));

probability = zeros(1,nt);


```

warning off;
for l = 1:nt

    Cx = diag(Cxtemp(:,l))*(180^2);
    Cy = diag(Cytemp(:,l))*(180^2);
    X = (Xtemp(:,l)- 302.1144444)*180;
    Y = (Ytemp(:,l)- 347.7716667)*180;
    W = ones(totalruns,1);
    sigma_xbar_tgo = inv((W')*inv(Cx)*W);
    sigma_ybar_tgo = inv((W')*inv(Cy)*W);
    sigma = sqrt(sigma_xbar_tgo);
    xbar_tgo = sigma_xbar_tgo*(((W')*inv(Cx)*X)); % Calculate mean impact pt
    ybar_tgo = sigma_ybar_tgo*(((W')*inv(Cy)*Y)); % Calculate mean impact pt
    r_not = sqrt(xbar_tgo^2 + ybar_tgo^2);
    R = 300;

    g = zeros(1,100);
    h = zeros(1,100);
    probtemp = 0;

    recur = 1;
    g_not = exp(-(r_not^2)/(2*sigma^2)); % g_not
    h_not = 1 - exp(-(R^2)/(2*sigma^2)); % h_not
    probtemp_not = g_not*h_not;

    g(1) = (1/1)*((r_not^2)/(2*sigma^2))*g_not;
    h(1) = -(1/factorial(1))*(((R^2)/(2*sigma^2))^(1))*exp(-(R^2)/(2*sigma^2)) + h_not;
    probtemp = probtemp_not + g(1)*h(1);

    while (recur <= 100)
        g(recur+1) = (1/(recur+1))*((r_not^2)/(2*sigma^2))*g(recur);
        h(recur+1) = (-1/factorial(recur+1))*(((R^2)/(2*sigma^2))^(recur+1))*exp(-(R^2)/(2*sigma^2)) + h(recur);
        probtemp = probtemp + g(recur+1)*h(recur+1);
        recur = recur + 1;
    end

    probability(1,l) = probtemp;
end

% Initialise variables
count1 = 0;
count2 = 0;
count3 = 0;
count4 = 0;

```

```

count5 = 0;
count6 = 0;
SsumfirstleftTSS = 0;
SsumfirstleftTSSntoJIHigh = 0;
SsumfirstleftTSSntoJIModerate = 0;
SsumfirstleftPL = 0;
SsumfirstleftPLntoJIHigh = 0;
SsumfirstleftPLntoJIModerate = 0;

% Take the average of the pooled tracks
for m = 1:totalruns
    if ~isnan(sumfirstleftTSS(m))
        count1 = count1+1;
        SsumfirstleftTSS=SsumfirstleftTSS+sumfirstleftTSS(m);
    end

    if ~isnan(sumfirstleftTSSntoJIHigh(m))
        count2 = count2+1;

SsumfirstleftTSSntoJIHigh=SsumfirstleftTSSntoJIHigh+sumfirstleftTSSntoJIHigh(m);
    end

    if ~isnan(sumfirstleftTSSntoJIModerate(m))
        count3 = count3+1;

SsumfirstleftTSSntoJIModerate=SsumfirstleftTSSntoJIModerate+sumfirstleftTSSntoJIModerate(m);
    end

    if ~isnan(sumfirstleftPL(m))
        count4 = count4+1;
        SsumfirstleftPL=SsumfirstleftPL+sumfirstleftPL(m);
    end

    if ~isnan(sumfirstleftPLntoJIHigh(m))
        count5 = count5+1;
        SsumfirstleftPLntoJIHigh=SsumfirstleftPLntoJIHigh+sumfirstleftPLntoJIHigh(m);
    end

    if ~isnan(sumfirstleftPLntoJIModerate(m))
        count6 = count6+1;

SsumfirstleftPLntoJIModerate=SsumfirstleftPLntoJIModerate+sumfirstleftPLntoJIModerate(m);
    end

```

end

```
meanfirstleftTSS = SsumfirstleftTSS/count1;
meanfirstleftTSSntoJIHigh = SsumfirstleftTSSntoJIHigh/count2;
meanfirstleftTSSntoJIModerate = SsumfirstleftTSSntoJIModerate/count3;
meanfirstleftPL = SsumfirstleftPL/count4;
meanfirstleftPLntoJIHigh = SsumfirstleftPLntoJIHigh/count5;
meanfirstleftPLntoJIModerate = SsumfirstleftPLntoJIModerate/count6;
meantruetime data = mean(sumkdata(:,1,:));
meandistuncert = mean(sumkdata(:,2,:));
meandistapart = mean(sumkdata(:,3,:));
meanesttime data = mean(sumkdata(:,6,:));
```

% Calculate the sample variance of the estimated time to impact

```
for p = 1:nt
    sumtimediff=0;
    timediff=0;
    for q = 1:totalruns
        timediff = (sumkdata(q,6,p)-meanesttime data(p))^2;
        sumtimediff=sumtimediff+timediff;
    end
    meansampletimesigma(p) = sqrt(sumtimediff /(totalruns-1));
end
```

```
figure(30);
clf;
plot(meantruetime data(:),meandistuncert(:),'b',meantruetime data(:),meandistapart(:),'g')
legend('Distance Uncertainty','Distance Apart');
buf=sprintf('Estimated End Point');
title(buf)
xlim([min(lowest)*DT MAX_TIME-1*DT])
ylim([0 1000])
xlabel('Simulated Time (s)')
ylabel('Distance (m)')
```

```
figure(31);
clf;
plot(meantruetime data(:),abs((MAX_TIME-meantruetime data(:))-
meanesttime data(:)),'b',meantruetime data(:),meansampletimesigma,'r')
legend('Estimated Error','Standard Deviation');
buf=sprintf('Error between Time Estimate and Truth');
title(buf)
xlim([min(lowest)*DT MAX_TIME-1*DT])
ylim([0 100])
xlabel('Simulated Time (s)')
```

```

ylabel('Time (s)')

figure(32);
clf;
plot(meantruetimedata(:),probability(:),'k')
legend('Probability of Impact');
buf=sprintf('Probability of Impact');
title(buf)
xlim([min(lowest)*DT MAX_TIME-1*DT])
ylim([0 1])
xlabel('Simulated Time (s)')
ylabel('Probability of Impact')

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This file runs the track-to-track algorithm. The default condition is to
% do a new run every time. There is also the option to use previous
% observation runs so that different filters can be compared.

globals;
ginit;

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

% If we choose to use previous data,
% then the existence of various variables needs to be checked
if ~NEW_RUN
    if ~exist('observations','var') | ~exist('targets','var') | ...
        ~exist('platforms','var') | ~exist('times','var')
        error('No existing Data to perform simulation');
    end
    [num_nodes,num_times]=size(observations);
    nt=length(times);
    if (num_nodes ~= NUM_NODES) | (num_times ~= nt)
        error('Observation Record is of incorrect dimension');
    end
    if nt ~= length(targets)
        error('Target set is of incorrect dimension');
    end
    [num_times,num_plat]=size(platforms);
    if (num_nodes ~= num_plat) | (num_times ~= nt)
        error('Platform set is of incorrect dimension');
    end
    buf=sprintf('Old Run Commencing with %d nodes and %d time
steps\n',num_nodes,num_times);
    disp(buf);
end

% If we are doing a new run then simulate the target and platform motions
if NEW_RUN

```

```

clear all;
globals;
ginit;
NEW_RUN=1;
times=0:DT:MAX_TIME;
num_times=length(times);
buf=sprintf('New Run Commencing with %d nodes and %d time
steps\n',NUM_NODES,num_times-1);
disp(buf);

% This code currently apply to only one target
% One target considerably simplifies computation
buf=sprintf('Generating Target Set\n'); disp(buf);
targets=generate__targets(times);

% Simulate platforms motion, assuming they are known
buf=sprintf('Generating Platform Trajectories\n'); disp(buf);
platforms=generate__platforms(times);

% Generate all target's attack destinations
destinations=generate__destinations(times);
else
% If we are using previous data set
% then there is no need to simulate targets or platforms
ginit;
clear Nodes;
end

% In every case, the sensors which sit on the platforms are defined
% Also, this allows the filters to be changed between runs
buf=sprintf('Defining Sensors\n'); disp(buf);
filter=get_filter_params;

for i=1:NUM_NODES
    Nodes(i)=make__sensor(i,filter);
end

% Establish initial topology
com_net=eye(NUM_NODES);

% Establish the time base
ntimes=length(times);

% Define space to record results
Results_y=zeros(ntimes,NUM_NODES,filter.XDIM);

```

```
Results_Y=zeros(ntimes,NUM_NODES,filter.XDIM,filter.XDIM);
```

```
%-----  
%  
% START OF FILTER LOOPS HERE  
%  
%-----
```

```
% Run loop for all sensors at each time.
```

```
% This is as close as we can get to running this in parallel
```

```
for n=1:ntimes  
    kdata(1,n)=0;  
    kdata(2,n)=0;  
    kdata(3,n)=0;  
    kdata(4,n)=0;  
    kdata(5,n)=0;  
    kdata(6,n)=0;  
    kdata(7,n)=99999;  
    kdata(8,n) = 0;  
    kdata(9,n) = 0;  
    kdata(10,n) = 0;  
    kdata(11,n) = 0;  
    leftTSS(n)=NaN;  
    leftTSSntoJIHigh(n)=NaN;  
    leftTSSntoJIModerate(n)=NaN;  
    leftPL(n)=NaN;  
    leftPLntoJIHigh(n)=NaN;  
    leftPLntoJIModerate(n)=NaN;
```

```
end
```

```
distapart = 0;  
distuncert = 0;  
disttoeach = 0;  
speedtoeach = 0;  
timetoeach = 0;
```

```
for i=1:ntimes  
    startplot=99999;  
    buf=sprintf('Step: time=%d',i-1); disp(buf);  
    % Time in this loop  
    t=times(i);
```

```
% Generate an observation for each node at this time
```

```
for s=1:NUM_NODES
```

```

if NEW_RUN % new observation simulation required
    obs(s).report=sensor__report(Nodes(s),platforms,targets,s,i);
    observations(s,i)=obs(s); % record observations for plotting
else % use old observation data
    obs(s)=observations(s,i);
end
end

% First, do a local prediction
for s=1:NUM_NODES
    Nodes(s)=local_predict(Nodes(s),t);
end

% Second, a local update generating y tilde at each site
for s=1:NUM_NODES
    Nodes(s)=local_update(Nodes(s),obs(s),s);
end

% Record results
for s=1:NUM_NODES
    Results_y(i,s,:)=Nodes(s).est.y;
    Results_Y(i,s,:)=Nodes(s).est.Y;
    Results_yp(i,s,:)=Nodes(s).pred.y;
    Results_Yp(i,s,:)=Nodes(s).pred.Y;
end

% Carry out track-to-track association
global trackest1;
global trackestcov1;
global trackpred1;
global trackpredcov1;
global trackest2;
global trackestcov2;
global trackpred2;
global trackpredcov2;
chi = chi2inv(alpha,dof);
countest = 0; countpred = 0;
trackpredcov2=zeros(XSIZE, XSIZE);
trackpred2=zeros(XSIZE, 1);

warning off;
% carry out track-to-track fusion for predicted and estimated tracks
for s=1:NUM_NODES
    trackpredcov = inv(Nodes(s).pred.Y);
    trackpred = trackpredcov*Nodes(s).pred.y;

```



```

    if ~isinf(trackpredcov)
        countpred = countpred+1;
    end
end
if countpred == 1
    for s=1:NUM_NODES
        trackpredcov = inv(Nodes(s).pred.Y);
        trackpred = trackpredcov*Nodes(s).pred.y;
        if ~isinf(trackpredcov)
            trackpredcov1 = inv(Nodes(s).pred.Y);
            trackpred1 = trackpredcov1*Nodes(s).pred.y;
        end
    end
elseif countpred == 0
    trackpredcov1 = inf(XSIZE, XSIZE);
    trackpred1 = nan(XSIZE,1);
elseif countpred > 1
    for s=1:NUM_NODES
        if countpred > 0
            trackpredcov = inv(Nodes(s).pred.Y);
            trackpred = trackpredcov*Nodes(s).pred.y;
            if ~isinf(trackpredcov)
                trackpredcov1 = inv(Nodes(s).pred.Y);
                trackpred1 = trackpredcov1*Nodes(s).pred.y;
                countpred = countpred - 1;
                if s+1 <= NUM_NODES && countpred > 0
                    for q = s+1:NUM_NODES
                        trackpredcovv = inv(Nodes(q).pred.Y);
                        trackpredv = trackpredcovv*Nodes(q).pred.y;
                        if ~isinf(trackpredcovv)
                            trackpredcov2 = inv(Nodes(q).pred.Y);
                            trackpred2 = trackpredcov2*Nodes(q).pred.y;
                            countpred = countpred - 1;
                        end
                    end
                    if countpred >= 0
                        tracksumpredcov = trackpredcov1 + trackpredcov2;
                        trackpreddiff = trackpred1 - trackpred2;
                        if real(abs(trackpreddiff*inv(tracksumpredcov)*trackpreddiff)) <=
chi
                                trackpred1 = trackpredcov2*inv(tracksumpredcov)*trackpred1
+...
                                trackpredcov1*inv(tracksumpredcov)*trackpred2;
                                trackpredcov1
                                =
trackpredcov1*inv(tracksumpredcov)*trackpredcov2;
else

```

```

        trackpred1 = trackest1;
        trackpredcov1 = trackestcov1;
    end
end
end
end
end
end
end
end
TrackPredResults_y(i,:)=trackpred1;
TrackPredResults_Y(i,,:)=trackpredcov1;

for s=1:NUM_NODES
    trackestcov = inv(Nodes(s).est.Y);
    trackest = trackestcov*Nodes(s).est.y;
    if ~isinf(trackestcov)
        countest = countest+1;
    end
end
if countest == 1
    for s=1:NUM_NODES
        trackestcov = inv(Nodes(s).est.Y);
        trackest = trackestcov*Nodes(s).est.y;
        if ~isinf(trackestcov)
            trackestcov1 = inv(Nodes(s).est.Y);
            trackest1 = trackestcov1*Nodes(s).est.y;
        end
    end
elseif countest == 0
    trackestcov1 = inf(XSIZE, XSIZE);
    trackest1 = nan(XSIZE,1);
elseif countest > 1
    for s=1:NUM_NODES
        if countest > 0
            trackestcov = inv(Nodes(s).est.Y);
            trackest = trackestcov*Nodes(s).est.y;
            if ~isinf(trackestcov)
                trackestcov1 = inv(Nodes(s).est.Y);
                trackest1 = trackestcov1*Nodes(s).est.y;
            end
            countest = countest - 1;
            if s+1 <= NUM_NODES && countest > 0
                for q = s+1:NUM_NODES
                    trackestcovv = inv(Nodes(q).est.Y);
                    trackestv = trackestcovv*Nodes(q).est.y;

```

```

        if ~isinf(trackestcovv)
            trackestcov2 = inv(Nodes(q).est.Y);
            trackest2 = trackestcov2*Nodes(q).est.y;
            countest = countest - 1;
        end
        if countest >= 0
            tracksumestcov = trackestcov1 + trackestcov2;
            trackestdiff = trackest1 - trackest2;
            if real(abs(trackestdiff*inv(tracksumestcov)*trackestdiff)) <= chi
                trackest1 = trackestcov2*inv(tracksumestcov)*trackest1 + ...
                    trackestcov1*inv(tracksumestcov)*trackest2;
                trackestcov1 = trackestcov1*inv(tracksumestcov)*trackestcov2;
            end
        end
    end
end
end
end
end
end
end
TrackEstResults_y(i,:)=trackest1;
TrackEstResults_Y(i,,:)=trackestcov1;

P=trackestcov1;
if P ~= inf
    % Calculate the position uncertainty of the target
    % at present location
    x=trackest1;
    sigma=sqrt(P);
    xuncert = sigma(1,1);
    yuncert = sigma(3,3);
    distuncert = sqrt(xuncert^2 + yuncert^2);
    b = circle([x(1), x(3)],distuncert);

    % Project the position uncertainty of the target
    % at final destination
    FF=[1 (ntimes-i)*DT 0 0;
        0 1 0 0;
        0 0 1 (ntimes-i)*DT;
        0 0 0 1];

    GG= [1 0 0 0;
        0 1 0 0;
        0 0 1 0;
        0 0 0 1];

```

```

QQ= [((ntimes-i)*DT)^2*SIGMA_XDOT^2 0 0 0;
      0 SIGMA_XDOT^2 0 0;
      0 0 ((ntimes-i)*DT)^2*SIGMA_YDOT^2 0;
      0 0 0 SIGMA_YDOT^2];

MM=(inv(FF))'*(inv(P))*(inv(FF));
SSigma=GG'*MM*GG+inv(QQ);
OOmega=MM*GG*inv(SSigma);
projYpred=MM-(OOmega*SSigma*OOmega');
projYpred=force_sym(projYpred);
projP = inv(projYpred);
projPsigma = sqrt(projP);
projxuncert = projPsigma(1,1);
projyuncert = projPsigma(3,3);
projdistuncert = sqrt(projxuncert^2 + projyuncert^2);
projypred=(eye(4,4)-(OOmega*GG'))*(inv(FF))'*(inv(P)*x);
projstate= projP*projypred;

% Check if SAW has entered Port Limits
if min(inpolygon (b.X, b.Y, TSS_X, TSS_Y))== 0 &&...
    min(inpolygon (b.X, b.Y, PL_X, PL_Y)) == 0
    buf=sprintf('Warning: Entered Port Limits at Timestep=%d',i-1); disp(buf);

% Check if SAW is turning towards Jurong Island
dittoreach = sqrt( abs(x(1)-302.1144444)^2+...
    abs(x(3)-347.7716667)^2);
speedtoreach = sqrt(x(2)^2+x(4)^2);
timetoreach = dittoreach/speedtoreach;
% Calculate the time target left Port Limits
leftPL(i) = MAX_TIME-timetoreach;
% Calculate the distance between the projected target end point
% and Jurong Island
distapart = sqrt((projstate(1) - 302.1144444)^2+(projstate(3) - 347.7716667)^2);

% Threat level assessment
% Send warning if target is within 0.5nm to Jurong Island
if distapart < 5 && MAX_TIME - (i-1)*DT >= 0
    buf=sprintf('Warning: Heading Towards Jurong Island. Time to Impact = %.2f
sec',timetoreach); disp(buf);
    if projdistuncert > distapart
        buf=sprintf('Warning: High Confidence of Impact');disp(buf);
        startplot=i-1;
        leftPLntoJIHigh(i) = MAX_TIME-timetoreach;
    else

```

```

        buf=sprintf('Warning: Moderate Confidence of Impact');disp(buf);
        startplot=i-1;
        leftPLntoJIModerate(i) = MAX_TIME-timetoreach;
    end
end

% Check if SAW has left Traffic Separation Scheme
elseif min(inpolygon (b.X, b.Y, TSS_X, TSS_Y))== 0 &&...
    min(inpolygon (b.X, b.Y, PL_X, PL_Y)) == 1
    buf=sprintf('Warning: Left Traffic Separation Scheme at Timestep=%d',(i-1));
disp(buf);

% Check if SAW is turning towards Jurong Island
dittoreach = sqrt( abs(x(1)-302.1144444)^2+...
    abs(x(3)-347.7716667)^2);
speedtoreach = sqrt(x(2)^2+x(4)^2);
timetoreach = dittoreach/speedtoreach;
% Calculate the time target left TSS
leftTSS(i) = MAX_TIME-timetoreach;

% Calculate the distance between the projected target
% end pointand Jurong Island
distapart = sqrt((projstate(1) - 302.1144444)^2 + ...
    (projstate(3) - 347.7716667)^2);

% Threat level assessment
% Send warning if target is within 0.5nm to Jurong Island
if distapart < 5 && MAX_TIME - (i-1)*DT >= 0
    buf=sprintf('Warning: Heading Towards Jurong Island. Time to Impact = %.2f
sec',timetoreach); disp(buf);
    if projdistuncert > distapart
        buf=sprintf('Warning: High Confidence of Impact');disp(buf);
        startplot=i-1;
        leftTSSntoJIHigh(i) = MAX_TIME-timetoreach;
    else
        buf=sprintf('Warning: Moderate Confidence of Impact');disp(buf);
        startplot=i-1;
        leftTSSntoJIModerate(i) = MAX_TIME-timetoreach;
    end
end
end

kdata(1,i)=times(i);
kdata(2,i)=projdistuncert*180;
kdata(3,i)=distapart*180;

```

```

kdata(4,i)=disttoreach*180;
kdata(5,i)=speedtoreach*180;
kdata(6,i)=timetoreach;
kdata(7,i)=startplot;
kdata(8,i)=projP(1,1);
kdata(9,i)=projP(3,3);
kdata(10,i)=projstate(1);
kdata(11,i)=projstate(3);

    end
end

disp('Plotting Data...');

figure(20);
clf;
plot(kdata(1,:),kdata(2,:), 'b',kdata(1,:),kdata(3,:), 'g')
legend('Distance Uncertainty','Distance Apart');
buf=sprintf('Estimated End Point');
title(buf)
xlim([min(kdata(7,1000:MAX_TIME/DT))*DT MAX_TIME-1*DT])
axis 'auto y'
xlabel('Simulated Time (s)')
ylabel('Distance (m)')

figure(21);
clf;
plot(kdata(1,:),kdata(4,:), 'b',kdata(1,:),kdata(5,:), 'g',kdata(1,:),kdata(6,:), 'r')
legend('Distance to Impact','Speed to Impact', 'Time to Impact');
buf=sprintf('Projected Distance, Speed and Time to Impact');
title(buf)
xlim([min(kdata(7,1000:MAX_TIME/DT))*DT MAX_TIME-1*DT])
axis 'auto y'
xlabel('Simulated Time (s)')

% Record the times the target crossed TSS, PL, & corresponding threat level
firstleftTSS = min(leftTSS(1000:MAX_TIME/DT));
firstleftTSSntoJIHigh = min(leftTSSntoJIHigh(1000:MAX_TIME/DT));
firstleftTSSntoJIModerate = min(leftTSSntoJIModerate(1000:MAX_TIME/DT));
firstleftPL = min(leftPL(1000:MAX_TIME/DT));
firstleftPLntoJIHigh = min(leftPLntoJIHigh(1000:MAX_TIME/DT));
firstleftPLntoJIModerate = min(leftPLntoJIModerate(1000:MAX_TIME/DT));

time = 0;
for i=1:ntimes

```

```

P=squeeze(TrackEstResults_Y(i,:,:));
if P == inf
time = i;
end
end
warning on;

post__sim(targets,observations,platforms,destinations);
plot__tracks(targets,Results_y,Results_Y,Results_yp,Results_Yp,...
            TrackEstResults_y,TrackEstResults_Y,TrackPredResults_y,...
            TrackPredResults_Y,times)
plot__errors(targets,TrackEstResults_y,TrackEstResults_Y,...
            TrackPredResults_y,TrackPredResults_Y,times,2,time)
plot__coms(platforms,com_net)
% This file defines the global variables

% General information
global TSS_X;      % Traffic Separation Scheme x-coordinates
global TSS_Y;      % Traffic Separation Scheme y-coordinates
global PL_X;       % Port Limits x-coordinates
global PL_Y;       % Port Limits y-coordinates
global DT;         % simulation time-step
global MAX_TIME;   % maximum simulation time

% Target definitions
global TARGET_TYPE; % defines type of target (xy or V phi)
global NUM_TARGETS; % number of targets to be simulated

% Platform definitions
global PLATFORM;    % Platform data [x0, y0, phi0, vel]
global NUM_PLATFORMS; % number of platforms
global MAX_RANGE;   % maximum observable range
global SIGMA_XDOT;
global SIGMA_YDOT;

% Destination definitions
global DESTINATION; % Destination data [x0, y0, phi0, vel]
global NUM_DESTINATIONS; % number of destinations

% Communications definitions
global NUM_NODES;   % number of sensor nodes
global NUM_CHANS;   % number of channels per node
global FULL_CON;    % connectivity

% State and Sensor definitions

```

```

global SIGMA_Q;      % process noise standard deviation for feature model
global SIGMA_RANGE_MPA; % Range observation noise
global SIGMA_BEARING_MPA; % Bearing observation noise
global SIGMA_RANGE_MPAC; % Range observation noise
global SIGMA_BEARING_MPAC; % Bearing observation noise
global SIGMA_RANGE_PV; % Range observation noise
global SIGMA_BEARING_PV; % Bearing observation noise
global SIGMA_RANGE_PCG; % Range observation noise
global SIGMA_BEARING_PCG; % Bearing observation noise
global Rfilter_MPA; % observation noise covariance for MPA
global Rfilter_MPAC; % observation noise covariance for MPAC
global Rfilter_PV; % observation noise covariance for PV
global Rfilter_PCG; % observation noise covariance for PCG
global F; % state transition equation
global XSIZE; % state dimension
global ZSIZE; % observation dimension

% Results "database"
global kdata;
global esttimesigma;
global startplot;
global firstleftTSS;
global firstleftTSSntoJIHigh;
global firstleftTSSntoJIModerate;
global firstleftPL;
global firstleftPLntoJIHigh;
global firstleftPLntoJIModerate;

% Track association test parameters
global alpha; % level of confidence for track association
global dof; % degrees of freedom for Chi-statistics

% This file initializes the global variables

% General polygon for Traffic Separation Scheme (TSS)
X = [738.9629; 700.8405; 503.0354; 343.3085; 301.0754; 188.7756; 142.3798;
291.7565; 356.6971; 474.1785; 703.9221; 738.9538; 738.9629];
Y = [443.8223; 412.1040; 377.3690; 303.7337; 330.3691; 359.0929; 316.1193;
230.0498; 289.3964; 351.7903; 379.3511; 408.0001; 443.8223];
TSS_X = X;
TSS_Y = Y;

% General polygon for Port Limits, estimated 0.5 mile beyond the TSS
[PL_X, PL_Y] = bufferm2('xy',X,Y,5,'out');

```



```

MAX_TIME=11500;          % simulation duration
DT=10;                   % simulation time-step

% Target definitions
NUM_TARGETS=1;           % number of targets to be tracked

% Platform definitions
NUM_PLATFORMS=9;

% Destination definitions
NUM_DESTINATIONS=1;

% Number of sensor nodes
NUM_NODES=NUM_PLATFORMS;
NUM_CHANS=0;              % no data sharing between nodes
FULL_CON=0;              % no connectivity between all nodes

% State and Sensor definitions
SIGMA_RANGE_MPA=5;        % 0.5nm range error equivalent
SIGMA_BEARING_MPA=0.0525; % 3 degrees bearing error in radians
SIGMA_RANGE_MPAC=6;       % 0.6nm range error equivalent
SIGMA_BEARING_MPAC=0.0875; % 5 degrees bearing error in radians
SIGMA_RANGE_PV=3;         % 0.3nm range error equivalent
SIGMA_BEARING_PV=0.0525;  % 3 degrees bearing error in radians
SIGMA_RANGE_PCG=4;        % 0.4nm range error equivalent
SIGMA_BEARING_PCG=0.0875; % 5 degrees bearing error in radians
Rfilter_MPA = [SIGMA_RANGE_MPA^2 0;0 SIGMA_RANGE_MPA^2];
Rfilter_MPAC = [SIGMA_RANGE_MPAC^2 0;0 SIGMA_RANGE_MPAC^2];
Rfilter_PV = [SIGMA_RANGE_PV^2 0;0 SIGMA_RANGE_PV^2];
Rfilter_PCG = [SIGMA_RANGE_PCG^2 0;0 SIGMA_RANGE_PCG^2];
XSIZE=4;                 % number of state components
ZSIZE=2;                 % number of sensor measurement components
SIGMA_XDOT=0.00286;      % std dev of 1knot in the x-direction
SIGMA_YDOT=0.00286;      % std dev of 1knot in the y-direction

% Results "database"
kdata = [7 MAX_TIME/DT+1];

% Track association test parameters
alpha = 0.99;
dof = NUM_NODES;
function [latb,lonb] = bufferm2(varargin) %lat,lon,dist,direction,npts,outputformat)
%BUFFERM2 Computes buffer zone around a polygon
%
% [latb,lonb] = bufferm2(lat,lon,dist,direction)

```

```

% [latb,lonb] = bufferm2(lat,lon,dist,direction,npts)
% [latb,lonb] = bufferm2(lat,lon,dist,direction,npts,outputformat)
% [xb, yb] = bufferm2('xy',x,y,dist,direction,npts,outputformat)
%
% This function was originally designed as a replacement for the Mapping
% Toolbox function bufferm, which calculates a buffer zone around a
% polygon. The original bufferm function had some serious bugs that could
% result in incorrect buffer results and/or errors, and was also very slow.
% As of R2006b, those bugs have been fixed. However, this version still
% maintains a few advantages over the original:
%
% - Can be applied to polygons in either geographical space (as in
%   bufferm) or in cartesian coordinates.
%
% - Better treatment of polygon holes. The original function simply
%   filled in all holes; this version trims or pads holes according to the
%   buffer width given.
%
% Input and output format is identical to bufferm unless the 'xy' option is
% specified, so it can be used interchangeably.
%
% Input variables:
%
% lat:      Latitude values defining the polygon to be buffered.
%           This can be either a NaN-delimited vector, or a cell
%           array containing individual polygonal contours (each of
%           which is a vector). External contours should be listed
%           in a clockwise direction, and internal contours (holes)
%           in a counterclockwise direction.
%
% lon:      Longitude values defining the polygon to be buffered.
%           Same format as lat.
%
% dist:     Width of buffer, in degrees of arc along the surface
%           (unless 'xy' is used, in which case units correspond to
%           x-y coordinates)
%
% direction: 'in' or 'out'
%
% npts:     Number of points used to construct the circles around
%           each polygon vertex. If omitted, default is 13.
%
% outputformat: 'vector' (NaN-delimited vectors), 'cutvector'
%              (NaN-clipped vectors with cuts connecting holes to the
%              exterior of the polygon), or 'cell' (cell arrays in

```

```

%           which each element of the cell array is a separate
%           polygon), defining format of output. If omitted,
%           default is 'vector'.
%
% 'xy':      If first input is 'xy', then data will be assumed to
%           lie on a cartesian plane rather than on a sphere. Use
%           x and y coordinates as first two inputs rather than lat
%           and lon. Units of x, y, and distance should be the
%           same.
%
% Output variables:
%
% latb:      Latitude values for buffer polygon
%
% lonb:      Longitude values for buffer polygon
%
% Example:
%
% load conus
% tol = 0.1; % Tolerance for simplifying polygon outlines
% [reducedlat, reducedlon] = reducem(gtlakelat, gtlakelon, tol);
% dist = 1; % Buffer distance in degrees
% [latb, lonb] = bufferm2(reducedlat, reducedlon, dist, 'out');
% figure('Renderer','painters')
% usamap({'MN','NY'})
% geoshow(latb, lonb, 'DisplayType', 'polygon', 'FaceColor', 'yellow')
% geoshow(gtlakelat, gtlakelon,...
%         'DisplayType', 'polygon', 'FaceColor', 'blue')
% geoshow(uslat, uslon)
% geoshow(statelat, statelon)
%
% See also:
%
% bufferm, polybool

% Copyright 2010 Kelly Kearney

%-----
% Check input
%-----

error(nargchk(3,7,nargin));

% Determine if geographic or cartesian

```

```

if ischar(varargin{1}) && strcmp(varargin{1}, 'xy')
    geo = false;
    param = varargin(2:end);
else
    geo = true;
    param = varargin;
end

% Set defaults if not provided as input

nparam = length(param);

if geo
    [lat, lon, dist] = deal(param{1:3});
else
    [lon, lat, dist] = deal(param{1:3}); % lon = x, lat = y for mental clarity, will switch
    back at end
end

if nparam < 4
    direction = 'out';
else
    direction = param{4};
end

if nparam < 5
    npts = 13;
else
    npts = param{5};
end

if nparam < 6
    outputformat = 'vector';
else
    outputformat = param{6};
end

% Check format and dimensions of input

if ~ismember(direction, {'in', 'out'})
    error('Direction must be either "in" or "out".');
end

if ~ismember(outputformat, {'vector', 'cutvector', 'cell'})
    error('Unrecognized output format flag.');
```

```

end

if ~isnumeric(dist) || numel(dist) > 1
    error('Distance must be a scalar.')
end

if ~isnumeric(npts) || numel(npts) > 1
    error('Number of points must be a scalar.')
end

if iscell(lat)
    for il = 1:numel(lat)
        if ~isvector(lat{il}) || ~isvector(lon{il}) || ~isequal(length(lat{il}), length(lon{il}))
            error('Lat (or x) and lon (or y) must be vectors or cells of vectors with identical dimensions');
        end
        lat{il} = lat{il}(:);
        lon{il} = lon{il}(:);
    end
else
    if ~isvector(lat) || ~isvector(lon) || ~isequal(length(lat), length(lon))
        error('Lat (or x) and lon (or y) must be vectors or cells of vectors with identical dimensions');
    end
    lat = lat(:);
    lon = lon(:);
end

%-----
% Split polygon(s) into
% separate faces
%-----

if iscell(lat)
    [lat, lon] = polyjoin(lat, lon); % In case multiple faces in one cell.
end

[latcells, loncells] = polysplit(lat, lon);

%-----
% Create buffer shapes
%-----

plotflag = 0;

```

```

if plotflag
    Plt.x = lon;
    Plt.y = lat;

end

latcrall = cell(0);
loncrall = cell(0);

for ipoly = 1:length(latcells)

    % Circles around each vertex

    if geo
        [latc, lonc] = calccircgeo(latcells{ipoly}, loncells{ipoly}, dist, npts);
    else
        [lonc, latc] = calccirccart(loncells{ipoly}, latcells{ipoly}, dist, npts);
    end

    % Rectangles around each edge

    if geo
        [latr, lonr] = calcrecgeo(latcells{ipoly}, loncells{ipoly}, dist);
    else
        [lonr, latr] = calcreccart(loncells{ipoly}, latcells{ipoly}, dist);
    end

    % Union of circles and rectangles

    if plotflag
        Plt.rectx = lonr;
        Plt.recty = latr;
        Plt.circx = lonc;
        Plt.circy = latc;
    end

    [latc, lonc] = multipolyunion(latc, lonc);
    [latr, lonr] = multipolyunion(latr, lonr);

    if plotflag
        Plt.rectcombox = lonr;
        Plt.rectcomboy = latr;
        Plt.circcombox = lonc;
        Plt.circcomboy = latc;
    end
end

```

```

end

[loncr, latcr] = polybool('+', lonr, latr, lonc, latc);

% Union of new circle/rectangle combo with that from other faces

[loncrall, latcrall] = polybool('+', loncrall, latcrall, loncr, latcr);

% Plotting (for debugging only)

if plotflag

    Plt.allx = loncrall;
    Plt.ally = latcrall;

    if ipoly == 1
        figure;
        plot(Plt.x, Plt.y, 'k', 'linewidth', 2);
        hold on
    end

    plot(cat(2, Plt.rectx{:}), cat(2, Plt.recty{:}), 'b');
    plot(cat(2, Plt.circx{:}), cat(2, Plt.circy{:}), 'r');
    plot(Plt.allx{1}, Plt.ally{1}, 'g', 'linewidth', 2);

end

end

%-----
% Calculate union/difference
%-----

switch direction
case 'out'
    [lonb, latb] = polybool('+', loncells, latcells, loncrall, latcrall);
case 'in'
    [lonb, latb] = polybool('-', loncells, latcells, loncrall, latcrall);
end

if plotflag
    [Plt.yfinal, Plt.xfinal] = polyjoin(latb, lonb);
    plot(Plt.xfinal, Plt.yfinal, 'linestyle', '--', 'color', [0 .5 0], 'linewidth', 2);
end

```

```

%-----
% Reformat output
%-----

if ~geo
    y = latb; % Switch, since cartesion uses opposite order
    x = lonb;
    latb = x;
    lonb = y;
end

switch outputformat
    case 'vector'
        [latb, lonb] = polyjoin(latb, lonb);
    case 'cutvector'
        [latb, lonb] = polycut(latb, lonb);
    case 'cell'
end

%*****
%*****

function [latc, lonc] = calccircgeo(lat, lon, radius, npts)
% lat and lon: n x 1 vectors
% radius: scalar

radius = ones(length(lat),1) * radius;
[latc, lonc] = scircle1(lat, lon, radius, [], [], [], npts);
latc = num2cell(latc, 1);
lonc = num2cell(lonc, 1);

function [latr, lonr] = calcrecgeo(lat, lon, halfwidth)
% lat and lon: n x 1 vectors
% halfwidth: scalar

range = halfwidth * ones(length(lat)-1, 1);

az = azimuth(lat(1:end-1), lon(1:end-1), lat(2:end), lon(2:end));

[latbl1,lonbl1] = reckon(lat(1:end-1), lon(1:end-1), range, az-90);
[latbr1,lonbr1] = reckon(lat(1:end-1), lon(1:end-1), range, az+90);
[latbl2,lonbl2] = reckon(lat(2:end), lon(2:end), range, az-90);
[latbr2,lonbr2] = reckon(lat(2:end), lon(2:end), range, az+90);

```



```

latr = [latbl1 latbl2 latbr2 latbr1 latbl1]';
lonr = [lonbl1 lonbl2 lonbr2 lonbr1 lonbl1]';
latr = num2cell(latr, 1);
lonr = num2cell(lonr, 1);

function [latu, lonu] = multipolyunion(lat, lon)
% lat and lon are n x 1 cell arrays of vectors

latu = lat{1};
lonu = lon{1};

for ip = 2:length(lat)
    [lonu, latu] = polybool('+', lonu, latu, lon{ip}, lat{ip});
end
[latu, lonu] = polysplit(latu, lonu);

function [xc, yc] = calccircart(x, y, radius, npts)

ang = linspace(0, 2*pi, npts+1);
ang = ang(end-1:-1:1);
xc = bsxfun(@plus, x, radius * cos(ang));
yc = bsxfun(@plus, y, radius * sin(ang));
xc = num2cell(xc, 1);
yc = num2cell(yc, 1);

% if ~ispolycw(x,y)
%   [xc,yc] = poly2ccw(xc,yc);
% end

function [xrec, yrec] = calcrecart(x, y, halfwidth)

dx = diff(x);
dy = diff(y);

is1 = dx >= 0 & dy >= 0;
is2 = dx < 0 & dy >= 0;
is3 = dx < 0 & dy < 0;
is4 = dx >= 0 & dy < 0;

ish1 = dy == 0 & dx > 0;
ish2 = dy == 0 & dx < 0;

theta = zeros(5,1);

```

```

theta(is1 | is3) = atan(dy(is1 | is3)./dx(is1 | is3));
theta(is2 | is4) = -atan(dy(is2 | is4)./dx(is2 | is4));

[xl,xr,yl,yr] = deal(zeros(size(dx)));

xl(is1) = -halfwidth * sin(theta(is1));
xr(is1) = halfwidth * sin(theta(is1));
yl(is1) = halfwidth * cos(theta(is1));
yr(is1) = -halfwidth * cos(theta(is1));

xl(is2) = -halfwidth * sin(theta(is2));
xr(is2) = halfwidth * sin(theta(is2));
yl(is2) = -halfwidth * cos(theta(is2));
yr(is2) = halfwidth * cos(theta(is2));

xl(is3) = halfwidth * sin(theta(is3));
xr(is3) = -halfwidth * sin(theta(is3));
yl(is3) = -halfwidth * cos(theta(is3));
yr(is3) = halfwidth * cos(theta(is3));

xl(is4) = halfwidth * sin(theta(is4));
xr(is4) = -halfwidth * sin(theta(is4));
yl(is4) = halfwidth * cos(theta(is4));
yr(is4) = -halfwidth * cos(theta(is4));

xrec = [xl+x(1:end-1) xl+x(2:end) xr+x(2:end) xr+x(1:end-1) xl+x(1:end-1)];
yrec = [yl+y(1:end-1) yl+y(2:end) yr+y(2:end) yr+y(1:end-1) yl+y(1:end-1)];

xrec = num2cell(xrec, 2);
yrec = num2cell(yrec, 2);

```

% This function draws a circle for a given center and radius

function bubble = circle(center,radius)

Resolution = 100;

THETA=linspace(0,2*pi,Resolution);

RHO=ones(1,Resolution)*radius;

[X,Y] = pol2cart(THETA,RHO);

bubble.X=X+center(1);

bubble.Y=Y+center(2);

```
% This function generates the SAW's intended destinations  
% (i.e. Areas A1, A2, and A3 of Jurong Island)
```

```
function destinations = generate__destinations(times)
```

```
% Define the global variables  
globals;
```

```
% Define the destination locations
```

```
for i=1:NUM_DESTINATIONS
```

```
    % Area A1
```

```
    if i==1
```

```
        destinations(1,i) = make_destination;
```

```
        destinations(1,i).time=times(1);
```

```
        destinations(1,i).id=i;
```

```
        destinations(1,i).x = 302.1144444;
```

```
        destinations(1,i).y = 347.7716667;
```

```
    % Now iterate for all times
```

```
    [temp,ntimes]=size(times);
```

```
    for n=2:ntimes
```

```
        destinations(n,i)=destination_step(destinations(n-1,i),times(n));
```

```
    end
```

```
end
```

```
    % Area A2
```

```
    if i==2
```

```
        destinations(1,i) = make_destination;
```

```
        destinations(1,i).time=times(1);
```

```
        destinations(1,i).id=i;
```

```
        destinations(1,i).x = 322.7311111;
```

```
        destinations(1,i).y = 367.2116667;
```

```
    % Now iterate for all times
```

```
    [temp,ntimes]=size(times);
```

```
    for n=2:ntimes
```

```
        destinations(n,i)=destination_step(destinations(n-1,i),times(n));
```

```
    end
```

```
end
```

```
    % Area A3
```

```
    if i==3
```

```
        destinations(1,i) = make_destination;
```

```
destinations(1,i).time=times(1);
destinations(1,i).id=i;

destinations(1,i).x = 343.3450000;
destinations(1,i).y = 382.5572222;

% Now iterate for all times
[temp,ntimes]=size(times);
for n=2:ntimes
    destinations(n,i)=destination_step(destinations(n-1,i),times(n));
end
end
end
```

```
% This function initialises a new destination  
  
function new=make_destination  
  
new=struct('id',0,'time',0,'x',0,'y',0,'phi',0,'vel',0,'gamma',0);
```

% This function is to step a destination

function next_destination=destination_step(destination,time)

globals;

% This is a no motion model

dt=time-destination.time;

next_destination = make_destination; % space for destination data structure

next_destination.id=destination.id;

next_destination.time = time;

next_destination.x = destination.x;

next_destination.y = destination.y;

next_destination.phi = destination.phi;

next_destination.vel = destination.vel;

next_destination.gamma = destination.gamma;

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function generates true target information for later observation
% and tracking. Dynamics of target are contained in "target_step", number
% targets is defined by the globally defined variable NUM_TARGETS. A data
% point is generated at each of a set of "times."

function targets = generate__targets(times)

% Define the global variables
globals;

% Define a SAW at pre-fixed location in the TSS
for i=1
    targets(1,i) = make_target;
    targets(1,i).time=times(1);
    targets(1,i).id=i;
    targets(1,i).x = 738.9616667;
    targets(1,i).y = 438.7050000;
    targets(1,i).gamma=0;

    % Now iterate for all times
    [temp,ntimes]=size(times);
    for n=2:ntimes
        targets(n,i)=target_step_SAW_path(targets(n-1,i),times(n));
    end
end

% Create non-SAW contacts at random locations within the TSS
% Not used since this thesis covers a single target
for i=2:NUM_TARGETS
    targets(1,i) = make_target;
    targets(1,i).time=times(1);
    targets(1,i).id=i;

    x = TSS_X(1)*rand;
    y = TSS_Y(1)*rand;

    while (inpolygon (x, y,TSS_X, TSS_Y) ~= 1)

```



```

    x = TSS_X(1)*rand;
    y = TSS_Y(1)*rand;
end

    targets(1,i).x = x;
    targets(1,i).y = y;
    targets(1,i).phi = 2*pi*rand -pi;
    targets(1,i).vel = MIN_TARGET_VEL + (MAX_TARGET_VEL-
MIN_TARGET_VEL)*rand;
    targets(1,i).gamma=0;

    % Now iterate for all time
    [temp,ntimes]=size(times);
    for n=2:ntimes
        targets(n,i)=target_step(targets(n-1,i),times(n));
    end
end

```

```
% Acknowledgement:  
% This code originates from H. Durrant-Whyte  
% Australian Centre for Field Robotics  
% The University of Sydney, Australia  
  
% This function initialises a new target  
  
function new=make_target  
  
new=struct('id',0,'time',0,'x',0,'y',0,'phi',0,'vel',0,'gamma',0);
```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This is the trajectory taken by the SAW

function next_target=target_step_SAW_path(target,time)

globals;

dt=time-target.time;
next_target = make_target;
next_target.id=target.id;
next_target.time = time;

% target.vel = 0.042870370 (in m/s which equates to 15knots)
% target.vel = 0.059160494 (in m/s which equates to 20knots)
% target.vel = 0.060018519 (in m/s which equates to 21knots)
% target.vel = 0.062876543 (in m/s which equates to 22knots)

if (next_target.time >= 0 && next_target.time < 1086)
    next_target.x = target.x + dt*0.042870370*cos(-pi+0.720);
    next_target.y = target.y + dt*0.042870370*sin(-pi+0.720);
    next_target.phi = -pi+0.720;
    next_target.vel = 0.042870370;

elseif (next_target.time >= 1086 && next_target.time < 5890)
    next_target.x = target.x + dt*0.042870370*cos(-pi+0.170);
    next_target.y = target.y + dt*0.042870370*sin(-pi+0.170);
    next_target.phi = -pi+0.170;
    next_target.vel = 0.042870370;

elseif (next_target.time >= 5890 && next_target.time < 10199)
    next_target.x = target.x + dt*0.042870370*cos(-pi+0.430);
    next_target.y = target.y + dt*0.042870370*sin(-pi+0.430);
    next_target.phi = -pi+0.430;
    next_target.vel = 0.042870370;

elseif (next_target.time >= 10199 && next_target.time < 11051)
    next_target.x = target.x + dt*0.042870370*cos(pi-0.770);
    next_target.y = target.y + dt*0.042870370*sin(pi-0.770);

```

```
next_target.phi = pi-0.770;
next_target.vel = 0.042870370;

elseif (next_target.time >= 11051 && next_target.time < 11500)
    next_target.x = target.x + dt*0.059160494*cos(pi-1.360);
    next_target.y = target.y + dt*0.059160494*sin(pi-1.360);
    next_target.phi = pi-1.360;
    next_target.vel = 0.059160494;

elseif (next_target.time >= 11500)
    next_target.x = target.x;
    next_target.y = target.y;
    next_target.phi = 0;
    next_target.vel = 0;
end
```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function generates trajectories for platforms

function platforms = generate__platforms(times)

% Define the global variables
globals;

% Define 5 fixed platform initial locations

% Model the MPA radar stations
for i=1:NUM_PLATFORMS
    if i==1
        platforms(1,i) = make_platform;
        platforms(1,i).time=times(1);
        platforms(1,i).id=i;
        platforms(1,i).x = 285.4700000;
        platforms(1,i).y = 360.8527778;
        platforms(1,i).phi = 2*pi*rand -pi;
        platforms(1,i).vel = 0;
        platforms(1,i).gamma=0;

        % Now iterate for all time
        [temp,ntimes]=size(times);
        for n=2:ntimes
            platforms(n,i)=platform_step(platforms(n-1,i),times(n));
        end
    end

    if i==2
        platforms(1,i) = make_platform;
        platforms(1,i).time=times(1);
        platforms(1,i).id=i;
        platforms(1,i).x = 342.7972222;
        platforms(1,i).y = 311.9850000;
        platforms(1,i).phi = 2*pi*rand -pi;
        platforms(1,i).vel = 0;
        platforms(1,i).gamma=0;
    end
end

```

```

% Now iterate for all time
[temp,ntimes]=size(times);
for n=2:ntimes
    platforms(n,i)=platform_step(platforms(n-1,i),times(n));
end
end

if i==3
    platforms(1,i) = make_platform;
    platforms(1,i).time=times(1);
    platforms(1,i).id=i;
    platforms(1,i).x = 413.1333333;
    platforms(1,i).y = 349.8194444;
    platforms(1,i).phi = 2*pi*rand -pi;
    platforms(1,i).vel = 0;
    platforms(1,i).gamma=0;

    % Now iterate for all time
    [temp,ntimes]=size(times);
    for n=2:ntimes
        platforms(n,i)=platform_step(platforms(n-1,i),times(n));
    end
end

if i==4
    platforms(1,i) = make_platform;
    platforms(1,i).time=times(1);
    platforms(1,i).id=i;
    platforms(1,i).x = 461.4761111;
    platforms(1,i).y = 403.4261111;
    platforms(1,i).phi = 2*pi*rand -pi;
    platforms(1,i).vel = 0;
    platforms(1,i).gamma=0;

    % Now iterate for all time
    [temp,ntimes]=size(times);
    for n=2:ntimes
        platforms(n,i)=platform_step(platforms(n-1,i),times(n));
    end
end

if i==5
    platforms(1,i) = make_platform;
    platforms(1,i).time=times(1);

```

```

platforms(1,i).id=i;
platforms(1,i).x = 517.0844444;
platforms(1,i).y = 421.9305556;
platforms(1,i).phi = 2*pi*rand -pi;
platforms(1,i).vel = 0;
platforms(1,i).gamma=0;

% Now iterate for all time
[temp,ntimes]=size(times);
for n=2:ntimes
    platforms(n,i)=platform_step(platforms(n-1,i),times(n));
end
end

% Define 3 mobile platform initial locations

% Model the CPC
if i==6
    platforms(1,i) = make_platform;
    platforms(1,i).time=times(1);
    platforms(1,i).id=i;
    platforms(1,i).x = 474.1785056;
    platforms(1,i).y = 351.7903000;
    platforms(1,i).phi = 2*pi*rand -pi;
    platforms(1,i).vel = 0;
    targets(1,i).gamma=0;

    % Now iterate for all time
    [temp,ntimes]=size(times);
    for n=2:ntimes
        platforms(n,i)=platform_step_PCG1_path(platforms(n-1,i),times(n));
    end
end

if i==7
    platforms(1,i) = make_platform;
    platforms(1,i).time=times(1);
    platforms(1,i).id=i;
    platforms(1,i).x = 474.1785056;
    platforms(1,i).y = 351.7903000;
    platforms(1,i).phi = 2*pi*rand -pi;
    platforms(1,i).vel = 0;
    targets(1,i).gamma=0;

    % Now iterate for all time

```

```

[temp,ntimes]=size(times);
for n=2:ntimes
    platforms(n,i)=platform_step_PCG2_path(platforms(n-1,i),times(n));
end
end

% Model the PV
if i==8
    platforms(1,i) = make_platform;
    platforms(1,i).time=times(1);
    platforms(1,i).id=i;
    platforms(1,i).x = 474.1785056;
    platforms(1,i).y = 351.7903000;
    platforms(1,i).phi = 2*pi*rand -pi;
    platforms(1,i).vel = 0;
    targets(1,i).gamma=0;

    % Now iterate for all time
    [temp,ntimes]=size(times);
    for n=2:ntimes
        platforms(n,i)=platform_step_PV_path(platforms(n-1,i),times(n));
    end
end

% Given the speed and sensor coverage of the MPaA,
% its surveying of the small TSS can be modeled as a "fixed" sensor
% with complete of the TSS during its operation.
% Model the MPaA
if i==9
    platforms(1,i) = make_platform;
    platforms(1,i).time=times(1);
    platforms(1,i).id=i;
    platforms(1,i).x = 474.1785056;
    platforms(1,i).y = 351.7903000;
    platforms(1,i).phi = 2*pi*rand -pi;
    platforms(1,i).vel = 0;
    targets(1,i).gamma=0;

    % Now iterate for all time
    [temp,ntimes]=size(times);
    for n=2:ntimes
        platforms(n,i)=platform_step_MPAC(platforms(n-1,i),times(n));
    end
end
end
end

```



```
% Acknowledgement:  
% This code originates from H. Durrant-Whyte  
% Australian Centre for Field Robotics  
% The University of Sydney, Australia  
  
% This function initialises a new platform  
  
function new=make_platform  
  
new=struct('id',0,'time',0,'x',0,'y',0,'phi',0,'vel',0,'gamma',0);
```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% This is a no motion platform model for MPA radar station

function next_platform=platform_step(platform,time)

globals;

dt=time-platform.time;
next_platform = make_platform;
next_platform.id=platform.id;
next_platform.time = time;
next_platform.x = platform.x;
next_platform.y = platform.y;
next_platform.phi = platform.phi;
next_platform.vel = platform.vel;
next_platform.gamma = platform.gamma;

```

```

% This is the platform motion model for one of the two CPCs for a day

function next_platform=platform_step_PCG1_path(platform,time)

globals;

dt=time-platform.time;
next_platform = make_platform;
next_platform.id=platform.id;
next_platform.time = time;

if (next_platform.time >= 0)
    next_platform.x = 474.1785056;
    next_platform.y = 351.7903000;
    next_platform.phi = platform.phi;
    next_platform.vel = platform.vel;
    next_platform.gamma = platform.gamma;
end

%-----
% This models the platform's patrol path. However, by making the assumption
% that the sensors have extended range to simplify sensor management, this
% modeling is not used here. This will be useful for future work with the
% inclusion of sensor management.
%
% if (next_platform.time >= 0 && next_platform.time < 18000)
%     next_platform.x = platform.x + dt*0.015061617*cos(1.519574188-0.5*pi);
%     next_platform.y = platform.y + dt*0.015061617*sin(1.519574188-0.5*pi);
%     next_platform.phi = 1.519574188-0.5*pi;
%     next_platform.vel = 0.015061617;
%     next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 18000 && next_platform.time < 32400)
%     next_platform.x = platform.x + dt*0.025102695*cos(1.519574188+0.5*pi);
%     next_platform.y = platform.y + dt*0.025102695*sin(1.519574188+0.5*pi);
%     next_platform.phi = 1.519574188+0.5*pi;
%     next_platform.vel = 0.025102695;
%     next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 32400 && next_platform.time < 46800)
%     next_platform.x = platform.x + dt*0.015061617*cos(1.519574188-0.5*pi);
%     next_platform.y = platform.y + dt*0.015061617*sin(1.519574188-0.5*pi);
%     %next_platform.phi = 1.519574188-0.5*pi;
%     %next_platform.vel = 0.015061617;

```

```

%   %next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 46800 && next_platform.time < 61200)
%   next_platform.x = platform.x + dt*0.025102695*cos(1.519574188+0.5*pi);
%   next_platform.y = platform.y + dt*0.025102695*sin(1.519574188+0.5*pi);
%   next_platform.phi = 1.519574188+0.5*pi;
%   next_platform.vel = 0.025102695;
%   next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 61200 && next_platform.time < 75600)
%   next_platform.x = platform.x + dt*0.015061617*cos(1.519574188-0.5*pi);
%   next_platform.y = platform.y + dt*0.015061617*sin(1.519574188-0.5*pi);
%   %next_platform.phi = 1.519574188-0.5*pi;
%   %next_platform.vel = 0.015061617;
%   %next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 75600 && next_platform.time < 86400)
%   next_platform.x = platform.x + dt*0.025102695*cos(1.519574188+0.5*pi);
%   next_platform.y = platform.y + dt*0.025102695*sin(1.519574188+0.5*pi);
%   next_platform.phi = 1.519574188+0.5*pi;
%   next_platform.vel = 0.025102695;
%   next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 86400)
%   next_platform.x = platform.x;
%   next_platform.y = platform.y;
%   next_platform.phi = 0;
%   next_platform.vel = 0;
%   next_platform.gamma = 0;
% end
%-----

```

```

% This is the platform motion model for one of the two CPCs for a day

function next_platform=platform_step_PCG2_path(platform,time)

globals;

dt=time-platform.time;
next_platform = make_platform;
next_platform.id=platform.id;
next_platform.time = time;

if (next_platform.time >= 0)
    next_platform.x = 474.1785056;
    next_platform.y = 351.7903000;
    next_platform.phi = platform.phi;
    next_platform.vel = platform.vel;
    next_platform.gamma = platform.gamma;
end

%-----
% This models the platform's patrol path. However, by making the assumption
% that the sensors have extended range to simplify sensor management, this
% modeling is not used here. This will be useful for future work with the
% inclusion of sensor management.
%
% if (next_platform.time >= 0 && next_platform.time < 18000)
%     next_platform.x = platform.x + dt*0.013140537*cos(-1.330818664+0.5*pi);
%     next_platform.y = platform.y + dt*0.013140537*sin(-1.330818664+0.5*pi);
%     next_platform.phi = -1.330818664+0.5*pi;
%     next_platform.vel = 0.013140537;
%     next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 18000 && next_platform.time < 32400)
%     next_platform.x = platform.x + dt*0.021900894*cos(-1.330818664-0.5*pi);
%     next_platform.y = platform.y + dt*0.021900894*sin(-1.330818664-0.5*pi);
%     next_platform.phi = -1.330818664-0.5*pi;
%     next_platform.vel = 0.021900894;
%     next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 32400 && next_platform.time < 46800)
%     next_platform.x = platform.x + dt*0.013140537*cos(-1.330818664+0.5*pi);
%     next_platform.y = platform.y + dt*0.013140537*sin(-1.330818664+0.5*pi);
%     next_platform.phi = -1.330818664+0.5*pi;
%     next_platform.vel = 0.013140537;

```

```

%   next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 46800 && next_platform.time < 61200)
%   next_platform.x = platform.x + dt*0.021900894*cos(-1.330818664-0.5*pi);
%   next_platform.y = platform.y + dt*0.021900894*sin(-1.330818664-0.5*pi);
%   next_platform.phi = -1.330818664-0.5*pi;
%   next_platform.vel = 0.021900894;
%   next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 61200 && next_platform.time < 75600)
%   next_platform.x = platform.x + dt*0.013140537*cos(-1.330818664+0.5*pi);
%   next_platform.y = platform.y + dt*0.013140537*sin(-1.330818664+0.5*pi);
%   next_platform.phi = -1.330818664+0.5*pi;
%   next_platform.vel = 0.013140537;
%   next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 75600 && next_platform.time < 86400)
%   next_platform.x = platform.x + dt*0.021900894*cos(-1.330818664-0.5*pi);
%   next_platform.y = platform.y + dt*0.021900894*sin(-1.330818664-0.5*pi);
%   next_platform.phi = -1.330818664-0.5*pi;
%   next_platform.vel = 0.021900894;
%   next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 86400)
%   next_platform.x = platform.x;
%   next_platform.y = platform.y;
%   next_platform.phi = 0;
%   next_platform.vel = 0;
%   next_platform.gamma = 0;
% end
%-----

```

```

% This is the platform motion model for PV for a day

function next_platform=platform_step_PV_path(platform,time)

globals;

dt=time-platform.time;
next_platform = make_platform;
next_platform.id=platform.id;
next_platform.time = time;

if (next_platform.time >= 0)
    next_platform.x = 474.1785056;
    next_platform.y = 351.7903000;
    next_platform.phi = platform.phi;
    next_platform.vel = platform.vel;
    next_platform.gamma = platform.gamma;
end

%-----
% This models the platform's patrol path. However, by making the assumption
% that the sensors have extended range to simplify sensor management, this
% modeling is not used here. This will be useful for future work with the
% inclusion of sensor management.
%
% if (next_platform.time >= 0 && next_platform.time < 43200)
%     next_platform.x = platform.x + dt*0.013972407*cos(0.152813367);
%     next_platform.y = platform.y + dt*0.013972407*sin(0.152813367);
%     next_platform.phi = 0.152813367;
%     next_platform.vel = 0.013972407;
%     next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 43200 && next_platform.time < 86400)
%     next_platform.x = platform.x + dt*0.013972407*cos(0.152813367+pi);
%     next_platform.y = platform.y + dt*0.013972407*sin(0.152813367+pi);
%     next_platform.phi = 0.152813367+pi;
%     next_platform.vel = 0.013972407;
%     next_platform.gamma = platform.gamma;
%
% elseif (next_platform.time >= 86400)
%     next_platform.x = platform.x;
%     next_platform.y = platform.y;
%     next_platform.phi = 0;
%     next_platform.vel = 0;

```

```
%  next_platform.gamma = 0;  
% end  
%-----
```



```

% This is the platform motion model for MPaA

function next_platform=platform_step_MPAC(platform,time)

globals;

next_platform = make_platform;
next_platform.id=platform.id;
next_platform.time = time;

if (next_platform.time >= 0)
    next_platform.x = 474.1785056;
    next_platform.y = 351.7903000;
    next_platform.phi = platform.phi;
    next_platform.vel = platform.vel;
    next_platform.gamma = platform.gamma;
end

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function is to get a fixed filter parameter block

function filter=get_filter_params

globals;

% Assume a constant velocity model;
filter.XDIM=4;
filter.ZDIM=2;
filter.Q= [DT^2*SIGMA_XDOT^2 0 0 0;
           0 SIGMA_XDOT^2 0 0;
           0 0 DT^2*SIGMA_YDOT^2 0;
           0 0 0 SIGMA_YDOT^2];
filter.H= [1 0 0 0; 0 0 1 0];

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% Returns a sensor data structure which contains all
% necessary information for a decentralised sensor.

function sensor=make__sensor(id,filter)

globals;
ginit;

% Define sensors for the 5 fixed platforms
if id==1
    sensor.id=id;
    sensor.time=0.0;
    sensor.r_err=SIGMA_RANGE_MPA;
    sensor.b_err=SIGMA_BEARING_MPA;
    sensor.point=0;      % zero point angle
    sensor.beam_view=2*pi;
    sensor.max_range=154; % an equivalent of 15nm
    sensor.filter=filter;
    sensor.est=make_track(id,filter.XDIM,0.0);
    sensor.pred=make_track(id,filter.XDIM,0.0);
    sensor.num_chans=NUM_CHANS;
    for i=1:NUM_CHANS
        sensor.chan_in(i)=make_track(i,filter.XDIM,0.0);
        sensor.chan_out(i)=make_track(i,filter.XDIM,0.0);
    end

elseif id==2
    sensor.id=id;
    sensor.time=0.0;
    sensor.r_err=SIGMA_RANGE_MPA;
    sensor.b_err=SIGMA_BEARING_MPA;
    sensor.point=0;
    sensor.beam_view=2*pi;
    sensor.max_range=154;
    sensor.filter=filter;
    sensor.est=make_track(id,filter.XDIM,0.0);
    sensor.pred=make_track(id,filter.XDIM,0.0);

```

```

    sensor.num_chans=NUM_CHANS;
    for i=1:NUM_CHANS
        sensor.chan_in(i)=make_track(i,filter.XDIM,0.0);
        sensor.chan_out(i)=make_track(i,filter.XDIM,0.0);
    end

elseif id==3
    sensor.id=id;
    sensor.time=0.0;
    sensor.r_err=SIGMA_RANGE_MPA;
    sensor.b_err=SIGMA_BEARING_MPA;
    sensor.point=0;
    sensor.beam_view=2*pi;
    sensor.max_range=154;
    sensor.filter=filter;
    sensor.est=make_track(id,filter.XDIM,0.0);
    sensor.pred=make_track(id,filter.XDIM,0.0);
    sensor.num_chans=NUM_CHANS;
    for i=1:NUM_CHANS
        sensor.chan_in(i)=make_track(i,filter.XDIM,0.0);
        sensor.chan_out(i)=make_track(i,filter.XDIM,0.0);
    end

elseif id==4
    sensor.id=id;
    sensor.time=0.0;
    sensor.r_err=SIGMA_RANGE_MPA;
    sensor.b_err=SIGMA_BEARING_MPA;
    sensor.point=0;
    sensor.beam_view=2*pi;
    sensor.max_range=154;
    sensor.filter=filter;
    sensor.est=make_track(id,filter.XDIM,0.0);
    sensor.pred=make_track(id,filter.XDIM,0.0);
    sensor.num_chans=NUM_CHANS;
    for i=1:NUM_CHANS
        sensor.chan_in(i)=make_track(i,filter.XDIM,0.0);
        sensor.chan_out(i)=make_track(i,filter.XDIM,0.0);
    end

elseif id==5
    sensor.id=id;
    sensor.time=0.0;
    sensor.r_err=SIGMA_RANGE_MPA;
    sensor.b_err=SIGMA_BEARING_MPA;

```

```

sensor.point=0;
sensor.beam_view=2*pi;
sensor.max_range=154;
sensor.filter=filter;
sensor.est=make_track(id,filter.XDIM,0.0);
sensor.pred=make_track(id,filter.XDIM,0.0);
sensor.num_chans=NUM_CHANS;
for i=1:NUM_CHANS
    sensor.chan_in(i)=make__track(i,filter.XDIM,0.0);
    sensor.chan_out(i)=make__track(i,filter.XDIM,0.0);
end

```

% Define sensors for the 4 mobile platforms

% Define the CPC sensor

```

elseif id==6
    sensor.id=id;
    sensor.time=0.0;
    sensor.r_err=SIGMA_RANGE_PCG;
    sensor.b_err=SIGMA_BEARING_PCG;
    sensor.point=0;      % zero point angle
    sensor.beam_view=2*pi;
    sensor.max_range=1000; % range extended to simulate
                        % presence at Jurong Island
    sensor.filter=filter;
    sensor.est=make_track(id,filter.XDIM,0.0);
    sensor.pred=make_track(id,filter.XDIM,0.0);
    sensor.num_chans=NUM_CHANS;
    for i=1:NUM_CHANS
        sensor.chan_in(i)=make_track(i,filter.XDIM,0.0);
        sensor.chan_out(i)=make_track(i,filter.XDIM,0.0);
    end
end

```

% Define the CPC sensor

```

elseif id==7
    sensor.id=id;
    sensor.time=0.0;
    sensor.r_err=SIGMA_RANGE_PCG;
    sensor.b_err=SIGMA_BEARING_PCG;
    sensor.point=0;
    sensor.beam_view=2*pi;
    sensor.max_range=1000;
    sensor.filter=filter;
    sensor.est=make_track(id,filter.XDIM,0.0);
    sensor.pred=make_track(id,filter.XDIM,0.0);
end

```

```

    sensor.num_chans=NUM_CHANS;
    for i=1:NUM_CHANS
        sensor.chan_in(i)=make_track(i,filter.XDIM,0.0);
        sensor.chan_out(i)=make_track(i,filter.XDIM,0.0);
    end

% Define the PV sensor
elseif id==8
    sensor.id=id;
    sensor.time=0.0;
    sensor.r_err=SIGMA_RANGE_PV;
    sensor.b_err=SIGMA_BEARING_PV;
    sensor.point=0;
    sensor.beam_view=2*pi;
    sensor.max_range=1000;
    sensor.filter=filter;
    sensor.est=make_track(id,filter.XDIM,0.0);
    sensor.pred=make_track(id,filter.XDIM,0.0);
    sensor.num_chans=NUM_CHANS;
    for i=1:NUM_CHANS
        sensor.chan_in(i)=make_track(i,filter.XDIM,0.0);
        sensor.chan_out(i)=make_track(i,filter.XDIM,0.0);
    end

% Define the MPaA sensor
elseif id==9
    sensor.id=id;
    sensor.time=0.0;
    sensor.r_err=SIGMA_RANGE_MPAC;
    sensor.b_err=SIGMA_BEARING_MPAC;
    sensor.point=0;
    sensor.beam_view=2*pi;
    sensor.max_range=1000;
    sensor.filter=filter;
    sensor.est=make_track(id,filter.XDIM,0.0);
    sensor.pred=make_track(id,filter.XDIM,0.0);
    sensor.num_chans=NUM_CHANS;
    for i=1:NUM_CHANS
        sensor.chan_in(i)=make_track(i,filter.XDIM,0.0);
        sensor.chan_out(i)=make_track(i,filter.XDIM,0.0);
    end
end
end

```

```
% Acknowledgement:  
% This code originates from H. Durrant-Whyte  
% Australian Centre for Field Robotics  
% The University of Sydney, Australia
```

```
% This function makes space for a track
```

```
function track=make_track(id,dim,time)
```

```
track.id=id;  
track.XDIM=dim;  
track.time=time;  
track.y=zeros(dim,1);  
track.Y=zeros(dim,dim);
```

```
% This function initialises a network structure
```

```
function network=init_net(platforms,nodes)
```

```
nsize=length(nodes);  
network=ones(nsize,nsize);
```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function constructs a report for a sensor at a particular time

function report=sensor__report(sensor,platforms,targets,nsensor,ntime)

globals;

report=zeros(NUM_TARGETS,5);

% Find platform location at this time
px=platforms(ntime,nsensor).x;
py=platforms(ntime,nsensor).y;

for tnum=1:NUM_TARGETS
    % Find target location
    tx=targets(ntime,tnum).x;
    ty=targets(ntime,tnum).y;
    % Compute true range and bearing
    dx=tx-px;
    dy=ty-py;
    range=sqrt(dx*dx + dy*dy);
    bearing=atan2(dy,dx);
    % determine if this is actually visible
    if range < sensor.max_range
        % Add error from sensor model
        report(tnum,1)=tnum;
        report(tnum,2)=range + sensor.r_err*(-1+2*rand);
        report(tnum,3)=bearing + sensor.b_err*(-1+2*rand);
        report(tnum,4)=px;
        report(tnum,5)=py;
    else
        report(tnum,1)=tnum;
        report(tnum,2)=NaN;
        report(tnum,3)=NaN;
        report(tnum,4)=NaN;
        report(tnum,5)=NaN;
    end
end
end

```



```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function performs local prediction for each node using the CVM

function node=local_predict(node,time)

globals;

dt=time-node.time;
if dt < 0
    error('Negative prediction step in state_pred')
end

% Compute matrices
F=[1 dt 0 0;
    0 1 0 0;
    0 0 1 dt;
    0 0 0 1];
G=[1 0 0 0;
    0 1 0 0;
    0 0 1 0;
    0 0 0 1];
Q=node.filter.Q;

% Do prediction
M=(inv(F))'*(node.est.Y)*(inv(F));
Sigma=G'*M*G+inv(Q);
Omega=M*G*inv(Sigma);
node.pred.Y=M-(Omega*Sigma*Omega');
node.pred.y=(eye(4,4)-(Omega*G'))*(inv(F))'*(node.est.y);
node.time=time;
% Seems to require for stability in Matlab
node.pred.Y=force_sym(node.pred.Y);

```

```
% Acknowledgement:  
% This code originates from H. Durrant-Whyte  
% Australian Centre for Field Robotics  
% The University of Sydney, Australia
```

```
% Modified by J. Ng on 30 May 2011.
```

```
% This function forces symmetry on a matrix
```

```
function sigma=force_sym(sigma)  
  
[nx,ny]=size(sigma);  
if nx ~= ny  
    error('Matrix must be square');  
end  
  
for i=1:nx  
    for j=i+1:nx  
        temp=(sigma(i,j)+sigma(j,i))/2;  
        sigma(i,j)=temp;  
        sigma(j,i)=temp;  
    end  
end
```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function performs information filter local update for each node

function node=local_update(node,obs,i)

globals;

% Observation model
H=node.filter.H;

if (isnan(obs.report(2:5))) % Wait for observation
    node.est.y=zeros(XSIZE,1);
    node.est.Y=zeros(XSIZE, XSIZE);
else % Update with observation
    % Extract observation
    [zx,zy,R]=xy_obs(obs.report,1,i); % R is not used
    z=[zx;zy];
    if i==1
        Ri=inv(Rfilter_MPA);
    elseif i==2
        Ri=inv(Rfilter_MPA);
    elseif i==3
        Ri=inv(Rfilter_MPA);
    elseif i==4
        Ri=inv(Rfilter_MPA);
    elseif i==5
        Ri=inv(Rfilter_MPA);
    elseif i==6
        Ri=inv(Rfilter_PCG);
    elseif i==7
        Ri=inv(Rfilter_PCG);
    elseif i==8
        Ri=inv(Rfilter_PV);
    elseif i==9
        Ri=inv(Rfilter_MPAC);
    end
    % Construct information terms
    info=H'*Ri*z;

```

```
Info=H'*Ri*H;  
% Add the information to prediction  
node.est.y=node.pred.y+info;  
node.est.Y=node.pred.Y+Info;  
% Seems to require the next step for stability in Matlab  
node.est.Y=force_sym(node.est.Y);  
end
```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function constructs the sensor network connectivity topology

function net=set_net_topology(net,nodes,time,METHOD);

globals;

switch upper(METHOD)
case 'FIXED' % topology is invariant
    return;
case 'BROADCAST'
    net=ones(NUM_NODES,NUM_NODES);
    for i=1:NUM_NODES
        net(i,i)=0; % nodes don't commuincate with themselves
    end
case 'RANDOM'
    net=net; % not implemented ye
otherwise
    disp('Warning: unknown network topology type');
    net=net % default is fixed
end

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% A function to "communicate" information between sensor nodes
% In general the philosophy is first to identify the nodes
% in the communication neighbourhood, then to compute a common
% information term locally. Finally the complete estimate is
% communicated and subtraction of common terms is done at the
% receiving node. This approach makes the fusion process robust to
% changes in communication topology.

function node=communicate(Nodes,com_net,node_num)

globals;

% This node
node=Nodes(node_num);

% Find the current communication neighbourhood
num_chan=1;
for i=1:NUM_NODES
    if com_net(node_num,i)==1
        % For connected nodes, compute common prior and estimate
        % The difference is the new information
        node.chan_out(num_chan)=Nodes(i).pred;
        % chan in contains the new observations
        node.chan_in(num_chan).y=Nodes(i).est.y-Nodes(i).pred.y;
        node.chan_in(num_chan).Y=Nodes(i).est.Y-Nodes(i).pred.Y;
        num_chan=num_chan+1;
    end
end

% Note the number of channels for later
nodes.num_chan=num_chan-1;

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function assimilates all the information from each sensor channel.
% The new information is simply the incoming minus the outgoing
% information, or the update minus the common information.

function node=assimilate(node,time);

globals;

% Initialisation
locali=node.est.y-node.pred.y;
localI=node.est.Y-node.pred.Y;
yc=node.pred.y;
Yc=node.pred.Y;

% Generate common prior using covariance intersection
if ~FULL_CON
    for i=1:node.num_chans
        [yc,Yc,omega]=ci_common(node.chan_out(i).y,node.chan_out(i).Y,yc,Yc);
    end
end
% Or through fully connected assumption
node.est.y=yc+locali;
node.est.Y=Yc+localI;

for i=1:node.num_chans
    node.est.y=node.est.y+node.chan_in(i).y;
    node.est.Y=node.est.Y+node.chan_in(i).Y;
end
% Seems to require the next step for stability in Matlab
node.est.Y=force_sym(node.est.Y);
% This function performs the covariance intersect update of two estimates
% (a with covariance A; b with covariance B) and an observation matrix H.
% The output is the output estimate (c with covariance C) and the value
% of omega which minimizes the determinant of C.

function [yc,Yc,omega] = ci_common(ya,Ya,yb,Yb)

```

```

globals;

H = [1 0 0 0;
     0 0 1 0;
     1 0 0 0;
     0 0 1 0];

if Ya(1)~=0 || Yb(1)~=0
    f=inline('1/det(omega*Yai+(1-omega)*Ybi)','omega', 'Yai', 'Ybi');
    Yai=inv(Ya);
    Ybi=inv(Yb);
    YbiH=H'*Ybi;
    YbiHH=YbiH*H;
    omega = fminsearch(f,0.5,[],Yai,Ybi);
    Yci=Yai*omega+YbiHH*(1-omega);
    Yc=inv(Yci);           % New covariance
    yc=Yc*(Yai*ya+YbiH*yb); % New mean
else
    Yc=zeros(XSIZE,XSIZE);
    yc=zeros(XSIZE,1);
    omega = 0;
end

```



```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function displays true target trajectories, platform trajectories,
% target destinations, and sensor observations following a simulation run

function [true,plat,obs,dest]=post_sim(targets,observations,platforms,destinations)

globals;
% Set plotting size
[nsamps,nplatforms]=size(platforms);
[nsamps,ndestinations]=size(destinations);
[nsamps,ntargets]=size(targets);
[nsensors,nsamps]=size(observations);

figure(1)
clf;
hold on
data=zeros(2,nsamps);
title('True Target Motions')
xlabel('x-position (m)')
ylabel('y-position (m)')

% Plot true target trajectories
for n=1:ntargets
    for i=1:nsamps
        data(1,i)=targets(i,n).x;
        data(2,i)=targets(i,n).y;
    end
    true=plot(data(1,:),data(2,:), 'b-');
end

% Plot true platform trajectories
for n=1:nplatforms
    for i=1:nsamps
        data(1,i)=platforms(i,n).x;
        data(2,i)=platforms(i,n).y;
    end
    if n == 1
        plat=plot(data(1,:),data(2,:), 'ko');
    end
end

```

```

elseif n == 2
    plat=plot(data(1,:),data(2:),'ko');
elseif n == 3
    plat=plot(data(1,:),data(2:),'ko');
elseif n == 4
    plat=plot(data(1,:),data(2:),'ko');
elseif n == 5
    plat=plot(data(1,:),data(2:),'ko');
elseif n == 6
    plat=plot(data(1,:),data(2:),'r*');
elseif n == 7
    plat=plot(data(1,:),data(2:),'r*');
elseif n == 8
    plat=plot(data(1,:),data(2:),'gh');
elseif n == 9
    plat=plot(data(1,:),data(2:),'bo');
end
end

% Plot true destination locations
for n=1:ndestinations
    for i=1:nsamps
        data(1,i)=destinations(i,n).x;
        data(2,i)=destinations(i,n).y;
    end
    dest=plot(data(1,:),data(2:),'rh');
end

% Plot sensor observations
for n=1:nsensors % for all sensors
    for i=1:nsamps % for all time
        report=observations(n,i).report;
        for m=1:ntargets % for all targets
            if report(m,1) > 0 % if they are seen
                [zx,zy,R]=xy_obs(report,m,n); % convert observation to global xy coordinates
                odata(1,m)=zx;
                odata(2,m)=zy;
            end
        end
    end
    if n == 1
        obs=plot(odata(1,:),odata(2:),'k. ');
    elseif n == 2
        obs=plot(odata(1,:),odata(2:),'k. ');
    elseif n == 3
        obs=plot(odata(1,:),odata(2:),'k. ');
    end
end

```

```

elseif n == 4
    obs=plot(odata(1,:),odata(2,:),'k.');
elseif n == 5
    obs=plot(odata(1,:),odata(2,:),'k.');
elseif n == 6
    obs=plot(odata(1,:),odata(2,:),'r.');
elseif n == 7
    obs=plot(odata(1,:),odata(2,:),'r.');
elseif n == 8
    obs=plot(odata(1,:),odata(2,:),'g.');
elseif n == 9
    obs=plot(odata(1,:),odata(2,:),'b.');
end
end
end

legend([true,plat,obs,dest],'Target      True      Position','Tracking      Stations','Target
Observations','Intended Target Destination')
hold off

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function extract an xy observation from a range-bearing report

function [zx,zy,R]=xy_obs(rep,n,sensor_id)

globals;

sr=sin(rep(n,3));
cr=cos(rep(n,3));
zx=rep(n,4)+rep(n,2)*cr;
zy=rep(n,5)+rep(n,2)*sr;
ROT=[cr -sr; sr cr];
range2=rep(n,2)*rep(n,2);
if sensor_id==1
    sigma=[SIGMA_RANGE_MPA^2 0;0 range2*SIGMA_BEARING_MPA^2];
elseif sensor_id==2
    sigma=[SIGMA_RANGE_MPA^2 0;0 range2*SIGMA_BEARING_MPA^2];
elseif sensor_id==3
    sigma=[SIGMA_RANGE_MPA^2 0;0 range2*SIGMA_BEARING_MPA^2];
elseif sensor_id==4
    sigma=[SIGMA_RANGE_MPA^2 0;0 range2*SIGMA_BEARING_MPA^2];
elseif sensor_id==5
    sigma=[SIGMA_RANGE_MPA^2 0;0 range2*SIGMA_BEARING_MPA^2];
elseif sensor_id==6
    sigma=[SIGMA_RANGE_PCG^2 0;0 range2*SIGMA_BEARING_PCG^2];
elseif sensor_id==7
    sigma=[SIGMA_RANGE_PCG^2 0;0 range2*SIGMA_BEARING_PCG^2];
elseif sensor_id==8
    sigma=[SIGMA_RANGE_PV^2 0;0 range2*SIGMA_BEARING_PV^2];
elseif sensor_id==9
    sigma=[SIGMA_RANGE_MPAC^2 0;0 range2*SIGMA_BEARING_MPAC^2];
end
R=ROT*sigma*ROT';

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function displays the fused target track following a simulation run

function plot_tracks(targets,y,Y,yp,Yp, ty, TY,typ,TYp, times)

globals;

% Set plotting size
[ntimes,nsensors,xdim1]=size(y);
[ntimes,nsensors,xdim2,xdim3]=size(Y);
odata=zeros(2,ntimes);
tdata=zeros(2,ntimes);
title('Target Motions')
xlabel('x-position')
ylabel('y-position')
xlim([250 750])
ylim([280 440])
hold on
warning off;

% Estimated individual trajectories
for s=1:nsensors
    for i=1:ntimes
        P=inv(squeeze(Y(i,s,:)));
        x=P*squeeze(y(i,s,:));
        odata(1,i,s)=x(1);
        odata(2,i,s)=x(3);
    end
    if s==1
        plot(odata(1,:,1),odata(2,:,1),'r--')
    elseif s==2
        plot(odata(1,:,2),odata(2,:,2),'g--')
    elseif s==3
        plot(odata(1,:,3),odata(2,:,3),'b--')
    elseif s==4
        plot(odata(1,:,4),odata(2,:,4),'m--')
    elseif s==5
        plot(odata(1,:,5),odata(2,:,5),'k--')
    end
end

```

```

elseif s==6
    plot(odata(1,:,6),odata(2,:,6),'r:')
elseif s==7
    plot(odata(1,:,7),odata(2,:,7),'g:')
elseif s==8
    plot(odata(1,:,8),odata(2,:,8),'b:')
elseif s==9
    plot(odata(1,:,9),odata(2,:,9),'m:')
end
end

```

```

% Estimated fused trajectory

```

```

for i=1:ntimes
    x=ty(i,:);
    tdata(1,i)=x(1);
    tdata(2,i)=x(3);
end

```

```

plot(tdata(1,:,1),tdata(2,:,1),'g-')

```

```

hold off

```

```

warning on;

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% Modified by J. Ng on 30 May 2011.

% This function produces various plots with regards to the SAW parameters

function plot_errors(targets,y,Y,yp,Yp,times,nfigure,trackstarttime)

globals;

% Initialise
[ntimes,nsensors,xdim1]=size(y);
[ntimes,nsensors,xdim2,xdim3]=size(Y);
data=zeros(10,ntimes);
pdata=zeros(10,ntimes); %predicted data
edata=zeros(10,ntimes); %estimated data
tdata=zeros(10,ntimes); %true data
esttimesigma=zeros(1,ntimes);
ntarget = NUM_TARGETS;
xmin = 4500;
xmax = MAX_TIME;

warning off;
% Calculating and storing the data for each time-step
for i=1:ntimes
    P=squeeze(Y(i,:,:));
    x=y(i,:);

    Pp=squeeze(Yp(i,:,:));
    xp=yp(i,:);

    data(1,i)=times(i);
    data(2,i)=(x(1)-targets(i,ntarget).x)*180;
    data(3,i)=(x(3)-targets(i,ntarget).y)*180;
    sigma=real(sqrt(P));
    data(4,i)=sigma(1,1)*180;
    data(5,i)=sigma(3,3)*180;
    data(6,i)=(real(sqrt(x(2)*x(2)+x(4)*x(4)))-targets(i,ntarget).vel)*180;
    data(7,i)=atan2(x(4),x(2))-targets(i,ntarget).phi;
    if abs(data(7,i)) > pi
        data(7,i) = 2*pi-abs(data(7,i));
    end
end

```

```

end
data(8,i)=sqrt(sigma(1,1)*sigma(1,1)+sigma(3,3)*sigma(3,3))*180;
data(9,i)=sqrt(sigma(2,2)*sigma(2,2)+sigma(4,4)*sigma(4,4))*180;
data(10,i)=real(sqrt(((x(2)/(x(2)^2 + x(4)^2))^2*(sigma(4,4)^2)...
+((x(4)/(x(2)^2 + x(4)^2))^2*(sigma(2,2)^2)))));

tdata(1,i)=times(i);
tdata(2,i)=targets(i,1).x*180;
tdata(3,i)=targets(i,1).y*180;
tdata(6,i)=targets(i,1).vel*180;
tdata(7,i)=targets(i,1).phi;

pdata(1,i)=times(i);
pdata(2,i)=xp(1)*180;
pdata(3,i)=xp(3)*180;
psigma=sqrt(Pp);
pdata(4,i)=psigma(1,1)*180;
pdata(5,i)=psigma(3,3)*180;
pdata(6,i)=sqrt(xp(2)*xp(2)+xp(4)*xp(4))*180;
pdata(7,i)=atan2(xp(4),xp(2));
if abs(pdata(7,i)) > pi
    pdata(7,i) = 2*pi-abs(pdata(7,i));
end
pdata(8,i)=sqrt(psigma(1,1)*psigma(1,1)+psigma(3,3)*psigma(3,3))*180;
pdata(9,i)=sqrt(psigma(2,2)*psigma(2,2)+psigma(4,4)*psigma(4,4))*180;
pdata(10,i)=sqrt((((xp(2))/(xp(2)^2+xp(4)^2))^2*(psigma(4,4)*psigma(4,4)))...
+((((xp(4))/(xp(2)^2+xp(4)^2))^2*(psigma(4,4)*psigma(2,2)))));

edata(1,i)=times(i);
edata(2,i)=x(1)*180;
edata(3,i)=x(3)*180;
esigma=real(sqrt(P));
edata(4,i)=esigma(1,1)*180;
edata(5,i)=esigma(3,3)*180;
edata(6,i)=(real(sqrt(x(2)*x(2)+x(4)*x(4))))*180;
edata(7,i)=atan2(x(4),x(2));
if abs(edata(7,i)) > pi
    edata(7,i) = 2*pi-abs(edata(7,i));
end
edata(8,i)=sqrt(esigma(1,1)*esigma(1,1)+esigma(3,3)*esigma(3,3))*180;
edata(9,i)=sqrt(esigma(2,2)*esigma(2,2)+esigma(4,4)*esigma(4,4))*180;
edata(10,i)=real(sqrt(((x(2)/(x(2)^2 + x(4)^2))^2*(esigma(4,4)^2)...
+((x(4)/(x(2)^2 + x(4)^2))^2*(esigma(2,2)^2)))));

esttimesigma(1,i)=sqrt(edata(8,i)^2+...

```



```

        (kdata(6,i)^2)*(edata(9,i)^2))/...
        edata(6,i);
end

% merging the heading standard deviations for plotting
p=1;q=1;i=1;
for j=1:(ntimes)*2
    if j/2-round(j/2)==0
        combtimedata(1,j)= times(i);
        combddata(8,j)=edata(8,p);
        combddata(9,j)=edata(9,p);
        combddata(10,j)=edata(10,p);
        p=p+1;
        i=i+1;
    else
        combtimedata(1,j)= times(i);
        combddata(8,j)=pdata(8,q);
        combddata(9,j)=pdata(9,q);
        combddata(10,j)=pdata(10,q);
        q=q+1;
    end
end

% Plot
figure(nfigure);
clf;
plot(data(1,:),abs(data(2,:)), 'b', data(1,:), data(4,:), 'r')
legend('Estimated Error', 'Standard Deviation');
buf=sprintf('Error between X-component Estimate and Truth');
title(buf)
xlim([xmin xmax])
axis 'auto y'
xlabel('Simulated Time (s)')
ylabel('X error (m)')

figure(nfigure+1);
clf;
plot(data(1,:),abs(data(3,:)), 'b', data(1,:), data(5,:), 'r')
legend('Estimated Error', 'Standard Deviation');
buf=sprintf('Error between Y-component Estimate and Truth');
title(buf)
xlim([xmin xmax])
axis 'auto y'
xlabel('Simulated Time (s)')
ylabel('Y error (m)')

```

```

figure(nfigure+2);
clf;
plot(data(1,:),abs(data(6,:)), 'b', data(1,:), data(9,:), 'r')
legend('Estimated Error', 'Standard Deviation');
buf=sprintf('Error between Velocity Estimate and Truth');
title(buf)
xlim([xmin xmax])
axis 'auto y'
xlabel('Simulated Time (s)')
ylabel('Velocity error (m/s)')

figure(nfigure+3);
clf;
plot(data(1,:),abs(data(7,:)), 'b', data(1,:), data(10,:), 'r')
legend('Estimated Error', 'Standard Deviation');
buf=sprintf('Error between Heading Estimate and Truth');
title(buf)
xlim([xmin xmax])
axis 'auto y'
xlabel('Simulated Time (s)')
ylabel('Heading error (rads)')

figure(nfigure+11);
clf;
plot(edata(1,:),abs((MAX_TIME-edata(1,:))-
kdata(6,:)), 'b', edata(1,:), esttimesigma(1,:), 'r')
legend('Estimated Error', 'Standard Deviation');
buf=sprintf('Error between Time Estimate and Truth');
title(buf)
xlim([min(kdata(7,1000:MAX_TIME/10))*DT xmax-1*DT])
axis 'auto y'
xlabel('Simulated Time (s)')
ylabel('Time (s)')

warning on;

```

```

% Acknowledgement:
% This code originates from H. Durrant-Whyte
% Australian Centre for Field Robotics
% The University of Sydney, Australia

% This function plots the network nodes and communication links

function plot_coms(platforms,com_net)

globals;

[nsamps,nplatforms]=size(platforms);

figure(14)
clf;
hold on
data=zeros(2,nplatforms);
ploc=zeros(2,2);
title('Platforms and Communication Links')
xlabel('x-position (m)')
ylabel('y-position (m)')

for n=1:nplatforms
    data(1,n)=platforms(1,n).x;
    data(2,n)=platforms(1,n).y;
end
plat=plot(data(1,:),data(2:,:),'ko');

for i=1:nplatforms
    for j=i:nplatforms
        if com_net(i,j)
            ploc(1,1)=data(1,i);
            ploc(1,2)=data(1,j);
            ploc(2,1)=data(2,i);
            ploc(2,2)=data(2,j);
        end
        links=plot(ploc(1,:),ploc(2:,:),'r');
    end
end

legend([plat,links],'Sensors','Communication Links');
hold off

```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

- [1] C. H. Teo. (2009, Jan 16). Speech at the Commissioning Ceremony of RSS Stalwart and RSS Supreme. Available:
http://www.mindef.gov.sg/content/imindef/news_and_events/nr/2009/jan/16jan09_nr2/16jan09_speech.print.html?Status=1 (accessed on 28 Apr 2011)
- [2] R. Lim. (2011, Jan 6). Speech at the Singapore Maritime Foundations's New Year Cocktail Reception. Available:
http://www.mpa.gov.sg/sites/global_navigation/news_center/speeches/speeches_detail.page?filename=sp110106.xml (accessed on 8 Jan 2011)
- [3] U.S. Energy Information Administration. (2008, Jan). World Oil Transit Chokepoints: Strait of Malacca [Online]. Available:
http://www.eia.doe.gov/emeu/cabs/World_Oil_Transit_Chokepoints/Background.html (accessed 8 Jan 2010)
- [4] P. Mukundan. (2010, Apr 21). Worldwide Hijackings Rise as Pirates Expand Area of Operation [Online]. Available: <http://www.icc-ccs.org/news/406-worldwide-hijackings-rise-as-pirates-expand-area-of-operation> (accessed 8 Jan 2011)
- [5] P. Chalk, "The Maritime Dimension of International Security: Terrorism, Piracy, and Challenges for the United States," RAND Corporation, 2008.
- [6] Systems Engineering and Analysis Cohort Nine, "Maritime Threat Response," Naval Postgraduate School, Monterey, CA, NPS-97-06-004, 2006.
- [7] Systems Engineering and Analysis Cohort Seven, "Maritime Domain Protection in the Straits of Malacca," Naval Postgraduate School, Monterey, CA, NPS-97-05-001, 2005.
- [8] R. L. Tomberlin, "Terrorism's Effect on Maritime Shipping," May 20, 2008.
- [9] G. W. Bush, "National Strategy for Maritime Security: National Plan to Achieve Maritime Domain Awareness," Washington D.C., The White House, Oct 2005.
- [10] T. V. Huynh and J. S. Osmundson, "A Systems Engineering Methodology for Analyzing Systems of Systems," Proc. of 1st Annual System of Systems Engineering Conf., Johnstown, PA, Jun 13–14, 2005.
- [11] T. V. Huynh and J. S. Osmundson, "An Integrated Systems Engineering Methodology for Analyzing Systems of Systems Architectures," Proc. of the Asia-Pacific Systems Engineering Conf., Singapore, Mar 23–24, 2007.

- [12] G. Galdorisi and R. Goshorn, "Maritime Domain Awareness: The Key to Maritime Security Operational Challenges and Technical Solutions," 11th International Command and Control Research and Technology Symp., San Diego, CA, 2006.
- [13] Director, Force Transformation, Office of the Secretary of Defense, "The Implementation of Network-Centric Warfare," Washington D.C., Jan 5, 2005.
- [14] E. Charniak, C. K. Riesbeck, and D. V. McDermott, "Artificial Intelligence Programming", Lawrence Erlbaum Associates, Hillsdale, NJ, 1979.
- [15] E. Waltz and J. Llinas, *Multisensor Data Fusion*. Boston: Artech House, 1990, ch 1, p. 2.
- [16] M. E. Liggins and C. C. Kuo, "Distributed Fusion Architectures, Algorithms, and Performance within a Network-Centric Architecture," in *Handbook of Multisensor Data Fusion: Theory and Practice*, M. E. Liggins, D. L. Hall and J. Llinas, Ed., 2nd ed. New York: CRC Press, 2009, ch. 17, pp. 411–433.
- [17] S. Bateman, C. Z. Raymond, and J. Ho, "Safety and Security in the Malacca and Singapore Straits: An Agenda for Action," Institute of Defence and Strategic Studies, Nanyang Technological University, May 2006.
- [18] S. Bateman, J. Ho, and M. Mathai, "Shipping Patterns in the Malacca and Singapore Straits: An Assessment of the Risks to Different Types of Vessel," in *Contemporary Southeast Asia: A Journal of International and Strategic Affairs*, Institute of Southeast Asian Studies, Aug 2007, vol. 29, no. 2, pp. 309–332.
- [19] Maritime and Port Authority of Singapore, "Safety of Navigation in the Singapore Strait," Port Marine Circular, no. 20/2006, Nov 14, 2006.
- [20] SimPlus Pte. Ltd., "Working Paper for 'Carriage Capacity of the Straits of Malacca and Singapore'," Singapore, Oct 2009.
- [21] C. D. Epp, "Protection Against a Ship as a Weapon," M.S. thesis, Naval Postgraduate School, Monterey, CA, 2009.
- [22] H. K. Lim. (2009, Sep 25). Keynote address at Jurong Island Reclamation Completion Ceremony [Online]. Available: <http://app.mti.gov.sg/default.asp?id=148&articleID=19921> (accessed on 30 Apr 2011)
- [23] Singapore Economic Development Board, "Jurong Island Factsheet 2011," Singapore, 2011.
- [24] Maritime and Port Authority of Singapore, "Singaporean Notices to Mariners," Notices to Mariners, ed. no. 10/2008, p. 212, Oct 1, 2008.

- [25] Maritime and Port Authority of Singapore, "Prohibition on Movement of Vessels in Waters Surrounding (A) Jurong Island," Port Marine Circular, no. 21/2006, Dec 8, 2006.
- [26] Google Maps. Jurong Island [Online]. Available: <http://maps.google.com> (accessed on 16 Jan 2011)
- [27] Maritime Port Authority of Singapore. (2011). Port Statistics [Online]. Available: http://www.mpa.gov.sg/sites/global_navigation/publications/port_statistics/port_statistics.page (accessed on 24 Jan 2011)
- [28] J. Porter. (2007, Mar 23). Investigators scrutinise MSC Napoli boxes [Online]. Available: www.lloydslist.com/art/20017413448 (accessed on 24 Jan 2011)
- [29] G. G. Ong, "Ships Can Be Dangerous, Too: Coupling Piracy and Terrorism in Southeast Asia's Maritime Security Framework" in *Piracy in Southeast Asia: Status, Issues and Responses*, D. Johnson and M. Valencia, Ed., Singapore: Institute of Southeast Asian Studies, 2005, p. 52.
- [30] Ministry of Defence, Singapore. (2006, Aug 29). Factsheet: Maritime Port Authority (MPA) [Online]. Available: http://www.mindef.gov.sg/imindef/news_and_events/nr/2006/aug/29aug06_nr/29aug06_fs2.html (accessed on 30 Apr 2011)
- [31] Ministry of Defence, Singapore. (2010, Aug 10). Navy Assets [Online]. Available: http://www.mindef.gov.sg/content/imindef/mindef_websites/atozlistings/navy/assets/vessels.html (accessed on 30 Apr 2011)
- [32] Jane's Fighting Ships. (2010, 17 Aug). Singapore [Online].
- [33] K. S. Wong. (2000, Jan 29). The Launch of Police Coast Guard's New Patrol and Interceptor Boats at the Police Coast Guard HQ [Online]. Available: http://www.mha.gov.sg/news_details.aspx?nid=MTEEx-xVye8N73E4U%3D (accessed on 29 Jan 2011)
- [34] Y. N. Hoe. (2009, Jul 16). Police Coast Guard Beefs Up Operational Capability [Online]. Available: <http://www.channelnewsasia.com/stories/singaporelocalnews/view/442778/1.html> (accessed on 29 Jan 2011)
- [35] Maritime and Port Authority of Singapore, "Revised Reporting Procedures for Notification and Confirmation of a Vessel's Arrival in Singapore," Port Marine Circular, no. 20/2008, Nov 10, 2008.
- [36] R. Sweet. Command and Control Evaluation Workshop, MORS C2 MOE Workshop. Military Operations Research Society, Jan 1985.

- [37] B. Bruegge and A. H. Dutoit, *Object-Oriented Software Engineering: Using UML, Patterns, and Java*. Upper Saddle River, Pearson, NJ: Prentice Hall, 2004.
- [38] H. B. Mitchell, *Multi-Sensor Data Fusion: An Introduction*. NY: Springer, 2007, ch. 3, pp. 38–44.
- [39] S. Julier and J. K. Uhlmann, “General Decentralized Data Fusion with Covariance Intersection,” in *Handbook of Multisensor Data Fusion: Theory and Practice*, M. E. Liggins, D. L. Hall and J. Llinas, Ed., 2nd ed. New York: CRC Press, 2009, ch. 14, pp. 319–342.
- [40] J. R. Raol, *Multi-Sensor Data Fusion with MATLAB*, New York: CRC Press, 2010, pp. 21–22.
- [41] A. N. Steinberg, C.L. Bowman, and F.E. White, “Revisions to the JDL Model,” Joint NATO/IRIS Conf. Procs., Quebec, Oct 1998 and in *Sensor Fusion: Architectures, Algorithms, and Applications*, Procs. SPIE, Vol. 3719, 1999.
- [42] D. L. Hall and J. Llinas, “Multisensor Data Fusion,” in *Handbook of Multisensor Data Fusion: Theory and Practice*, M. E. Liggins, D. L. Hall and J. Llinas, Ed., 2nd ed. New York: CRC Press, 2009, ch. 1, pp. 1–13.
- [43] H. Durrant-Whyte, Multi Sensor Data Fusion, Australian Centre for Field Robotics, The University of Sydney NSW 2006, Australia, January 22, 2001, ver. 1.2.
- [44] Y. Bar-Shalom and T. E. Fortmann. *Tracking and Data Association*. Academic Press, 1988.
- [45] P. S. Maybeck. *Stochastic Models, Estimation and Control, Vol. I*. Academic Press, 1979.
- [46] Y. Bar-Shalom and X. Li. *Multitarget-Multisensor Tracking: Principles and Techniques*. YBS Publishing, 1995.
- [47] J.S. Przemieniecki. *Introduction to Mathematical Methods in Defense Analyses*. AIAA Education Series, 1990.
- [48] J. Manyika and H. Durrant-Whyte. *Data Fusion and Sensor Management: A Decentralized Information – Theoretic Approach*. Ellis Horwood Series in Electrical and Electronic Engineering, 1995.
- [49] H. Chen, T. Kirubarajan, and Y. Bar-Shalom, “Performance Limits of Track-to-track Fusion vs. Centralized Estimation: Theory and Application.” *IEEE Trans. Aerosp. Electron. Syst.*, 39, pp. 386–40, Apr 2003.
- [50] C. M. Rogers, “Methods of Multisensor Tracking,” *Proc. London Communications Symp.* 2003

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California
3. Thomas V. Huynh
Naval Postgraduate School
Monterey, California
4. Joseph Rice
Naval Postgraduate School
Monterey, California