

Adaptive Sampling in Autonomous Marine Sensor Networks

by

Donald Patrick Eickstedt

B.S., Electrical Engineering, Michigan Technological University (1987)

M.S., Computer Science, The Johns Hopkins University (1993)

Submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

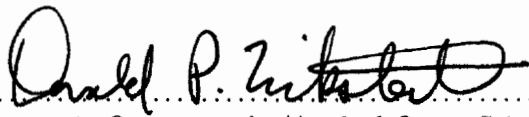
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

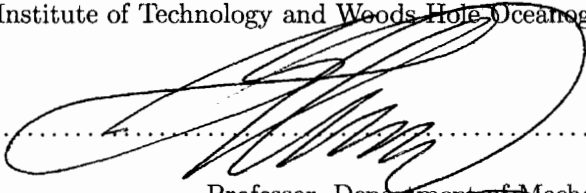
and the

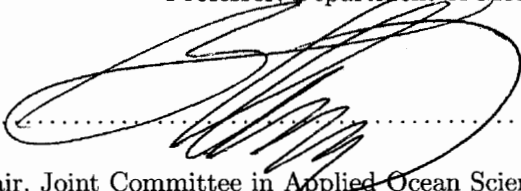
WOODS HOLE OCEANOGRAPHIC INSTITUTION

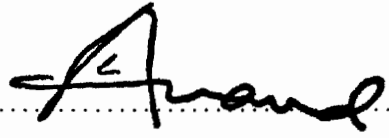
June 2006

© Massachusetts Institute of Technology 2006. All rights reserved.

Author 
Joint Program in Oceanography/Applied Ocean Science and Engineering
Massachusetts Institute of Technology and Woods Hole Oceanographic Institution
June, 2006

Certified by 
Henrik Schmidt
Professor, Department of Mechanical Engineering
Thesis Supervisor

Accepted by 
Henrik Schmidt
Chair, Joint Committee in Applied Ocean Science and Engineering
Massachusetts Institute of Technology and Woods Hole Oceanographic Institution

Accepted by 
Lallit Anand
Chair, Committee on Graduate Students, Department of Mechanical Engineering

Adaptive Sampling in Autonomous Marine Sensor Networks

by

Donald Patrick Eickstedt

Submitted to the Joint Program in Oceanography/Applied Ocean Science and
Engineering

and the Massachusetts Institute of Technology and Woods Hole Oceanographic
Institution

on June, 2006, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

In this thesis, an innovative architecture for real-time adaptive and cooperative control of autonomous sensor platforms in a marine sensor network is described in the context of the autonomous oceanographic network scenario. This architecture has three major components, an intelligent, logical sensor that provides high-level environmental state information to a behavior-based autonomous vehicle control system, a new approach to behavior-based control of autonomous vehicles using multiple objective functions that allows reactive control in complex environments with multiple constraints, and an approach to cooperative robotics that is a hybrid between the swarm cooperation and intentional cooperation approaches. The mobility of the sensor platforms is a key advantage of this strategy, allowing dynamic optimization of the sensor locations with respect to the classification or localization of a process of interest including processes which can be time varying, not spatially isotropic and for which action is required in real-time.

Experimental results are presented for a 2-D target tracking application in which fully autonomous surface craft using simulated bearing sensors acquire and track a moving target in open water. In the first example, a single sensor vehicle adaptively tracks a target while simultaneously relaying the estimated track to a second vehicle acting as a classification platform. In the second example, two spatially distributed sensor vehicles adaptively track a moving target by fusing their sensor information to form a single target track estimate. In both cases the goal is to adapt the platform motion to minimize the uncertainty of the target track parameter estimates. The link between the sensor platform motion and the target track estimate uncertainty is fully derived and this information is used to develop the behaviors for the sensor platform control system. The experimental results clearly illustrate the significant processing gain that spatially distributed sensors can achieve over a single sensor when observing a dynamic phenomenon as well as the viability of behavior-based control for dealing with uncertainty in complex situations in marine sensor networks.

Thesis Supervisor: Henrik Schmidt

Title: Professor, Department of Mechanical Engineering

Acknowledgments

This work is the result of not only my own effort but also that of the numerous individuals who have supported me along the way without whom this work would never have begun or would never have been finished.

First and foremost I thank my wife Margaret whose love and support have enabled me to accomplish one of my life's goals. You've been the force that's kept our family together through the past seven years. I can never repay the sacrifices you've made and I will never forget them either.

I would like to thank my adviser Henrik Schmidt for having the faith to choose me for his group and for letting me have as much responsibility as I could handle. Your patience with me as I struggled with this program at times and your support at several critical times is very appreciated and won't be forgotten. The same goes for Professor John Leonard whose key support at at least one critical point helped keep me in this program. Your insight and advice through the years is also greatly appreciated.

I would like to thank my other committee members Mike Benjamin and Hanumant Singh for your efforts on my committee. Even though you're both very busy I appreciate that you took the time to help guide me through this last part of my program. Your advice was invaluable. I would also like to thank Mike for his collaboration on a number of research experiments which appear in this report. I look forward to more collaboration in the future.

I would also like to thank my sponsor, the Office of Naval Research, who supported my work with a 3-year National Defense Science and Engineering Grant Fellowship and funded my research assistantships through the Generic Ocean Array Technology Sonar (GOATS) project, contract N00014-97-1-0202 and contract N00014-05-G-0106 Delivery Order 008, PLUSNET: Persistent Littoral Undersea Surveillance Network.

Contents

1	Introduction	17
1.1	Requirements for a Marine Sampling Network	20
1.1.1	Mobility	20
1.1.2	Adaptivity	21
1.1.3	Communications	23
1.1.4	Cooperation	23
1.1.5	Sensor Fusion	24
1.2	Other Adaptive Sampling Work	25
1.3	Preview of Results	25
1.4	Thesis Organization	26
2	Autonomous Oceanographic Sampling Networks	27
2.1	System Concept	28
2.1.1	Number of Sensor Platforms Required.	29
2.1.2	Sensor Platform Type	30
2.1.3	Acoustic Communications and Navigation.	32
2.1.4	Vehicle cost.	33
2.2	Enabling Technologies.	34
2.3	Historical Developments	34
2.3.1	The AOSN Project	34
2.3.2	The AOSN-II Project	35
2.3.3	The NEPTUNE Project	36
2.4	Relevance of AOSN to this Thesis	36

3	Adaptive Sensor Platform Control	39
3.1	The World Model-Based Approach	40
3.2	Behavior-Based Approaches	41
3.2.1	Subsumption Architecture	41
3.2.2	State-Configured Layered Control	42
3.2.3	Motor Schemas	43
3.2.4	The Multiple Objective Function Approach	44
3.3	Cooperative Control	45
3.3.1	Swarm Cooperation	46
3.3.2	Intentional Cooperation	47
3.3.3	Formation/Flocking Behaviors	47
4	The MOOS-IvP Autonomy Architecture	51
4.1	A Distributed Control Architecture for Marine Sensor Platforms	51
4.1.1	Behavior-Based Control with Interval Programming	51
4.1.2	The MOOS-IvP Autonomy Architecture	52
4.2	Sensor Platform Behaviors	53
4.2.1	The OpRegion Behavior	54
4.2.2	The Waypoint Behavior	54
4.2.3	The Orbit Behavior	54
4.2.4	The ArrayTurn Behavior	55
4.2.5	ArrayAngle Behavior	57
4.2.6	CloseRange Behavior	59
4.2.7	Classify Behavior	59
4.2.8	Formation Behavior	60
5	An AUV Intelligent Sensor for Real-Time Adaptive Sensing	63
5.1	A Logical Sonar Sensor	63
5.2	Design Goals for the Physical Sonar Sensor	65
5.3	Sonar Implementation	67
5.3.1	Mechanical/Electrical	68
5.3.2	Receiving Array Integration	68
5.3.3	Analog Processing Section	69

5.3.4	Host Computer Architecture	69
5.3.5	Data Acquisition Subsystem	70
5.3.6	Time Reference Subsystem	72
5.3.7	Acoustic Source	73
5.3.8	Software Architecture	73
5.4	Experimental Results	74
6	Example One: Adaptive Track and Classify	79
6.1	Introduction	79
6.2	Target Tracking with a Single Bearing Sensor	80
6.2.1	State Estimator Derivation	80
6.2.2	The Likelihood Function	82
6.2.3	The Cramer-Rao Lower Bound	83
6.2.4	Parameter Observability	84
6.2.5	Covariance of the Target Position Estimate	85
6.3	Experimental Setup	86
6.3.1	Simplifying Assumptions	86
6.3.2	The Marine Vehicle Platforms	86
6.3.3	Scenario	87
6.3.4	Behavior Configurations	87
6.4	Experimental Results	89
6.4.1	Mission 1507	89
6.4.2	Mission 1444	95
6.4.3	Mission 1422	101
7	Example Two: Adaptive Tracking with Multiple Sensors	107
7.1	Introduction	107
7.2	Bearings-Only Target Tracking with Two Sensors	108
7.2.1	2D Target Position Triangulation	109
7.2.2	Variance of the Target Position Estimate	110
7.2.3	Target Velocity Component Estimation	114
7.2.4	Scenario	117
7.2.5	Behavior Configurations	117

7.2.6	Kalman Filter Initialization	118
7.3	Experimental Results	118
7.3.1	Mission 1448	118
7.3.2	Mission 1144	125
7.3.3	Mission 1121	132
8	Summary and Conclusions	139
8.1	Summary of Contributions	139
8.2	Conclusions	141
8.3	Future Work	142
A	Supporting Equations	145
B	Behavior Code	147
B.1	ArrayTurn Behavior	147
B.1.1	BHV_ArrayTurn.h	147
B.1.2	BHV_ArrayTurn.cpp	148
B.1.3	AOF_ArrayTurn.h	150
B.1.4	AOF_ArrayTurn.cpp	151
B.2	ArrayAngle Behavior	154
B.2.1	AOF_ArrayAngle.h	154
B.2.2	AOF_ArrayAngle.cpp	155
B.2.3	BHV_ArrayAngle.h	159
B.2.4	BHV_ArrayAngle.cpp	160
B.3	CloseRange Behavior	164
B.3.1	BHV_CloseRange.h	164
B.3.2	BHV_CloseRange.cpp	165
B.4	Orbit Behavior	170
B.4.1	AOF_DonWpt2D.h	170
B.4.2	AOF_DonWpt2D.cpp	171
B.4.3	BHV_Orbit.h	173
B.4.4	BHV_Orbit.cpp	174
B.5	Formation Behavior	180

B.5.1	BHV_2VAngle.h	180
B.6	Formation Behavior	181
B.6.1	BHV_2VAngle.cpp	181
C	Sample Mission Files	187
C.1	MOOS Files	187
C.1.1	December 4, Mission 1507 - Sensor Vehicle 200	187
C.1.2	December 4, Mission 1507 - Target Vehicle 201	193
C.1.3	December 4, Mission 1507 - Classification Vehicle 206	198
C.2	Behavior Files	204
C.2.1	December 4, Mission 1507 - Sensor Vehicle 200	204
C.2.2	December 4, Mission 1507 - Target Vehicle 201	205
C.2.3	December 4, Mission 1507 - Classification Vehicle 206	206

List of Figures

1-1	Acoustic insonification of a sphere and cylinder.	19
1-2	Non-adaptive sampling paths for an AUV	22
2-1	An autonomous oceanographic sampling network	28
2-2	An Odyssey-III AUV	31
2-3	The Autonomous Benthic Explorer (ABE)	31
2-4	The Slocum glider.	32
3-1	Model-Based vs. Behavior-Based Robot Control Cycles	40
3-2	A depiction of the subsumption architecture.	41
3-3	The go-to-goal and obstacle motor schemas.	43
3-4	The vector sum of the goal and obstacle motor schemas.	44
3-5	The multiple objective function approach to action selection.	45
4-1	A MOOS community and the pHelmIvP process.	53
4-2	The Orbit behavior.	55
4-3	The ArrayTurn and ArrayAngle behaviors	56
4-4	Objective Function for the ArrayTurn Behavior	56
4-5	Objective Function for the ArrayTurn Behavior	57
4-6	Objective Function for the ArrayAngle Behavior	58
4-7	Objective Function for the ArrayAngle Behavior	58
4-8	Dynamic Weighting for the ArrayAngle Behavior	59
4-9	Dynamic Weighting for the CloseRange Behavior	60
4-10	The Formation behavior.	61
4-11	Formation Behavior Metric	61

5-1	The logical sonar sensor	64
5-2	AUVs with single and dual line arrays.	66
5-3	Real-time sonar architecture	67
5-4	Sonar pressure vessel in Odyssey-III payload section.	68
5-5	Heron-4 DSP board and rubidium oscillator.	70
5-6	Sonar pressure vessel in Odyssey-III payload section.	71
5-7	Online target detection with the sonar payload in active MCM mode.	75
5-8	Spectrogram of a multi-static ping reception.	76
5-9	Real-Time Target Bearings in FAF 05.	77
5-10	Triangulation of acoustic source in FAF 05.	78
6-1	The track and classify mission scenario.	80
6-2	Parameter observability based on sensor motion.	84
6-3	Bearing simulator uncertainty.	87
6-4	The kayak-based autonomous surface craft.	88
6-5	Platform motion for track and classify mission 1507.	90
6-6	Track solution results for mission 1507.	91
6-7	Target localization error for mission 1507.	92
6-8	Classification vehicle distance from target position estimate for mission 1507.	92
6-9	Fisher information about x_0 and x_1 for mission 1507.	93
6-10	Fisher information about x_2 and x_3 for mission 1507.	93
6-11	Condition number of the Fisher information matrix for mission 1507.	94
6-12	Platform motion for track and classify mission 1444.	96
6-13	Track solution results for mission 1444.	97
6-14	Target localization error for mission 1444.	98
6-15	Classification vehicle distance from target position estimate for mission 1444.	98
6-16	Fisher information about x_0 and x_1 for mission 1444.	99
6-17	Fisher information about x_2 and x_3 for mission 1444.	99
6-18	Condition number of the Fisher information matrix for mission 1444.	100
6-19	Platform motion for track and classify mission 1422.	102
6-20	Track solution results for mission 1422.	103
6-21	Target localization error for mission 1422.	104

6-22	Classification vehicle distance from target position estimate for mission 1422. . . .	104
6-23	Fisher information about x_0 and x_1 for mission 1422.	105
6-24	Fisher information about x_2 and x_3 for mission 1422.	105
6-25	Condition number of the Fisher information matrix for mission 1422.	106
7-1	The multi-sensor tracking scenario.	107
7-2	Coordinate frame for 2D multi-sensor tracking.	109
7-3	A plot of coefficient C_1	112
7-4	A plot of coefficient C_2 for 10m range.	112
7-5	A plot of coefficient C_2 for 20m range.	113
7-6	A plot of coefficient C_3	113
7-7	A plot of coefficient C_4	114
7-8	Vehicle motion for mission 1448.	120
7-9	Target track results for mission 1448.	121
7-10	Target track error for mission 1448.	122
7-11	Formation angle for mission 1448.	122
7-12	Target range for sensor platform one for mission 1448.	123
7-13	Target range for sensor platform two for mission 1448.	123
7-14	Speed estimate error for mission 1448.	124
7-15	Heading estimate error for mission 1448.	124
7-16	Vehicle motion for mission 1144.	127
7-17	Target track results for mission 1144.	128
7-18	Target track error for mission 1144.	129
7-19	Formation angle for mission 1144.	129
7-20	Target range for sensor platform one for mission 1144.	130
7-21	Target range for sensor platform two for mission 1144.	130
7-22	Speed estimate error for mission 1144.	131
7-23	Heading estimate error for mission 1144.	131
7-24	Vehicle motion for mission 1121.	134
7-25	Target track results for mission 1121.	135
7-26	Target track error for mission 1121.	136
7-27	Formation angle for mission 1121.	136

7-28	Target range for sensor platform one for mission 1121.	137
7-29	Target range for sensor platform two for mission 1121.	137
7-30	Speed estimate error for mission 1121.	138
7-31	Heading estimate error for mission 1121.	138
8-1	Target tracking with N sensors.	143
8-2	Multi-target tracking with multiple sensors.	143

Chapter 1

Introduction

The oceans are of vital importance to life on earth. They are a key component of the earth's climate and weather, a significant source of food, a highway over which much of the world's commerce travels, and an area which is vitally important for the national defense of many countries. The continental shelves hold vast reservoirs of energy, the deep ocean is thought to contain large deposits of important minerals, and many biologists believe there are deep sea organisms that hold great bio-pharmaceutical potential. And yet, more is probably known about the moon and other bodies in our solar system than is known about the oceans right here on earth.

A major reason for this lack of knowledge is the inhospitable nature of working in the ocean which rivals or exceeds that of working in space. The crushing pressures aside, it is the physical nature of the fluid medium of the ocean which puts great limitations on our ability to sense what is there. A great many of the sensing technologies developed for use on land utilize either optical or radio frequency (RF) energy. In the ocean, electromagnetic energy at both RF and optical frequencies is highly attenuated. Light is also highly attenuated by the ocean's turbidity, caused by tiny suspended particles of plankton or sediment which scatter or absorb the light energy. The tendency of the ocean to absorb RF energy also hampers navigation of underwater sensor platforms. On land, systems such as GPS can be used for the precision location of sensors to within a few meters while no system as accurate as GPS has been developed for navigating underwater.

Due to the relatively high density of seawater, acoustic energy is propagated fairly well and its use is one of the key methods used for underwater sensing, navigation, and communications. Acoustic propagation in the ocean is, however, significantly affected by the ocean's physical prop-

erties. Attenuation of acoustic energy by seawater increases greatly with frequency, relegating most applications to the lower frequency bands or short distances. Boundary conditions caused by the sea surface and sea bottom, the varying nature of the sound speed in the ocean caused mainly by temperature variations, and the tendency of the sea surface and the sea bottom to reflect acoustic energy lead to many difficult problems when trying to propagate acoustic energy for sensing, communications, and navigation. Other types of sensors are useful in the ocean environment as well including chemical, biological, magnetic, electromagnetic, conductivity, pressure, and thermal sensors. Each type of sensor will have its own characteristics modulated by the ocean environment.

This work is motivated by an interest in a fundamental problem in sensor system design and operation for sensing in the ocean which is also applicable to sensing systems on land and in space. That is, how can one sense processes or the characteristics of processes which are intentionally or unintentionally difficult to sense using a single sensor which can only sample the process from a single spatial location at a given instant in time. In this work we define a process as the field generated by a physical event or phenomenon which can be sensed by a physical sensor such as those described previously. Many processes of interest are time-varying and not spatially isotropic and, therefore, either the process itself or some of its characteristics may not be observable from a single sensor platform. For other processes, spatially distributed sensors can add significant processing gain, reducing the sensing time and improving our estimates of the process parameters.

For example, in a passive sonar context, single sensor platforms can localize contacts using passive bearing information but require temporal diversity to do so while multiple, distributed bearing sensors can immediately form a solution. In an active sonar context, the scattering of acoustic energy off of objects of different shapes is highly directional and is dependent on the spatial relationship between the source, receiver, and target. It is impossible for a single sensor platform carrying both source and receiver to capture the full scattered field which is useful in classifying the target shape. Fig. 1-1(a) and 1-1(b) show the acoustic energy scattered from both a sphere and a cylinder insonified by the beam from a sonar. As can be seen, the scattering from the cylinder is highly directional with no significant energy backscattered toward the acoustic source while the scattering from the sphere is more spatially isotropic. A group of distributed sensors would be able to capture the spatial distribution of this scattering for use in classification. For sampling transient oceanographic phenomena such as frontal dynamics, the spatial sampling

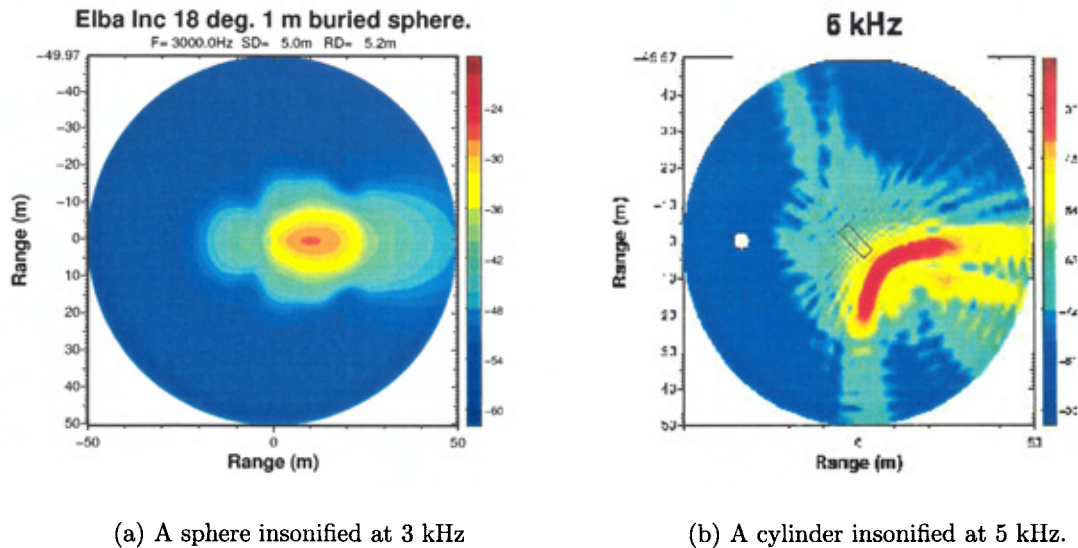


Figure 1-1: Acoustic insonification of a sphere and cylinder. As can be seen, the scattering from the cylinder is highly directional with no significant energy backscattered toward the acoustic source while the scattering from the sphere is more spatially isotropic.

resolution is related to the frequency content of the frontal process. Synoptic sampling coverage by multiple sensors can help avoid the temporal smearing that would occur in the data sampled by a single sensor platform. These examples, and numerous others, encompass a class of problems in marine sensing that can benefit from a multiple sensor approach.

In addition to being able to address the problems that cannot be solved using a single sensor, the use of mobile sensor platforms working in coordination offers several additional advantages. They may each have different payloads, sensors, and endurance capabilities. A network of small, inexpensive platforms with low-performance sensors may be able to use its spatial diversity to outperform systems using single, very expensive, high-performance sensors. The use of multiple platforms also may allow one platform to stay at the surface, with a higher bandwidth link to other robotic or human operated vehicles, while one or more other platforms operate under the surface at varying depths to optimize their sensor-oriented tasks. Network survivability is also enhanced as the loss of one or even possibly several inexpensive sensors can be absorbed with the redundancy inherent in such a network.

The question then remains as to how we can accomplish this sensing of dynamic phenomena in the ocean with multiple sensors. What would such a system look like and how would it behave? As difficult as it is to sense phenomena in the ocean with a single sensor, coordinating multiple

sensors seems a daunting challenge. We begin by attempting to define the requirements for such a system.

1.1 Requirements for a Marine Sampling Network

In this section we attempt to define some of the major requirements for a marine sampling network with the goal of being able to sense and characterize dynamic ocean phenomena both natural and man-made using multiple, cooperating sensor platforms. These requirements will lead us to define a system referred to as an autonomous oceanographic sampling network [1], described in greater detail in Chapter 2.

1.1.1 Mobility

One way to provide coverage of an area with multiple sensors would be to lay out a grid of fixed sensors all communicating back to a central data processing location. The grid spacing of the fixed sensors would be related to the process under observation. In fact, this method is used in many ocean sensing systems. For example, there is currently a network of ocean buoys that monitors parameters such as sea surface temperature, currents, conductivity, and ocean wave statistics for use in weather and hurricane forecasting systems. However, for many problems of interest, laying out a fine grid of fixed sensors is clearly impractical. This would be the case, for example, in applications where sensing must take place over a wide area with fine resolution or in deep water where the installation and maintenance costs of a sensor grid of the necessary size would be prohibitive. Fixed sensor systems are also not appropriate for applications where a temporary monitoring system is needed or in applications in which some action must be taken when certain conditions are sensed. The mobility of the sensor platforms is a key aspect of the adaptive sampling scenario, allowing dynamic optimization of the sensor locations with respect to the reduction in uncertainty of the process parameters we are attempting to estimate. A mobile sensor paradigm also allows resource optimization in scenarios where specialized sensors on mobile platforms can be brought to bear on a problem when more generalized sensors have made initial determinations. For example, in a mine countermeasures scenario, a network of low-frequency sonar platforms could localize a potential target and then call in additional sensor platforms with chemical sensors or sidescan sonars to gather additional information. In military

target tracking applications, kill vehicles could be vectored to a target by a network of sensors which are simultaneously tracking and classifying the target.

1.1.2 Adaptivity

In the absence of a fine grid of sensors which can spatially sample a phenomenon simultaneously from multiple points, the sensors must not only be mobile but they must also be able to autonomously adapt their motion in real time according to the sampled data. This requires tight coordination between the sensors and the vehicle control. Given that a sensor platform may carry multiple heterogeneous sensors, this requires a sensor integration model that abstracts sensory data for use by the sensor platform control system. In Chapter 5 we describe a sensor integration model that makes use of the concept of a logical sensor that abstracts away the details of the physical sensor. In Chapters 6 and 7 we use this model in two experiments using simulated bearing sensors where the output of the logical sensor is a target track.

Once a sensor platform receives sensory data, the platform control system must use this environmental state data to maneuver. Typically, our goal will be to maneuver the platform in such a way as to gain additional information about the process we are observing. This requires some sort of mapping between the environmental state data and the vehicle control parameters (rudder, elevator, speed, etc.) In Chapter 3, two major methods for doing this are described, the world model-based and the behavior-based methodologies. In the world model-based methodology, one large model is used to map the environmental state data to the control parameters. However, the very large state space inherent to a marine vehicle operating in any reasonably complex application is prohibitive for such an approach in my view. A sensor platform may be dealing not only with sensor data from an application specific sensor like an acoustic array but also with tasks like obstacle avoidance, path planning, and navigation. A direct mapping of sensor states to vehicle control variables is infeasible with such a large state space. A behavior-based control system, in contrast, uses a number of modular computing units termed "behaviors", all operating in parallel, to decide the vehicle's course of action during each control cycle. During a control cycle, each behavior will use the current sensor state data to compute its opinion on the next course of action. For example, each behavior may output its preferred course, speed, and depth for the vehicle. The issue then arises about how to select the preferred action when multiple behaviors disagree. One method would be to simply pick the output of the behavior

with the highest priority. This scheme was used by Brooks in his original layered control method [2]. This method, however, does not allow for the possibility of compromise between the preferred actions of different behaviors. In this work, we use a method for behavior-based control in which each behavior outputs its preferred action as an objective function over the vehicle control variables. During each control cycle, the preferred action is decided by performing a multi-function optimization over all of the objective functions. The optimization is performed using the Interval Programming Method (IvP) developed by Benjamin [3] to perform the optimization in a computationally efficient manner. This method is more fully described in Chapter 3.

The current state of the art in sampling with underwater vehicles is primarily limited to the use of non-adaptive, preplanned sampling missions where the collected data is stored for offline retrieval and analysis. Fig. 1-2 shows typical preplanned sampling paths for an autonomous underwater vehicle (AUV) used to sample environmental data off the coast of Elba, Italy. The AUV has no capability to react to data received from its environmental sensors other than the navigation sensors which keep it on its preplanned course.

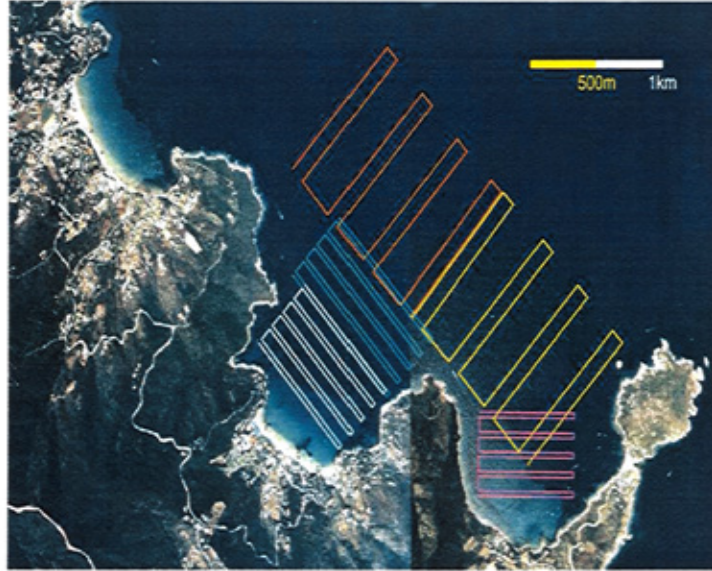


Figure 1-2: Several non-adaptive sampling paths for an AUV operating off the coast of Elba, Italy. Data is stored for offline retrieval. The sensor platform has no capability to react to the sensed data.

1.1.3 Communications

It seems an obvious conclusion that in order for multiple sensors to coordinate their actions and share state information, they must be able to communicate. This is easier said than done in the ocean environment however. On land, RF or fiber optic communications systems are capable of transmitting information on the order of megabits per second or greater. In the ocean, where RF energy is unusable over any distance and acoustic data transmission must be used, data transmission rates are orders of magnitude lower due to the propagation constraints imposed by the ocean environment (in particular, the relatively slow phase speed of acoustic waves). This directly impacts the amount of information that can be shared in a marine network and the types of network connectivity that can be used. Since all sensor platforms must share the same acoustic channel, this may also limit the number of platforms that can be active.

The amount of bandwidth needed in a cooperative sensor network is related to the sampling requirements of the process under observation. Processes with high frequency content require correspondingly high bandwidth. For processes with low frequency content, bandwidth requirements may be traded off for an increased sampling period. This issue is complicated by the fact that the acoustic channel may also be used simultaneously by sensors and navigation systems. Along with sensor platform navigation, a robust communications capability is one of the two critical supporting technologies needed to implement an effective sensor network.

1.1.4 Cooperation

While coordinated marine vehicles have their advantages, they present challenges in their joint control to reach their combined potential. Inter-vehicle communication is limited in bandwidth and carefully allocated. Any kind of central continuous control is likely infeasible. In multi-vehicle joint exercises involved with sensing dynamic phenomena, it may not be practical or effective to think in terms of a single vehicle state space to which proper actions can be assigned a priori. In Chapter 3 we describe a behavior-based control approach where a number of behaviors operating in parallel use the sensed environmental state data to maneuver the sensor platform. In this work, we use an approach to cooperation in which some of the behaviors on the sensor platforms are specifically designed to use state data from other sensor platforms in order to form a decision on preferred platform maneuvers. This state data is shared via the communications network. This is a form of highly decentralized cooperative control in which there is no central planner

dictating actions to the sensor platforms. This in keeping with the spirit of behavior-based control in which there is a tight coupling between control and the perceived environment. This scheme has an obvious advantage with respect to network survivability in that any network with central planning is vulnerable to the loss of the planner, whether that function resides on another sensor platform or on the surface. A network with decentralized control is more able to gracefully degrade with the loss of particular sensor nodes. More details on cooperation are discussed in Chapter 3.

1.1.5 Sensor Fusion

Data fusion is the synergistic combination of information from different sources such as sensors in order to provide a better understanding of the state of the world [4]. In our marine sensor network application, sensor data from multiple, distributed sensor platforms must be combined. These sensors may be heterogeneous and may have different resolutions. For example, data from both range and bearing sensors may need to be combined in surveillance and target tracking applications. In the target tracking example with distributed sensors discussed in Chapter 7, two independent bearing observations from distributed sensor platforms must be combined to estimate a target track. A significant issue in fusing data from multiple sources is in determining that distributed measurements correspond to the same environmental feature [4]. This is known as the data association problem. This issue arises for example in tracking applications where multiple targets may be present.

In order to properly fuse data from multiple sources and possibly heterogeneous sensors, accurate models of the process we want to observe and our sensor characteristics are imperative. Accurate process models allow us to derive the proper sensor platform behaviors given the state of the environment. These models must view the process from a probabilistic standpoint. It is not good enough to provide an estimate of a process parameter without also providing a notion of the uncertainty associated with the estimate. This also allows accurate simulation of adaptive sensor platform operation. The uncertainty of our estimates is related to a number of factors but primarily on the uncertainty of our sensor measurements, the uncertainty of the spatial location where the measurement was taken, and the time the measurement was taken. The uncertainty of the sensor measurements can be dealt with by having an accurate sensor model. The uncertainty in the measurement time can be easily dealt with by precision time synchronization between the

sensor platforms. A method for doing this is discussed in Chapter 5. As discussed previously, platform navigation is one of the two critical supporting technologies required in a sensor network. At the very least, the navigation uncertainty must remain bounded over the time period the sensor platform is in operation. If the navigation uncertainty grows over time this will introduce a growing uncertainty in our sensor measurements and hence in our estimates. This is clearly undesirable. A number of techniques for sensor platform navigation are discussed in Chapter 2.

1.2 Other Adaptive Sampling Work

A number of researchers and organizations have been conducting research into adaptive sampling systems and algorithms. The Woods Hole Oceanographic Institution has been investigating the use of AUVs for chemical plume tracing [5][6] and hydrothermal vent localization [7][8][9]. The Autonomous Undersea Systems Institute has been investigating adaptive sampling algorithms for oceanographic phenomena using both single and multiple cooperating AUVs [10][11]. Princeton University has been undertaken a major investigation into adaptive sampling with fleets of gliders for autonomous oceanographic sampling networks [12][13][14]. MIT Sea Grant has investigated the adaptive use of AUVs for long-term observation of ocean eddies [15]. A number of other investigators have researched generic techniques for adaptive sampling [16][17].

1.3 Preview of Results

In this work these challenges are addressed by presenting a novel architecture consisting of a network of sensor platforms each with an intelligent sensor supplying high-level environmental state data to a new type of behavior-based control system that is more suited to reactive control with multiple constraints than previous behavior-based implementations. Experimental results are presented for a 2-D target tracking application in which fully autonomous surface craft using simulated bearing sensors acquire and track a moving target in open water. In the first example, a single sensor vehicle adaptively tracks a target while simultaneously relaying the estimated track to a second vehicle acting as a classification platform. In the second example, two spatially distributed sensor vehicles adaptively track a moving target by fusing their sensor information to form a single target track estimate. In both cases the goal is to adapt the platform motion to minimize the uncertainty of the target track parameter estimates. The link between the sensor

platform motion and the target track estimate uncertainty is fully derived and this information is used to develop the behaviors for the sensor platform control system. The experimental results will clearly illustrate the significant processing gain that spatially distributed sensors can achieve over a single sensor when observing a dynamic phenomenon of interest. Behavior-based control is also shown as a viable method for dealing with uncertainty in complex control situations in marine sensor networks.

1.4 Thesis Organization

This thesis is organized in the following manner:

Chapter 2: Autonomous Oceanographic Sampling Networks. This chapter describes the ideas and the work to date regarding marine sensor networks originally envisioned in [1].

Chapter 3: Behavior-based Control of Autonomous Platforms. This chapter compares and contrasts two prevailing methods for robotic control, the world-model approach and the behavior-based approach as well as the various methods for action selection in behavior-based approaches. Here we describe the use of objective functions for action selection and the Interval Programming Method for computing the consensus action.

Chapter 4: The MOOS-IvP Autonomy Architecture. This chapter describes the MOOS-IvP autonomy architecture used on the autonomous sensor platforms.

Chapter 5: An Intelligent Acoustic Sensor. This chapter describes the concept of a logical sensor and the design of a real logical sensor for an AUV for use in a marine sensor network.

Chapter 6: Example One: Adaptive Track and Classify. This chapter describes an adaptive tracking experiment involving a tracking vehicle with a simulated bearing sensor which tracks a target and networks the target track estimate to another vehicle acting as a vehicle with a classification sensor which closes range with the target position estimate.

Chapter 7: Example Two: Adaptive Tracking with Multiple Sensors. This chapter describes an adaptive tracking experiment using two autonomous surface craft with simulated bearing sensors which cooperatively track a moving target.

Chapter 8: Summary and Conclusions. This chapter summarizes the original contributions of the thesis and gives an overview of future work.

Chapter 2

Autonomous Oceanographic Sampling Networks

By 1993, advances in robotics, communications, and sensor technology were reaching a critical mass that allowed, for the first time, the possibility of remote, unattended sampling of oceanic processes over wide areas. Autonomous underwater vehicles carrying a variety of sensors held promise for adaptive sampling of oceanographic processes and coupled ocean observation/modeling systems [1] [18]. Understandably, this possibility elicited excitement among not only physical scientists who wanted to gain a better understanding of oceanic processes but also among forward-thinking members of the defense establishment who saw a number of advantages. First, the understanding of basic oceanic processes and characteristics is vital in almost every aspect of conducting undersea warfare including, but not limited to, undersea communications and sonar performance. Second, having the capability of deploying undersea sensor networks opens up interesting possibilities not only for coastline defense but also for offensive operations. The primary motivation for the autonomous oceanographic sampling network (AOSN) concept originally developed in [1] by Curtin and Bellingham is that current methods for sampling undersea processes are limited and these limitations are inhibiting our understanding of a wide range of ocean science problems. They argue that the measurement of temporal and spatial gradients in the ocean far exceeding current capabilities are needed to validate current models of oceanic processes. Some examples of the types of problems that would benefit from new methods are given as the mechanisms of frontal dynamics, surface dynamics, stratified turbulence, cross-shelf transport, deep convection, and sea ice mass balance. They argue that the current sampling done

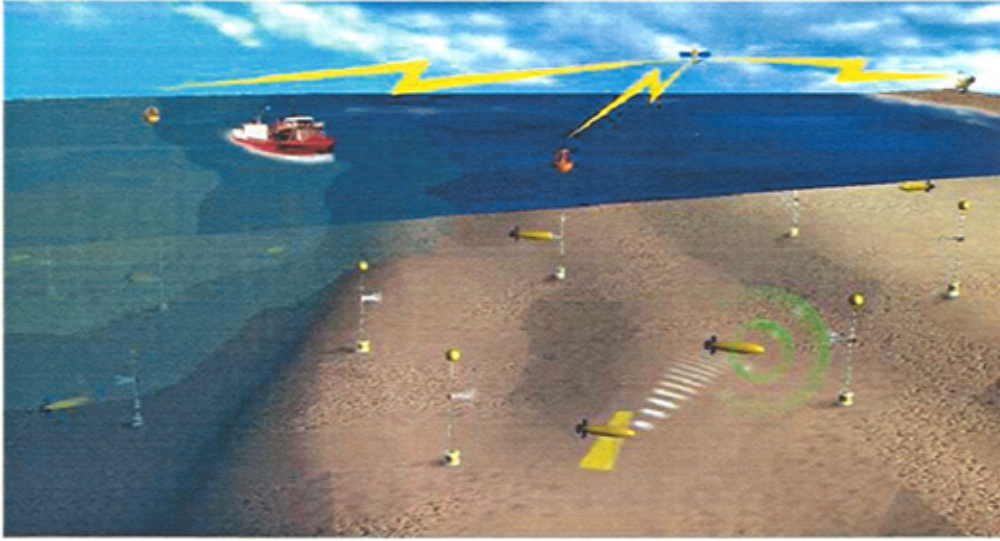


Figure 2-1: An Autonomous Oceanographic Sampling Network. Autonomous oceanographic sampling networks consist of a distributed system of fixed and mobile sensors networked together by communications nodes.

from ships, moorings, and floats only produces quasi-synoptic two-dimensional sections through evolving fields and that sparse sampling can introduce problems with temporal and spatial aliasing. The conclusion drawn by the authors is that a robust, distributed, autonomous system with low unit cost is necessary to affordably meet the requirements of sampling with long durations and high resolution. It is clear that such a system meets many of the requirements for marine sensor networks discussed in Chapter 1. In this chapter we will discuss the AOSN system concept and enabling technologies for AOSN implementation and review much of the work to date.

2.1 System Concept

An AOSN system consists of a distributed network of fixed and mobile sensors deployed in an area of interest. Nodes in this network are comprised of moorings or buoys equipped with acoustic beacons, acoustic communications modems, fixed sensors, power sources, and docking facilities for autonomous underwater vehicles (AUVs) and surface craft. A depiction of this concept is given in Fig. 2. The sensor platforms in this network can either be propeller-driven or buoyancy-driven (e.g gliders) underwater vehicles or surface craft. The AUVs traverse the network collecting data samples using a variety of sensors. Key observations can be transmitted in real-time to one of the network nodes while full data transfer occurs during docking with specialized buoys

designed to recharge the sensor platform batteries. Autonomous surface craft can be used as mobile navigation or communications nodes or they can be used to carry specialized sensors like sidescan sonars, for example. In oceanographic process sampling scenarios, acoustic transmission loss along the inter-nodal paths can be used to detect evolving fronts. A network controller then dispatches autonomous sensor platforms to the frontal region where they sample the evolving front and adaptively alter their motion in response to both the locally-sensed gradients and the global data set. In surveillance applications, autonomous sensor platforms can be programmed to patrol in distributed orbits waiting for a target to appear within sensor range. Upon target detection, these platforms can adaptively track the target and relay the track information to the communications network. Other sensor platforms with specialized classification sensors can then be vectored toward the target. One of the most important advantages of the sensor network approach to sampling is its ability to support cooperative sampling in which two or more sensor platforms maneuver themselves to increase their information about a process by exchanging state information. This allows processes to be sampled which would otherwise be difficult or impossible for a single sensor to sample.

The key advantages claimed for the AOSN concept are synoptic volume coverage, adaptive sampling, flexible control, energy management, and robustness to component failure. The authors of [1] assert that the practicality of the AOSN concept as described depends on the number of AUVs required, the type of AUV, and the performance of acoustic navigation and telemetry. Both the type and numbers of AUVs needed will heavily impact the cost of the network which, in the final analysis, is usually the deciding factor on the practicality of a system.

2.1.1 Number of Sensor Platforms Required.

Any synoptic survey system must be able to sample a process at a faster rate than significant changes occur in the structure if temporal aliasing is to be avoided. In order to avoid spatial aliasing, minimum spatial resolutions must also be maintained. These minimum sampling requirements of course drive the requirement for the number of vehicles needed to provide coverage of an area of ocean of a particular size. However, since autonomous sensor platforms have limited energy storage and the power required to operate a survey platform is heavily related to its speed, equations for the total energy required and the optimum number of vehicles required to survey an area of ocean with a particular resolution can be derived. These equations, derived in [1], show

that the optimum number of vehicles for oceanographic sampling varies with the area of the survey region and inversely with the required resolution and completion time as would be intuitively expected. There is also a weak increase in optimal vehicle number as vehicle size increases and hotel load decreases. As noted in the text, increasing the number of survey vehicles would allow the use of smaller (and presumably cheaper) vehicles because energy storage requirements are a major factor driving vehicle size.

Although the authors of [1] derive the optimum number of survey vehicles needed for a given survey in terms of minimum energy usage, it does not consider impact of using multiple vehicles on survey error or quantify the impact of adaptive sampling strategies. This situation was corrected in Bellingham and Willcox's 1996 paper [19]. In this paper, the impact on survey error of using multiple survey vehicles is computed for statistical ocean processes. The results show that while changing the physical parameters of the sensor platform (e.g. reducing hotel load) results in a mild increase in survey efficiency, the largest gains in error reduction come from using multiple vehicles and/or using adaptive sampling strategies. This is largely the result of an effective reduction in survey area per sensor platform. A paper describing the derivation of performance metrics for oceanographic surveys with AUVs is given in [20].

2.1.2 Sensor Platform Type

A network of many low-cost, light-weight underwater vehicles (see Fig. 2.1.2 and Fig. 2.1.2 [21]) is preferable to using a few, relatively expensive vehicles. Large vehicles are costly to build and operate while very small vehicles have difficulty integrating inexpensive sensors and computer hardware. In [1] the claim is made that moderate sized vehicles ($1 - 3\text{ m}$ long and 0.2 m to 0.8 m in diameter) are optimal due to their high maneuverability, high thrust to mass ratios, low cost, and ability to carry oceanographic sensors over ranges of at least hundreds of kilometers. Buoyancy driven vehicles (see Fig. 2.1.2) are also indicated for some applications, primarily observation of large-scale ocean processes. These vehicles are very power efficient but have low maneuverability and speed. Gliders could theoretically remain on station for periods of weeks to months but are power limited which tends to reduce the number of sensors that can be carried.

Since [1] was written, autonomous surface (see Fig. 6-4) have also emerged as a candidate sensor platform for marine sensor networks. These vehicles bring a number of advantages to the AOSN concept including the ability to carry a variety of sensors, but one of their greatest



Figure 2-2: An Odyssey-III AUV. This survey-class AUV is able to carry a wide variety of scientific sensors and is able to operate underwater unattended for long periods of time.



Figure 2-3: The Autonomous Benthic Explorer (ABE). ABE can survey the ocean floor at depths up to 5500 *m* carrying a variety of sensors such as cameras, sonar, and chemical sensors. ABE is operated by the Woods Hole Oceanographic Institution.

advantages is their ability to utilize the global positioning system. This allows autonomous surface craft to be able to act as mobile navigation and communications nodes within the network. This holds the promise for developing adaptively reconfigurable communications and navigation nodes. In particular, a reconfigurable communications network would be able to sense the local communications environment and be able to reconfigure their positions autonomously.

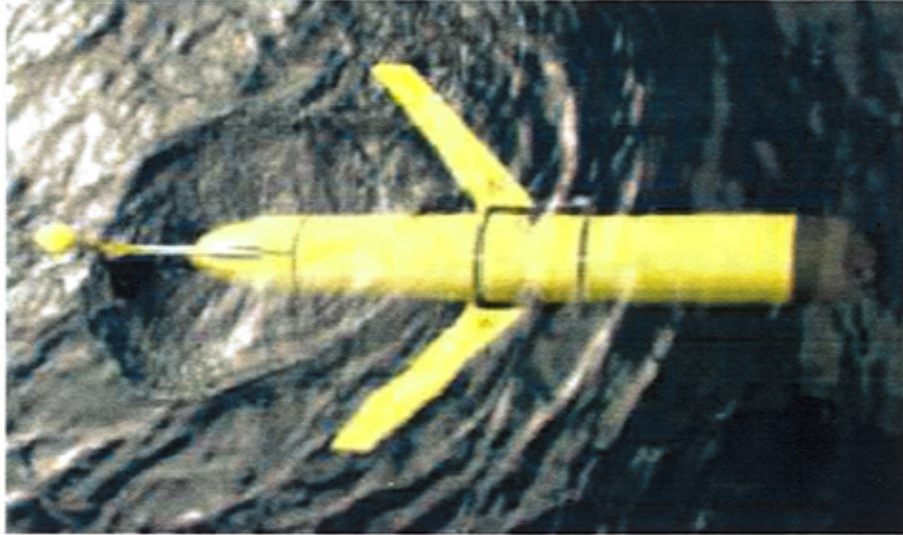


Figure 2-4: The Slocum glider. Gliders are very useful for sampling wide area oceanic phenomena. They are able to operate underwater for very long periods of time due to their use of thermal gradients in the ocean for their motion.

2.1.3 Acoustic Communications and Navigation.

In the AOSN concept, acoustic communications is necessary for communication between network nodes, between mobile sensor platforms and network nodes, and also possibly between remote fixed sensors and network nodes. The authors of [1] envision a sort of underwater "cellular telephone" system in which the network nodes perform much the same as current cellular base stations do. Data transfer rates for state-of-the-art acoustic modems (in 1994) are given as 10 *kbit/s* at 10 *km* and 3 *kbit/s* at 90 *km*. The authors claim that these data rates enable both the efficient telemetry of commands to the sensor platforms but also the transfer of large amounts of recorded data from the sensor platforms to the network nodes. A key issue here is energy efficiency, measured in *kbit/joule* per kilometer. While efficiencies in the range of

1 *kbit/joule/km* to 100 *kbit/joule/km* are manageable with sensor platforms and network nodes, careful trade-offs must be examined for remote sensors which may have to run unattended for months or years on battery power.

Substantial advances have been made in underwater acoustic communications since 1993. Some of these advances are described in [22] and [23]. Advances have been made both in terms of energy efficiency and in data rates. Advances in underwater navigation are less clear. The widespread use of GPS has made it easier to precisely locate surface buoys for long and short-baseline navigation systems but those networks are still plagued by the same poor performance at long ranges. Large strides have also been made in inertial navigation systems small enough to be used in AUVs (e.g. laser ring gyro systems) but there is still no way to get around the increasing error with distance problem. Theoretical advances have been made in stochastic mapping approaches such as concurrent mapping and localization [24] and other feature-based navigation techniques [25] but no practical system has yet been fielded. There is some promise in the use of autonomous surface craft as mobile navigation nodes in marine sensor networks, leveraging the surface craft's ability to position itself with GPS and its ability to communicate with underwater platforms via acoustic modem. Precise time synchronization between sensor platforms may allow precise inter-vehicle ranging simultaneous with acoustic communications.

2.1.4 Vehicle cost.

One of the most critical assertions made in [1] is that AUV costs will be driven lower by economies of scale as computer aided manufacturing and advanced materials combine with volume manufacturing. As noted earlier, the number of AUVs required is related to the area of ocean to be surveyed as well as the spatial sampling resolution and required survey time. In order to be able to achieve high-resolution surveys of any reasonable size, AUV costs must be as low as possible. The authors state that a reasonable cost objective is between \$10K and \$50K per vehicle.

Even though one of the critical assumptions was that low-cost manufacturing techniques and volume production would lower the cost of AUVs, this has not been the case. If anything, AUV survey vehicles are even more expensive than in 1993. For the most part this seems due to the fact that most AUVs are still hand-made research vehicles. The authors state that the optimum cost for AUVs is in the neighborhood of \$10K to \$20K while current prices can run 10 to 50 times that cost.

2.2 Enabling Technologies.

In [1], the authors assert that a number of "high leverage" technologies have the capability to substantially advance AOSN capabilities. One of these is intelligent software control of AUVs. A full implementation of the AOSN concept requires that AUVs be able to implement adaptive sampling including the ability to coordinate/cooperate with multiple vehicles in order to optimize the sampling strategies with regard to variables such as time, power consumption and perhaps, most importantly, survey error. The optimization of AUV control strategies is highly dependent on models of the process to be observed. This issue, of course, is the topic of this thesis.

Energy storage and power management of the AUVs is given as another key enabling technology. The survey range of an AUV is determined by its energy storage capacity and its power usage. Obviously, an increase in the energy density of energy storage systems has the capacity to extend AUV range and/or reduce vehicle size. An increase in propulsion efficiency or a decrease in the required hotel load would also have the same effect.

While a number of theoretical attempts at control strategies for multiple AUV systems and adaptive sampling have been made, no great leaps in progress have been made in this area. One of the more interesting attempts is described in [14] in which a virtual bodies and artificial potentials strategy was used to control adaptive sampling among multiple vehicles in an AOSN experiment. This is an example of an adaptive control method known as the world-model or "sense-plan-act" model discussed in Chapter 3.

Progress in energy storage has also been amazingly slow. While some attempts have been made to use fuel-cell technology to power AUVs, most AUVs continue to use conventional battery technology. Current state-of-the-art batteries for AUVs use the same lithium-polymer battery technology as found in laptop computers.

2.3 Historical Developments

2.3.1 The AOSN Project

The AOSN project was begun with the long-term goal to create and demonstrate a reactive survey system, capable of long-term unattended deployments in harsh environments. Sponsored by the Office of Naval Research, AOSN was a collaboration between the Massachusetts Institute of Technology, the Woods Hole Oceanographic Institution, the University of Washington, and

Northeastern University. The main thrust of the project was to develop both AUV technology and AOSN network infrastructure technology including acoustic communications [23] and vehicle docking technology [26] as well as the investigation of operational techniques. There were two major AOSN experiments, the Haro Strait experiment and the Labrador Sea experiment.

Haro Strait

The goal of the Haro Strait AOSN experiment (formally the Ocean Frontal Dynamics Primer Initiative) [27] [18] was to use advances in ocean modeling and AUV technology to study tidal mixing processes in the Haro Strait which lies between the San Juan Islands and Vancouver Island off of British Columbia. The main technical goals of the experiment were adaptive sampling, coordinated platform operations, and communications. More than 60 AUV runs were accomplished in an attempt to localize and characterize the strong tidal fronts as they moved through the strait. The experiment was jointly conducted between MIT, the Woods Hole Oceanographic Institution, the Institute for Oceanographic Science in British Columbia, Harvard University, and the University of Victoria.

Labrador Sea

The goal of the Labrador Sea AOSN deployment was to observe convection plumes in a responsive, repetitive manner by providing a long-term unattended deployment capability [18]. This was the first attempt at long-term deployment of AUVs and gliders in an AOSN and used moorings and buoys in addition to the sensor platforms for RF communications and unattended recharging of vehicle batteries. Communications moorings were used to relay commands to the sensor platforms as well as receive science data from the platforms when the platforms were docked.

2.3.2 The AOSN-II Project

AOSN-II is an ONR-sponsored, multi-institutional, collaborative research program with the central objective to quantify the gain in predictive skill for principal circulation trajectories, transport at critical points and near-shore bioluminescence potential in Monterey Bay as a function of model-guided, remote adaptive sampling using a network of AUVs. A partial AOSN implementation, AOSN-II will use a fleet of Slocum gliders [28] to adaptively sample the oceanic processes in Monterey Bay. The key research goals of the experiment are a focus on adaptive sampling and

the opportunity to use the glider network as a reconfigurable, mobile sensor array. A major test of the AOSN-II concept using Slocum gliders was performed in 2003 [14].

2.3.3 The NEPTUNE Project

The NEPTUNE project [29] [22] is a perfect example of the implementation of a prototypical AOSN. The goal of the NEPTUNE project is to establish a regional ocean observatory in the Pacific Ocean off the northeast coast of the United States. A 3000 *km* network of fiber optic cables will encircle and cross the Juan de Fuca tectonic plate, an area of nearly 500 *km* by 1000 *km* in size. Approximately 25 experimental sites will be established at nodes along the cable. Each experimental site will be instrumented to sample physical, chemical, and biological parameters using a combination of fixed and mobile sensors interconnected by an acoustic network. The AUVs will reside at depth at each node where they will recharge and respond to real-time events such as underwater volcanic eruptions. Real-time data and command and control capabilities will be available via the Internet. Costing an estimated \$250M to develop and operate over the first five years, NEPTUNE will be partially operational by 2007.

2.4 Relevance of AOSN to this Thesis

The focus of this thesis is a fundamental problem in sensor design and operation for real-time observation of ocean processes with mobile sensor platforms. This problem relates to the observation of processes which intentionally or unintentionally are difficult to view using a single sensor platform and which require the use of multiple sensor platforms, each of which can sense the process from a different "viewpoint." Necessarily, this requires close coordination between the sensor platforms and the ability of the platforms to adapt their sampling (sensing) strategy to information received and processed in real-time. The goal is to be able to use numbers of small, inexpensive sensor platforms for this task. One of the applications of great interest examined in this thesis is the tracking of acoustic targets using passive (bearings-only) sensory data. Chapters 6 and 7 give experimental examples of adaptive, networked target tracking with both single and multiple sensor platforms.

One of the features of the AOSN paradigm that is ideally suited for the target search and identification problem is its sensor-adaptive nature, i.e. the pre-planned movement of the sensors (the AUVs in this case) can change according to the nature of their sensor readings. This allows

for the possibility of adaptively cooperating sensors and the optimization of sensor movement. Current oceanographic sensor missions with mobile platforms typically involve a single vehicle sampling various oceanographic processes (temperature, salinity, water chemistry, bioluminescence, sonar returns, etc.) within a designated volume of ocean. Collected data is stored on the vehicle for later post-processing. While this technique is extremely valuable for collecting data on certain types of processes, it also has severe drawbacks for observing other types of processes. Specifically, this technique fails to address situations where it is desirable to observe processes in the ocean that simultaneously evolve in time and space and for which real-time action is required based upon the evolution of the process. This would include the detection and classification of stationary and moving underwater targets using moving sonar sensor platforms.

One of the primary projects which is sponsoring my work has been the Generic Ocean Array Technology program [30]. The GOATS program, a subprogram of AOSN, has as its goal the detection and classification of both proud and buried targets in very shallow water. In line with the AOSN paradigm, this is to be done by enabling a fleet of adaptively cooperating AUVs (communicating via acoustic modems) equipped with acoustic receiving arrays to process multi-static sonar return data from targets insonified with a low-frequency acoustic source mounted on one of the vehicles. Because the scattered field is not spatially isotropic in general it is thought that, by analyzing the spatial and temporal nature of the multi-static returns, we can thereby simultaneously detect and classify the target in real-time. We believe that our proposed method has significant advantages over traditional sonar methods for finding and identifying proud and buried targets. By using the AOSN paradigm, we enable real-time transmission of target information to other nodes in the network. This could enable the reallocation of resources within the network in real-time to deal with the threat.

Another program sponsoring my proposed thesis work entitled Persistent Littoral Undersea Surveillance (PLUS) follows the AOSN paradigm. PLUS is a research program that will explore naval systems for clandestine undersea surveillance to provide the location of submarines in far-forward and/or contested waters. PLUS emphasizes mobile and/or fixed multiple sensing nodes, networked to provide an adaptive and/or relocatable sensing grid. One concept being strongly examined is that of a network of AUVs with passive sonar. AUVs would be capable of passive acoustic detection and have the capability of adaptively tracking a moving target and interacting with others AUVs in the network either to increase the accuracy of the target track estimates or to hand off tracks to a distant portion of the network. The AOSN concept as described in this

paper and subsequently developed, is directly applicable to this work.

Chapter 3

Adaptive Sensor Platform Control

Although the state-of-the-art in marine sensor platform hardware technology has made impressive gains in recent years, the paradigms and software necessary to make truly adaptive sampling a reality have lagged behind. Sensor platforms have been relegated to the role of pre-programmed data collector, following a deterministic survey path through the ocean with data offloaded for analysis after mission completion. Fig. 1-2 shows the path of a number of actual survey missions completed by an AUV off the coast of Italy during an recent experiment. Although this type of survey mission is invaluable for many scientists, the marine sensor platforms currently lack the ability to respond to external events generated by on-board processing of the collected data. In order to adaptively control a sensor platform in real-time, the platform control system must be able to make decisions about its future course based on information that is streaming in from its sensors. Often times, the preferred course must be determined even though the platform may be trying to simultaneously satisfy multiple, conflicting goals. These decisions must be made quickly enough to satisfy the physical control laws governing the vehicle's motion. A number of different approaches to this fundamental robotics problem have been advocated including the use of comprehensive world models as well as the pre-computing of the actions for all possible vehicle states. These control models are what is referred to in [3] as the “sense-plan-act” (SPA) or world-model approach. As noted earlier in Chapter 1, due to the very large state spaces in any realistic world model, these methods can lead to an intractable computing problem especially when a vehicle must interact in a timely fashion with a changing environment. To address this problem, we turn our attention to the use of behavior-based methods which are thought to be well suited for reactive control in complex environments. In this chapter, we will compare and contrast

the behavior-based and world-model approaches, describe a number of behavior-based methods in detail, provide an innovative solution to the preferred action selection problem inherent in behavior-based methods, and describe a distributed sensor platform control architecture well suited to supporting adaptive, cooperative control in marine sensor networks.

3.1 The World Model-Based Approach

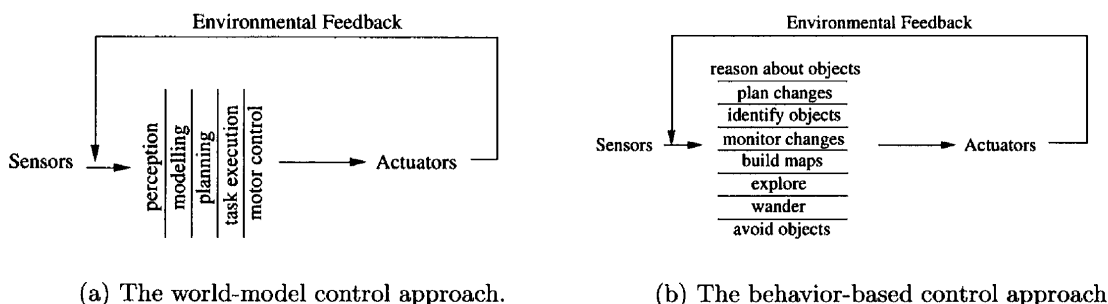


Figure 3-1: This figure compares the traditional world-model approach to robot control (a) which breaks the control into sequential functional modules with the behavior-based approach (b) in which task-achieving behaviors act in parallel [2].

Fig. 3-1(a) shows the basic SPA approach in which, during each control cycle, the sensors are sampled, a planning model is used to map the sensor states to the control output and decide the task execution, and the actuators change the vehicle state. Feedback is implicitly provided on the next control cycle through the sensors. One simple way to do this is to preplan the control output for all combinations of sensor input. The combinatorial explosion of the state space for even mildly complex environments make this method prohibitive, however. Most implementations of the SPA control methodology use some form of a model of the world which maps the sensor states to the actuator states. This can actually work quite well for very simple situations. However, in complex situations where many simultaneous constraints must be met (e.g. obstacle avoidance, navigation, adaptive sampling, cooperation, etc.), the size of the state space can be prohibitive. A very recent example of a control application in this genre is given in [31] where the authors steer an ultralight aircraft using visual sensors with a fly-inspired control model.

3.2 Behavior-Based Approaches

In an attempt to deal with the problems encountered by the traditional AI-type robot control architectures in use in the early 1980s (in particular the world-model approach), a new control paradigm came into prominence. Termed "behavior-based", this new approach sought to decompose the control architecture of a robot into discrete modules termed "behaviors", each operating in parallel and each able to provide a preferred control output during each control cycle (see Fig. 3-1(b)). One of the early advocates of this approach was Rodney Brooks, whose subsumption architecture [2] has been since incorporated into many robotic systems. According to Brooks, behavior-based approaches are able to deal with one of the biggest issues in robotics, the issue of how to control a robot in the face of multiple, conflicting goals especially when the control system must deal with the environment in real-time. One of the defining characteristics of behavior-based control methodologies is the tight coupling between the environment (as sensed by the robot in real-time) and the vehicle behavior. A reliance on long-range planning is not a characteristic of this method. The common issue faced by all behavior-based control models, however, is the issue of how to arbitrate between the conflicting opinions of different behaviors. In this section we will examine a number of behavior-based approaches with increasingly complex arbitration schemes.

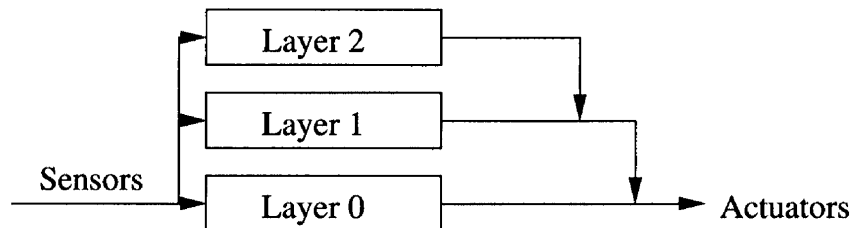


Figure 3-2: This figure depicts the layered control model (subsumption architecture) [2]. Each layer is a task-oriented behavior such as those shown in Fig. 3-1(b). Sensors inputs are processed in parallel. At every control cycle each behavior can output its preferred actuator control parameters. The output of higher-level (lower priority) behaviors can be suppressed if a lower-level behavior provides an output of the same control parameter.

3.2.1 Subsumption Architecture

In Brooks' subsumption architecture for robotic control, a network of augmented finite-state machines (AFSMs) provides control input to the low-level control system of the robot. Each

of the AFSMs, known as behaviors, competes for the right to provide its control input to the control system but only the behavior with the highest priority is allowed to do so. Each of the AFSMs can contain its own set of internal timers, sensor inputs, and memory storage. Each of the AFSMs can also be wired to inhibit or enable the action of another AFSM. The attraction of this approach is that a bottom-up control system can be defined whereby the interaction of many simple behaviors running in parallel can result in seemingly complex behavior. In Brooks' subsumption architecture, each behavior is prioritized. During each control cycle, each behavior is able to propose a subset of vehicle control parameters (e.g. course, speed) with the highest priority behavior being able to send its parameters to the vehicle hardware for action (see Fig. 3-2). In this way Brooks felt, the control system could be responsive to higher level goals while still being able to service low-level (but potentially high priority) goals like vehicle safety.

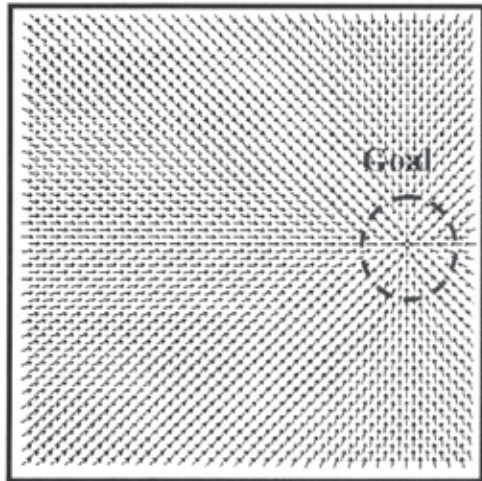
A common criticism of this "winner takes all" arbitration scheme is that it leaves no room for inter-behavior compromise. Given multiple priorities, it seems likely that the objectives of one or more behaviors can be met simultaneously, if not with 100% efficiency, then with some measure of efficiency which contributes to the overall mission objective. For example, in the single bearing target tracking example described in Chapter 6, the sensor platform must manage several competing behaviors including closing range with the target while trying to keep the proper angle between the target and the simulated acoustic line array.

3.2.2 State-Configured Layered Control

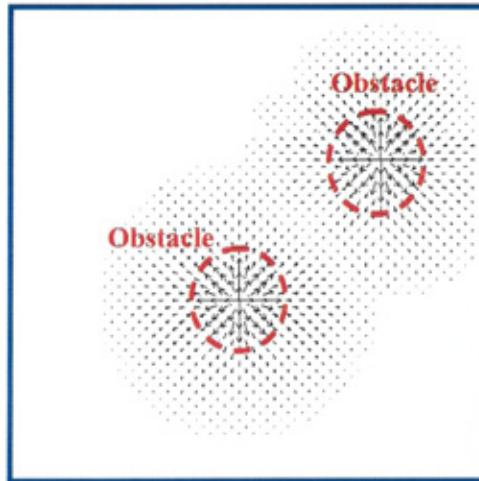
A form of Brooks' subsumption architecture, termed state-configured layered control [32], has been incorporated into the control system of the Odyssey autonomous underwater vehicle at MIT. In state-configured layered control, only certain behaviors are actually active at any given time depending on the state of the vehicle state table (to reduce computational costs). This can simply be seen as being Brooks' original scheme with the vehicle states being inhibitory or enabling inputs to the behaviors. To date, only an extremely simple version of state-configured layered control has been implemented on the AUVs at MIT. In particular, the behaviors do not have the ability to enable or inhibit other behaviors directly (except possibly indirectly through the use of the vehicle state table) and do not have access to internal timer constructs. Additionally, only the most simple behaviors have been developed such as moving the AUV in straight lines or circles and possible repeated sequential combinations of straight lines and circles or polygons.

3.2.3 Motor Schemas

One behavior-based control methodology which uses an arbitration scheme designed to compromise between competing behaviors is that of motor schema [33]. In the motor schema control scheme, each behavior (motor schema) uses a potential field representation to represent desired paths and obstacles. At each control cycle, each motor schema outputs a desired vehicle velocity vector which is then vector-summed by the arbiter. This vector is then used to provide steering and speed commands to the vehicle hardware. As obstacles are discovered by the vehicle sensors, new motor schemas can be instantiated and used in subsequent control cycles. Fig. 3-3(a) and Fig. 3-3(b) shows representations of two motor schema vector fields, the go-to-goal schema and the obstacle schema. Fig. 3-4 shows the vector sum of these two schema. Placed anywhere in the field, a robot using motor schema control will follow the vector field to the goal. This scheme differs from the subsumption architecture in two important ways. First, a compromise between all active behaviors is reached by calculating the vector sum of their respective outputs. There is no layering or priority in this scheme. Second, new schemas can be instantiated dynamically as features in the environment are encountered.



(a) The Go-to-goal motor schema.



(b) The obstacle motor schema.

Figure 3-3: This figure shows the vector fields associated with two motor schemas, the go-to-goal motor schema (a) and the obstacle motor schema (b).

Balch and Arkin describe five motor schema behaviors which they used to control the robots in their experiment. These included a schema for avoiding obstacles, one for maintaining the proper

distance from other vehicles, one for moving toward a navigational goal, one for maintaining formation, and one for introducing noise into the output vector to break deadlocks. There are

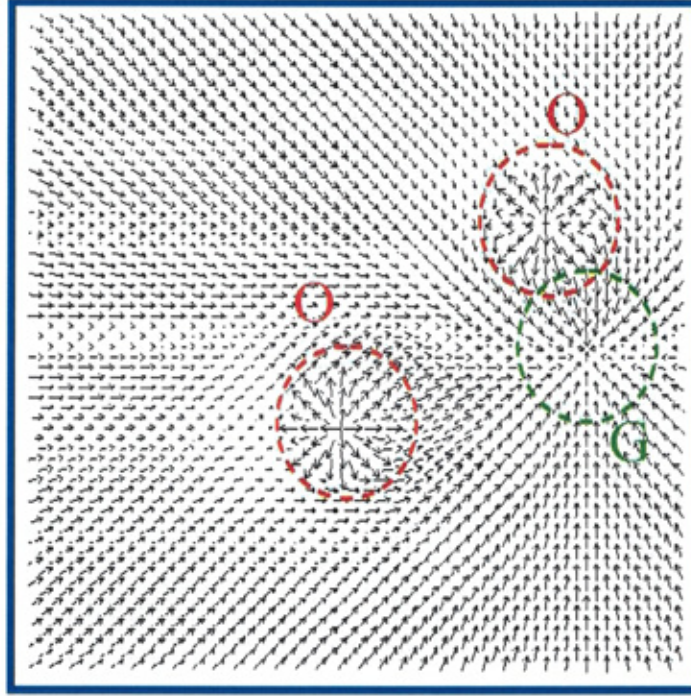


Figure 3-4: This figure shows the vector sum of the go-to-goal and obstacle motor schemas. Placed anywhere in the field, a robot using motor schema control will follow the vector field to the goal.

several negative aspects to the motor schema approach. First, there is no guarantee that the simple vector sum of two schemas will result in stable platform motion. In fact, the simple sum of velocity is subject to the same problems common to all potential field control approaches such as the unintentional introduction of local minima, attractors and limit cycles in which the robot can become trapped. Second, the average behavior is not necessarily the optimum behavior. The motor schema approach allows no consideration or ranking of alternative actions. Other work investigating the use of potential functions for robotic control can be found in [34][35][36][37].

3.2.4 The Multiple Objective Function Approach

In [3], Benjamin takes the view that simple arbitration schemes that suppress all but the most important behavior or average or convolve an overall action from multiple behaviors are methods that lead to unacceptable shortcomings in overall vehicle behavior, a view also shared by

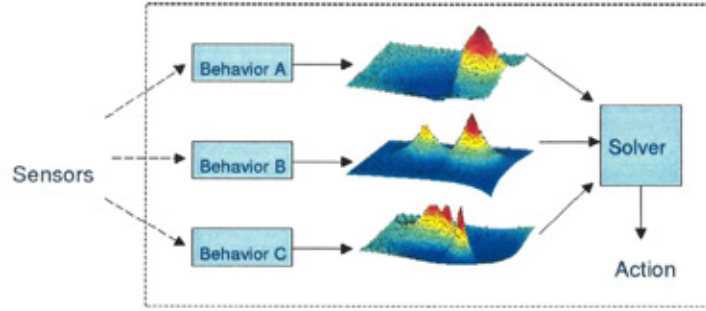


Figure 3-5: This figure depicts the multiple objective function approach to action selection in a behavior-based control system. Each behavior outputs a *function* over the vehicle control parameters such as heading, course and speed. The preferred action is then found by multi-function optimization over all the objective functions.

others [38] [39] [40]. Benjamin argues that each behavior should produce alternative actions in addition to the preferred action in order to allow multiple, interacting behaviors to effectively compromise. This leads to a behavior-based control method in which each behavior produces an objective function whose domain is the vehicle control parameters (e.g. course, speed, and depth) during each control cycle. These objective functions are then subjected to a multi-function optimization to produce the preferred control parameters (see Fig. 3-5). The control output is then, in some sense, a collective decision based on all of the competing goals for the vehicle. The major drawback to this approach is that the multi-function optimization is computationally intensive, especially for small sensor platforms with limited computing resources. In [3], Benjamin describes a computationally efficient method for solving the multi-function optimization. In this method, called the Interval Programming (IvP) method, the objectives functions are described in a piecewise linear manner. The multiple objective function approach using the IvP model was used to execute the adaptive control experiments described in Chapters 6 and 7 and is described further in Section 4.1.1.

3.3 Cooperative Control

In my opinion, in order to find a workable solution to the seemingly complex problem of multi-robot cooperation, one needs to have a view of what the nature of an autonomous robot is. In my own study of this issue, I believe that Brooks said it best in his Pseudo-Definition 3 [41] where he states: "An autonomous (artificial) creature is one that is able to maintain a

long term dynamic with its environment without intervention. Once an autonomous artificial creature is switched on, it does what it is in its nature to do.” It therefore seems reasonable that cooperative behavior could be built using the behavior-based control approach of the single robot but with certain of its behaviors being cooperative, ”cooperative behaviors” if you will. In our current context, these cooperative behaviors would require state information about the other robots or environmental state information collected by the other robots, either actively or passively gained. As a practical matter of course, cooperative robot behavior also requires certain technological achievements including the ability of the robots to communicate or at least to be able to sense the state of the other robots. One issue that cannot be overlooked is that of the robustness of the behaviors to the reality of the world. In the real world, communication may be slow or intermittent, sensor readings are always noisy, and unexpected events may occur. Any cooperative robotic behaviors must be robust with respect to these issues. In [42], Parker breaks robot cooperation into two distinct genres termed “swarm cooperation” and “intentional cooperation”. Parker draws parallels between these two types of robotic cooperation and forms of societies in the animal kingdom.

3.3.1 Swarm Cooperation

Swarm cooperation is characterized by large groups of heterogeneous robots that perform repetitive tasks over large areas. Such examples could include mapping, mining, sorting, exploring, surveying, and construction. In this type of cooperation, robots do not explicitly cooperate to complete a task. Typically they can sense the environment and the positions of the other robots in the group. Each robot bases its motion on its own local control law and the global behavior of the group is emergent from the interactions of each robot acting according to that control law. One of the most interesting cooperative group behaviors is formation control which is covered in detail in Section 3.3.3. A number of researchers have considered robotic systems using this type of cooperative behavior including Brooks et al [43] who considered control strategies for soil-moving robots on a simulated lunar base, Nguyen et al. [44] with their Basic UXO Gathering System (BUGS), and Konolige et al [45] with their multi-agent system for mapping and exploration. The only multi-robot system implemented in a marine sensor environment that I am aware of was the multi-glider system used in the AOSN-II experiment discussed in Section 2.3.2 and detailed in [14].

3.3.2 Intentional Cooperation

Intentional cooperation among multiple robots is usually characterized by small groups of possibly heterogeneous robots which actively cooperate to accomplish a task or mission. Parker notes that, traditionally, research into this type of robotic cooperation breaks down into two camps, those using traditional “sense-plan-act” models of adaptive control and task allocation models and those from the distributed AI community where task allocation is the driving force behind the cooperative architectures.

In this work we implement a form of intentional cooperation different from that generally found in the literature. In this approach there is no mechanism for task allocation because each robot knows how to do its task based on the behaviors it is programmed with. Cooperation is implemented by allowing the robots to share state data about themselves and the environment. I compare this approach to the cooperation used by some insects in which “searcher” insects are responsible for searching the environment for food. When food is found, the searcher insects will communicate the location to other, “gatherer” insects which then gather the food and bring it into the communal storage. This simplifies task allocation because each robot is responsible for only one task, the one it is programmed for and its behavior is directly linked to the state of the environment and the other robots. The tracking example detailed in Chapter 7 uses this form of cooperation in which the sensor platforms share sensor data. Based on the sensor data, the platforms position themselves appropriately to efficiently track the target. One criticism of this approach might be that it requires some knowledge of a mapping between the environment state and the behavior of the robot. This is true but the use of a behavior-based approach makes this a tractable method.

3.3.3 Formation/Flocking Behaviors

Formation keeping is an important aspect of navigation for many types of military maneuvers with troops and vehicles as well as a number of search and rescue, agricultural, and security patrol applications. As autonomous robots gain a place as members of these formations, an automated method for keeping each vehicle in formation is necessary. Formation keeping is especially important where sensor assets are limited for it allows an efficient partition of the search space among the members of a group. In [46], Balch describes a behavior-based approach to robotic control that could potentially allow a robotic team to reach navigational goals, avoid

hazards, and simultaneously remain in formation with the type of formation used being dependent on the task assigned to the team.

As the article notes, formation control is well known in the animal kingdom where it is used to gain many of the same benefits as those we seek for our robotic teams, including the ability to maximize sensor coverage and provide for group protection [47][48]. These aggregate flocking and schooling behaviors are a combination of both a desire to remain within the group but yet remain some distance from the other members of the group. In nature, the flocking behavior is dependent only on each individual having local knowledge of the environment and the positions of its nearest neighbors. The parameters which control the desired group size and individual distances no doubt depend on the state of the group (e.g. feeding, fleeing, traveling, etc.) In this article, a number of pre-specified geometrical formations are considered for which each individual robot's position relative to the group is specified and maintained. Each of these formations is examined for its appropriateness in particular task environments.

Four formations were examined in the article, line, column, diamond, and wedge. At every step in the control cycle, each vehicle computes its proper position in the formation. The maintain-formation schema then generates a movement vector toward the desired location. Each robot determines its proper position in the formation based on its knowledge of the location of the other robots. Each robot determines the position of the other robots by either direct perception, via dead reckoning or via transmitted GPS coordinates. Three different methods for each robot to determine its proper position in the formation identified:

- **Unit Center Referenced:** Each robot computes the centroid of the formation and determines its preferred position relative to that "unit center".
- **Leader Referenced:** Each robot computes its preferred position relative to the position of the lead robot. The lead robot does not maintain formation.
- **Neighbor Referenced:** Each robot maintains its position relative to one other robot in the formation.

Other work on formation control can be found in [49][50].

In a swarm, each vehicle navigates according to its own perception of its local environment (including its perception of other robots in the swarm) and its own local control law. In [14], a swarm of gliders is used to adaptively sample various processes in Monterey Bay. In this case, the

glider swarm maintains formation using a method called Virtual Bodies and Artificial Potentials (VBAP) which is a complex method of implementing adaptive leader-referenced formation control in a provably stable manner. The type of formation keeping used in the Monterey experiment used a type of leader-referenced formation using what is called a “virtual leader”. A virtual leader is simply a point in space acting as a leader for the formation rather than a physical sensor platform.

As discussed earlier, formation control is an example of “swarm” cooperation in multiple robot systems. One characteristic of this type of cooperative system is that all of the robots in the system are homogeneous. Another characteristic is that there is no explicit cooperation between the vehicles. While the first characteristic, that of vehicle homogeneity, has little impact on my proposed thesis work, an approach with no explicit cooperation between vehicles is not a viable option. For the moving target tracking problem, the solution for the target track can be computed using bearing measurements taken from multiple sensor platforms. In order to compute this solution, however, information must be explicitly exchanged between vehicles. For the bi-static and/or multi-static classification of stationary targets, it is possible that a swarm type approach to cooperation might be feasible if each vehicle knew the positions of all other vehicles in the swarm and there were a large number of vehicles. In this case, each member of the swarm would compute its own estimate of target locations and classifications. However, one of the goals of my research is to identify how to use simultaneous sensor measurements from distributed sensor platforms in order to optimize the vehicle trajectories in order to improve detection and/or classification performance. In order to accomplish this, the vehicles would need to explicitly exchange target information and perhaps also path planning information.

Chapter 4

The MOOS-IvP Autonomy Architecture

4.1 A Distributed Control Architecture for Marine Sensor Platforms

In this section we discuss the general autonomy architecture that has been developed for use on mobile marine sensor platforms and how the particular components that reflect the contribution of this work fit into that architecture. This architecture was used to execute the autonomous tracking experiments described in Chapters 6 and 7.

4.1.1 Behavior-Based Control with Interval Programming

By using multi-objective optimization in action selection, behaviors produce an *objective function* rather than a single preferred action ([51, 52, 40]). The Interval Programming (IvP) model specifies both a scheme for representing functions of unlimited form as well as a set of algorithms for finding the globally optimal solution. All functions are piecewise linearly defined, thus they are typically an *approximation* of a behavior's true underlying utility function. Search is over the weighted sum of individual functions and uses branch and bound to search through the combination space of pieces rather than the decision space of actions. The only error introduced is in the discrepancy between a behavior's true underlying utility function and the piecewise approximation produced to the solver. This error is preferable compared with restricting the

function form of behavior output to say linear or quadratic functions. Furthermore, the search is much faster than brute force evaluation of the decision space, as done in [40]. The decision regarding function approximation accuracy is a local decision to the behavior designer, who typically has insight into what is sufficient. The solver guarantees a globally optimal solution and this work validates that such search is feasible in a vehicle control loop of 4Hz on a 600MHz computer.

To enhance search speed, the initial decision provided to the branch and bound algorithm is the output of the previous cycle, since typically the optimal prior action remains an excellent candidate in the present, until something changes in the world. Indeed when something *does* change dramatically in the world, such as hitting a way-point, the solve time has been observed to be up to 50% longer, but still comfortably under practical constraints.

Although the use of objective functions is designed to coordinate multiple simultaneously active behaviors, helm behaviors can be easily conditioned on variable-value pairs in the MOOS database to run at the exclusion of other behaviors. Likewise, behaviors can produce variable-value pairs upon reaching a conclusion or milestone of significance to the behavior. In this way, a set of behaviors could be run in a plan-like sequence, or run in a layered relationship as originally described in [53].

4.1.2 The MOOS-IvP Autonomy Architecture

This work uses the MOOS-IvP architecture for autonomous control. MOOS-IvP is composed of the Mission Oriented Operating Suite (MOOS), a open source software project for coordinating software processes running on an autonomous platform, typically under GNU/Linux. MOOS-IvP also contains the IvP Helm, a behavior-based helm that runs as a single MOOS process and uses multi-objective optimization with the Interval Programming (IvP) model for behavior coordination, [51, 54]. See [55] and [56] for other examples of MOOS-IvP on autonomous marine vehicles.

A MOOS community contains processes that communicate through a database process called the MOOSDB, as shown in Fig. 4-1(a). MOOS ensures a process executes its “Iterate” method at a specified frequency and handles new mail on each iteration in a publish and subscribe manner. The IvP Helm runs as the MOOS process pHelmIvP (Fig. 4-1(b)). Each iteration of the helm contains the following steps: (1) mail is read from the MOOSDB, (2) information is updated

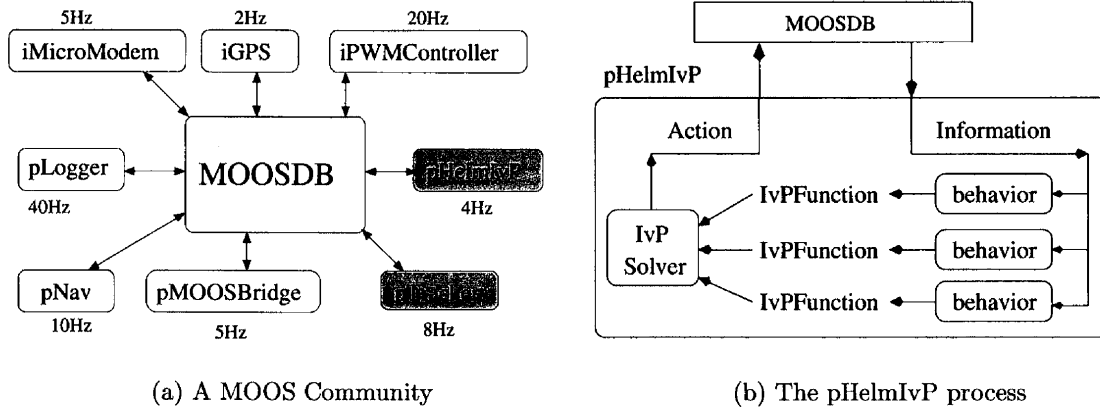


Figure 4-1: The IvP Helm runs as a process called pHelmIvP in a MOOS community. MOOS may be composed of processes for data logging (pLogger), data fusion (pNav), actuation (iPWMController), sensing (iGPS), communication (pMOOSBridge, iMicroModem), and much more. They can all be run at different frequencies as shown.

for consumption by behaviors, (3) behaviors produce an objective function if applicable, (4) the objective functions are resolved to produce a single action, and (5) the action is posted to the MOOSDB for consumption by low-level control MOOS processes. The behaviors responsible for control in the tracking and classification vehicles are discussed in Section 4.2.

4.2 Sensor Platform Behaviors

This section describes the behaviors used in the IvP Helm to execute the experimental missions described in Chapters 6 and 7. Three different types of behaviors are used on the sensor platform to produce a complete mission:

1. Safety behaviors - These behaviors are run in parallel with the other behaviors in order to maintain a safe operating environment for the sensor platform. Safety behaviors can include behaviors for avoiding obstacles, staying in a safe operating zone, mission timeouts, and behaviors for emergency events which require immediate attention.
2. Non-adaptive behaviors - These behaviors are usually sequenced with the adaptive behaviors to provide a robust capability for the sensor platform to operate during times when the adaptive behaviors are not active. These can include transiting to and from the operations areas and patrolling the operations area.

3. Adaptive behaviors - These behaviors activate when a target is detected and adaptively control the platform motion during the tracking phase using target track information from the intelligent sensor.

4.2.1 The OpRegion Behavior

The OpRegion behavior is a safety behavior responsible for insuring the sensor platform remains in a predetermined safe operating area. The behavior is configured with a single polygon and will result in an all-stop signal ($\text{THRUST}=0$) to the low level controllers if the vehicle leaves the operation area. The OpRegion behavior does not produce an objective function. It just informs the helm that there is a critical condition that should trump all behaviors and produce the action of all-stop. In this sense, the relationship of behavior is not unlike Brooks' layered approach where a critically important module can trump all others without seeking compromise.

4.2.2 The Waypoint Behavior

The Waypoint behavior is responsible for moving the sensor platform from one point to another along the shortest path. The behavior is configured with a list of waypoints and produces objective functions that favorably rank actions with smaller detour distances along the shortest path to the next waypoint. This behavior is used by the target vehicle in the experiments to form a constant velocity motion, for example, and multiple waypoints can be sequenced together to form platform motion along arbitrary polygons. Every vehicle is typically configured with an instance of this behavior having a single waypoint just off the starting area, conditioned on both a "mission=complete" or "return=true" condition for returning all vehicles upon mission completion or recalling them mid-mission should the need occur. The objective function for this behavior is three-dimensional over course, speed, and time.

4.2.3 The Orbit Behavior

The Orbit behavior (see Fig. 4-2) is responsible for providing a patrol capability in which the vehicle will orbit a fixed point. Given an orbit center, the behavior dynamically determines a list of waypoints to form the orbit. Parameters to this behavior allow the choice of clockwise/counter-clockwise orbits as well as the number of waypoints in the orbit path and the vehicle speed. The objective functions for this behavior are identical to the standard waypoint objective functions

described in Section 4.2.2. The Orbit behavior can be conditioned to be active when no target is being tracked and to deactivate itself upon target detection. The orbit behavior always has a weighting of 1.0.

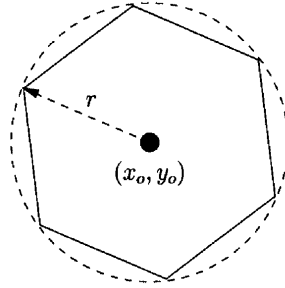
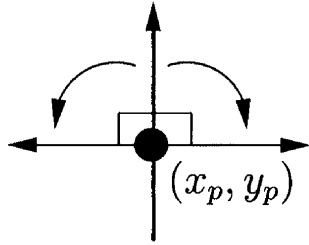


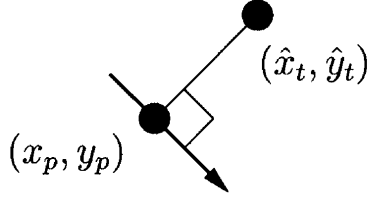
Figure 4-2: The Orbit behavior. The Orbit behavior is responsible for keeping the sensor platform in orbit around a fixed point at a fixed radius.

4.2.4 The ArrayTurn Behavior

The ArrayTurn behavior (see Fig. 4-3(a)) is responsible for providing a vehicle turning motion such that sensor platforms with acoustic line arrays can determine which side of the array the target is on. This behavior requires tight integration with the acoustic sensor which signals when the left/right ambiguity has been cleared. The objective function for this behavior is one-dimensional over course and bimodal, with the modes centered around the two possible course choices which are ninety degrees from the vehicle's course when the behavior is activated (he course fix). The mode that is centered at the course closest to the vehicle's current course is weighted in order to prevent frequent oscillation between the two modes. Fig. 4-4 shows a plot of the objective function for this behavior for a course fix of zero degrees and a current course of five degrees. Note how the mode closest to the current course is weighted slightly higher. Fig. 4-4 shows a plot of the objective function for the ArrayTurn behavior for a course fix of zero degrees and a current course of fifty degrees. Note how the mode closest to the current course has increased its weight relative to the other mode for the situation shown in Fig 4-4. The ArrayTurn behavior has a constant weighting of 1.0.



(a) The ArrayTurn behavior



(b) The ArrayAngle behavior

Figure 4-3: The ArrayTurn and ArrayAngle Behaviors are both one-dimensional over over course and bimodal. The ArrayTurn behavior is responsible for turning the vehicle up on a target detection in order to clear the left/right ambiguity on the line array. The ArrayAngle behavior is responsible for keeping the line array as close as possible to broadside with the target given other motion constraints.

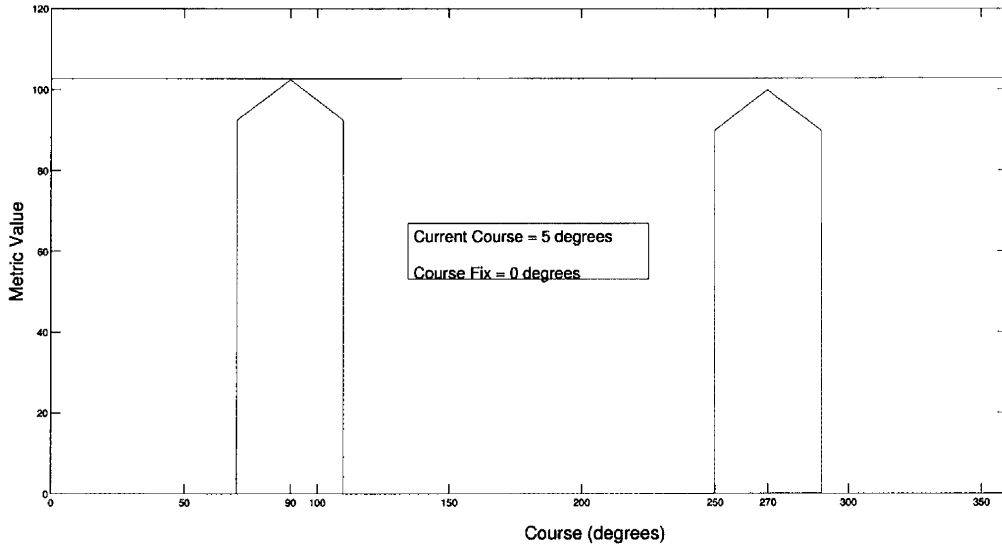


Figure 4-4: Objective function for the ArrayTurn behavior. This figure shows a plot of the objective function for the ArrayTurn behavior for a course fix of zero degrees and a current course of five degrees. Note how the mode closest to the current course is weighted slightly higher.

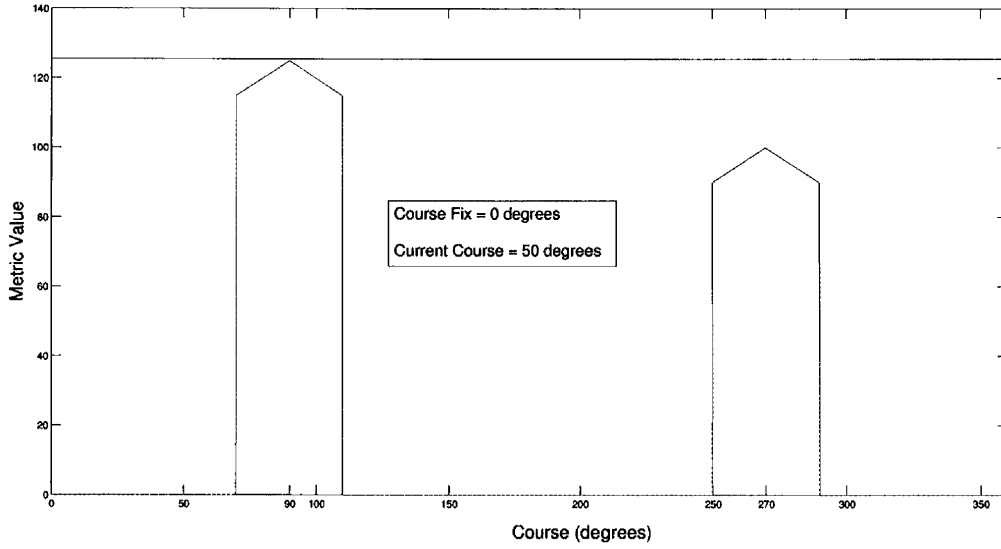


Figure 4-5: Objective function for the ArrayTurn behavior. This figure shows a plot of the objective function for the ArrayTurn behavior for a course fix of zero degrees and a current course of fifty degrees. Note how the mode closest to the current course has increased its weight relative to the other mode for the situation shown in Fig 4-4.

4.2.5 ArrayAngle Behavior

The ArrayAngle behavior (see Fig. 4-3(b)) is responsible for holding a vehicle course such that sensor platforms with acoustic line arrays will have the array as close as possible to broadside with the target given the other constraints on vehicle motion. The objective function for this behavior is one-dimensional over course and bimodal, with the modes centered around the two possible course choices that keep the array oriented at broadside with respect to the target. The mode that is centered at the course closest to the vehicle's current course is weighted in order to prevent frequent oscillation between the two modes. Fig 4-6 shows a plot of the objective function for the ArrayAngle behavior for a target bearing of zero degrees and a current course of fifty degrees. Note how the mode closest to the current course is weighted slightly higher. Fig 4-7 shows a plot of the objective function for the ArrayAngle behavior for a target bearing of zero degrees and a current course of minus fifty degrees. Note how the mode closest to the current course has increased its weight relative to the other mode for the situation shown in Fig 4-6. Fig 4-8 shows a plot of the dynamic weighting for the ArrayAngle behavior. Beyond a specified maximum range, the weighting of the ArrayAngle behavior is 0.1 otherwise it is weighted at 1.0.

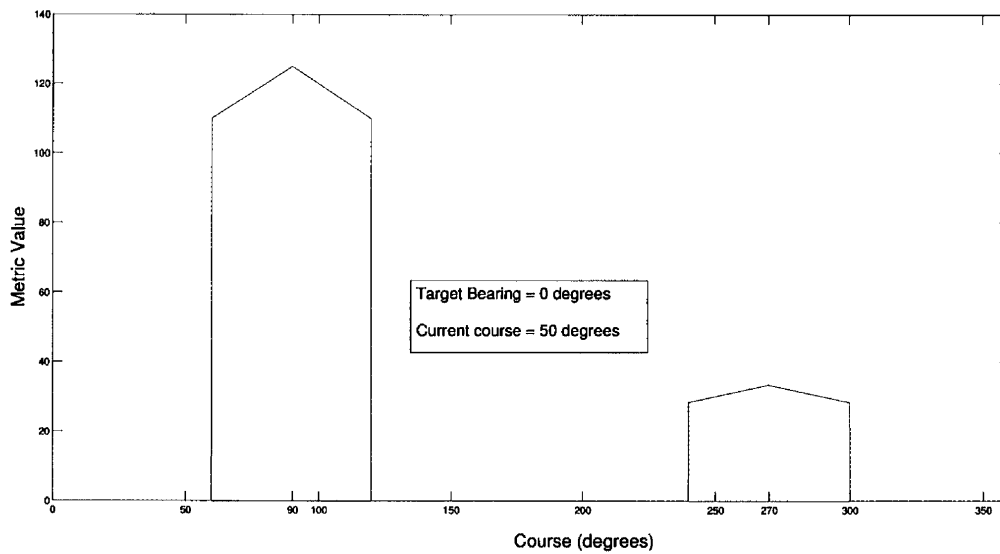


Figure 4-6: Objective function for the ArrayAngle behavior. This figure shows a plot of the objective function for the ArrayAngle behavior for a target bearing of zero degrees and a current course of fifty degrees. Note how the mode closest to the current course is weighted slightly higher.

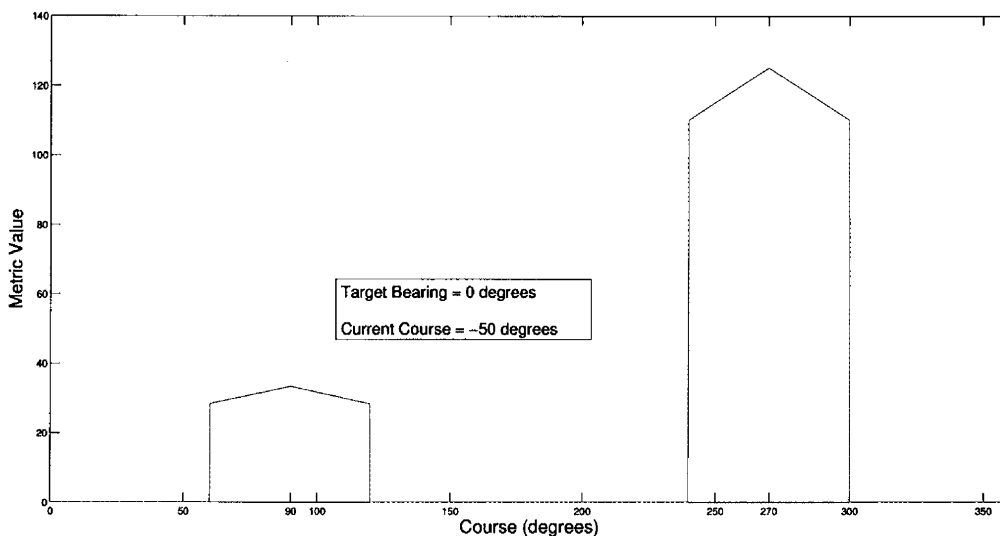


Figure 4-7: Objective function for the ArrayAngle behavior. This figure shows a plot of the objective function for the ArrayAngle behavior for a target bearing of zero degrees and a current course of minus fifty degrees. Note how the mode closest to the current course has increased its weight relative to the other mode for the situation shown in Fig 4-6.

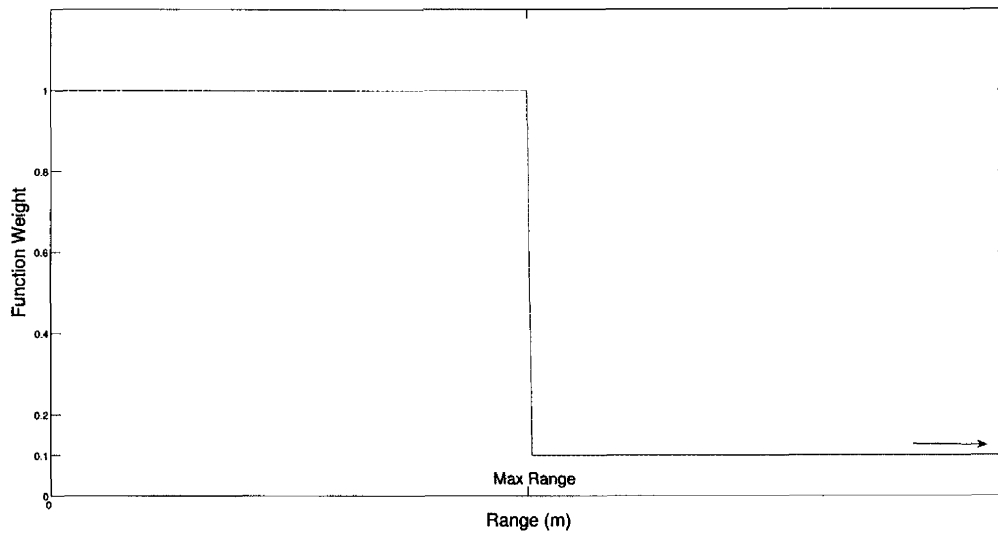


Figure 4-8: Dynamic Weighting for the ArrayAngle Behavior. This figure shows a plot of the dynamic weighting for the ArrayAngle behavior. Beyond a specified maximum range, the weighting of the ArrayAngle behavior is 0.1 otherwise it is weighted at 1.0

4.2.6 CloseRange Behavior

The CloseRange behavior is designed to close the distance to a target being tracked by the on board sensor subject to a minimum approach distance. The behavior produces objective functions that are three-dimensional over course, speed, and time and rates actions favorably that have a smaller closest point of approach (CPA). Fig. 4-9 shows a plot of the dynamic weighting for the CloseRange behavior. Below a specified minimum range, the CloseRange behavior has a weight of zero, and increases linearly to a weight of 1.0 at a range of 333m beyond the minimum range. These values are configurable depending on the scale of the experiment.

4.2.7 Classify Behavior

The Classify behavior used in this demonstration is active on the classify vehicle and is identical to the CloseRange behavior described in 4.2.6 with the exception that the target track information is provided from an external source (in this case the tracking vehicle), instead of an on-board sensor.

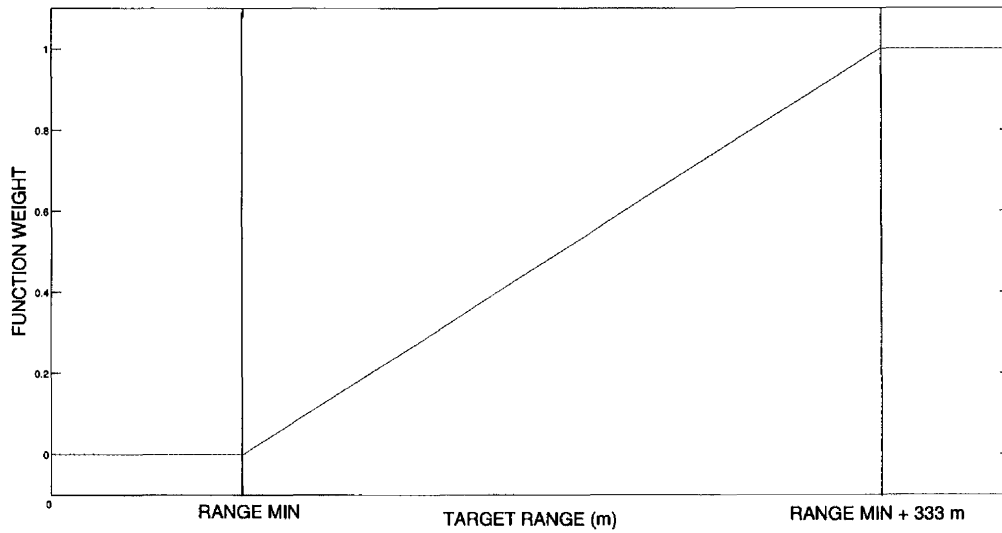


Figure 4-9: Dynamic Weighting for the CloseRange Behavior. This figure shows a plot of the dynamic weighting for the CloseRange behavior. Below a specified minimum range, the CloseRange behavior has a weight of zero, and increases linearly to a weight of 1.0 at a range of 333 *m* beyond the minimum range. These values are configurable depending on the scale of the experiment.

4.2.8 Formation Behavior

The formation behavior is responsible for maintaining two sensor platforms in formation in a track and trail scenario behind the target using the current target position estimate as a virtual leader. The optimal formation consists of the sensor platforms maintaining a ninety degree angle with respect to the target position estimate while trailing at a fixed trail distance r . The objective functions for this behavior are three dimensional over course, speed and time. shows a plot of the metric applied to a proposed combination of course, speed, and time, that results in a value for the separation angle between this sensor platform and its partner sensor platform. It should be noted that the separation is computed using the current position of the other sensor platform which is also calculating the separation angle. This can lead to dynamic instability problems if there is not enough damping in the vehicle motion. The formation behavior always has a weighting of 1.0.

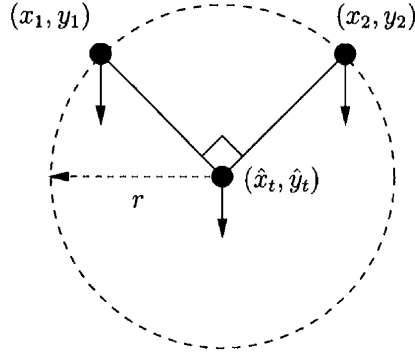


Figure 4-10: Formation behavior for 2-vehicle cooperative target tracking. The formation behavior is responsible for maintaining two sensor platforms in formation in a track and trail scenario behind the target using the current target position estimate as a virtual leader. The optimal formation consists of the sensor platforms maintaining a ninety degree angle with respect to the target position estimate while trailing at a fixed trail distance r .

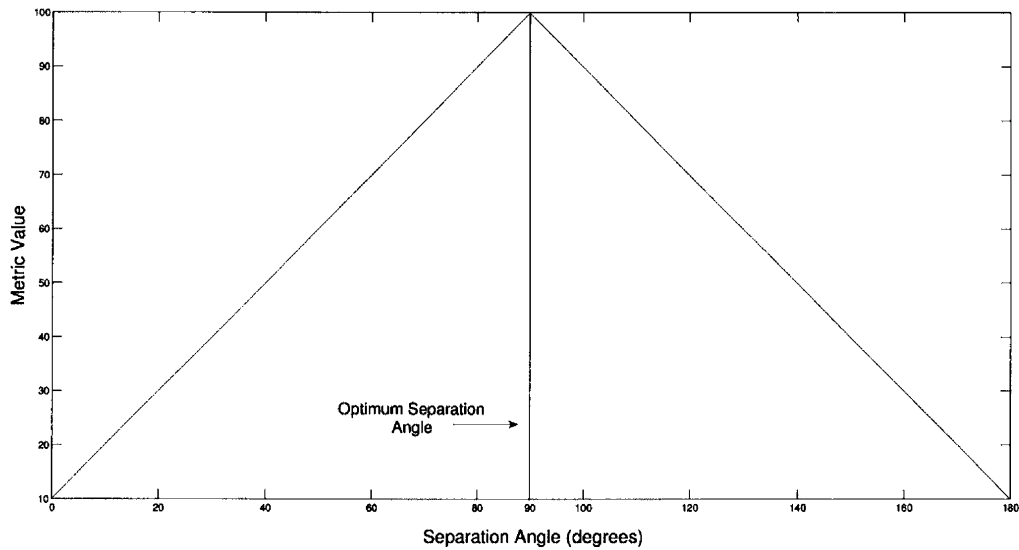


Figure 4-11: Formation Behavior Metric. This figure shows a plot of the metric applied to a proposed combination of course, speed, and time, that results in a value for the separation angle between this sensor platform and its partner sensor platform. It should be noted that the separation is computed using the current position of the other sensor platform which is also calculating the separation angle. This can lead to dynamic instability problems if there is not enough damping in the vehicle motion.

Chapter 5

An AUV Intelligent Sensor for Real-Time Adaptive Sensing

As discussed in Section 1.1.2, adaptive sensor platform motion requires tight integration between the sensors and the control system. However, the concept of a “logical sensor” [57][58][59] allows an abstract view of a sensor that allows the actual details of the physical sensor to be hidden or abstracted away in much the same way as an abstract data type does in software engineering [4]. This is especially useful if multiple physical sensors contribute to forming a piece of sensory information. In [4], the authors give an example of a logical sensor-based range finder based on the inputs from three physical sensors, two of which are optical cameras and the third being an ultrasonic sensor. The logical sensor contains all of the processing algorithms necessary to form a range from the three physical sensors. The control system of a sensor platform with this logical range finder sensor would then have access to the composite range output without having to worry about how that range was developed. Such a logical sonar sensor has been developed to support a number of adaptive sampling projects such as the GOATS program for multi-sensor mine counter-measures (MCM) and the PLUSNet program for detection and tracking of moving underwater targets.

5.1 A Logical Sonar Sensor

The logical sonar sensor consists of the physical acoustic sampling hardware as well as algorithms that abstract the real-time data into higher forms of information suitable for the behavior-based

control system. Because of the distributed MOOS architecture, the actual sensor and processing algorithms (MOOS processes) of the logical sensor may well reside in a separate vehicle payload from the main vehicle control computer. The tracking vehicles in this work use a set of tracking algorithms that run in a single MOOS process called pTracker (see Fig. 4-1(a)). This process subscribes to target bearing data from the MOOS database as input to the tracking algorithms. The bearing data is either produced by another MOOS process interfaced with a physical bearings-only sensor, or the bearing data is produced by an alternative MOOS process that simulates bearings-only sensor data. The pTracker process then produces and posts track solution information to the MOOSDB to be consumed by any other MOOS process including inter-vehicle communications processes like pMOOSBridge or iAcousticModem or the behaviors in the vehicle control system. Feedback from the platform behaviors is available for dynamically changing the sensor parameters in response to the platform state. More information on the algorithms for the pTracking process is given in Chapter 6 for the passive tracking with a single sensor platform and in Chapter 7 for passive tracking with two sensor platforms.

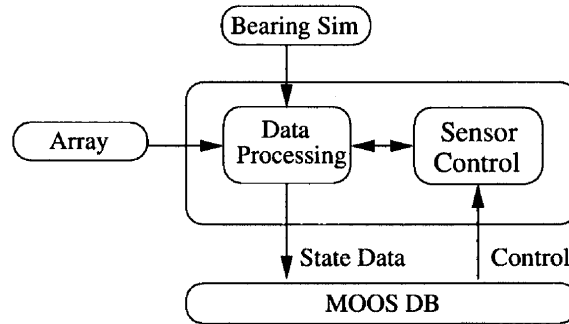


Figure 5-1: The logical sonar sensor. Rather than passing raw bearing data directly to the platform control system, the sensor processes the bearing data into a higher level of abstraction suitable for a behavior-based control system. Feedback from the platform behaviors is available for dynamically changing the sensor parameters in response to the platform state.

The integration of the physical acoustic sensor and the processing algorithms into a logical tracking and localization sensor allows the behavior-based control system to make decisions about vehicle control based on high-level state information like a target track or a target location without having to worry about how the information was formed. This modular, distributed approach to sensing integrates very well with the MOOS-IvP architecture for adaptive vehicle control.

5.2 Design Goals for the Physical Sonar Sensor

Given our principal goal of being able to perform multi-static detection and classification of proud and buried targets in real-time as well as passive tracking of acoustic targets using AUVs, the following design goals were adopted for the sonar payload:

1. **Mechanical/Electrical** The sonar payload should integrate with the Odyssey-III AUV manufactured by the Bluefin Robotics Corporation and currently used by both MIT and the U.S. Navy as research vehicles (see 2.1.2). The power consumption of the payload should allow at least three hours of vehicle run-time before recharging of the batteries is necessary. Because the application demands operation only in shallow or very shallow water, a depth rating of 100 msw is considered sufficient. The pressure vessel should be capable of dissipating up to 150 watts of internally generated heat while keeping the internal temperature at a maximum of 45 degrees Celsius while underwater.
2. **Receiving Array Integration** The sonar payload should be capable of integrating with both single and dual 16-element line arrays.
3. **Acoustic Source** The sonar payload should be able to drive an acoustic source with up to 200 watts of power in the 4-24 kHz range. The acoustic source should be side-looking with respect to the main vehicle axis and be adjustable in angle from fully horizontal to fully vertical downward. The sonar payload should have the capability to drive the acoustic source with a number of different waveforms that can be chosen on the fly.
4. **Data Acquisition** Given the need for high fidelity processing of the sonar data, the sonar data acquisition should provide 16-channel simultaneous sampling with a precision of at least 16 bits and a maximum sampling rate of 100 kHz. In order to obviate the need for analog anti-aliasing filters, sigma-delta conversion will be used on all analog to digital (A/D) converters. All acquired data samples should be saved to hard disk for offline processing.
5. **Time Synchronization** The sonar payload should have the ability to synchronize itself to Coordinated Universal Time (UTC) with an accuracy of at least 1 microsecond while the AUV is on the surface. While submerged, the sonar time reference should drift no more than 1 microsecond per hour. The payload time reference system should be able to interface to the data acquisition system such that time tags can be generated by the time



(a) An AUV with a single line array.



(b) An AUV with a dual line array.

Figure 5-2: This figure show the Odyssey AUV in two configurations, one with a 16 element single line array cut for $15kHz$ and one with dual 8 element arrays cut for $7.5kHz$.

reference system in response to hardware triggers from the data acquisition system (for the purpose of time-tagging acquired data samples).

6. **Vehicle Communications** The sonar payload should have the ability to communicate with the main vehicle computer via Ethernet.

7. **Integration as a Logical Sensor** One of the key robotic techniques that will enable advanced, high-level vehicle control in a cooperative, multi-vehicle framework is the concept of the logical sensor. Under this concept, a sensor (the sonar payload in this case) will communicate with the main vehicle operating system to send meta-data and requests and to receive commands. Meta-data is high-level, processed data as opposed to raw data samples. For example, a piece of meta-data that might be sent from the sonar payload to the vehicle operating system is “target detected at coordinate (x,y,z)” or a set of target track parameters. As can be seen, this technique uses a level of data abstraction that is one level higher than is normally seen from a sensor, e.g. converting a voltage signal into a pressure. Another hallmark of the logical sensor concept is that it allows the sensor to send requests to the vehicle operating system. For example, the sonar payload may send requests for specific vehicle movements as needed to optimize the target detection and classification based upon its processing of received data. The vehicle operating system determines how best to deal with requests from all logical sensors in accordance with mission parameters.

The integration of the sonar payload as a logical sensor requires it to have full-duplex communications with the main vehicle operating system as well as the capability to run whatever algorithms are necessary for the real-time detection, classification, and tracking of targets.

8. **Ease of use as a research platform** Because this system is intended to be used as a research platform maintained and operated by graduate students, ease of use and the ability to upgrade or change the system are of considerable importance. Every effort should be made to use commercial off-the-shelf hardware components and software that is well known and widely available to program the system.

5.3 Sonar Implementation

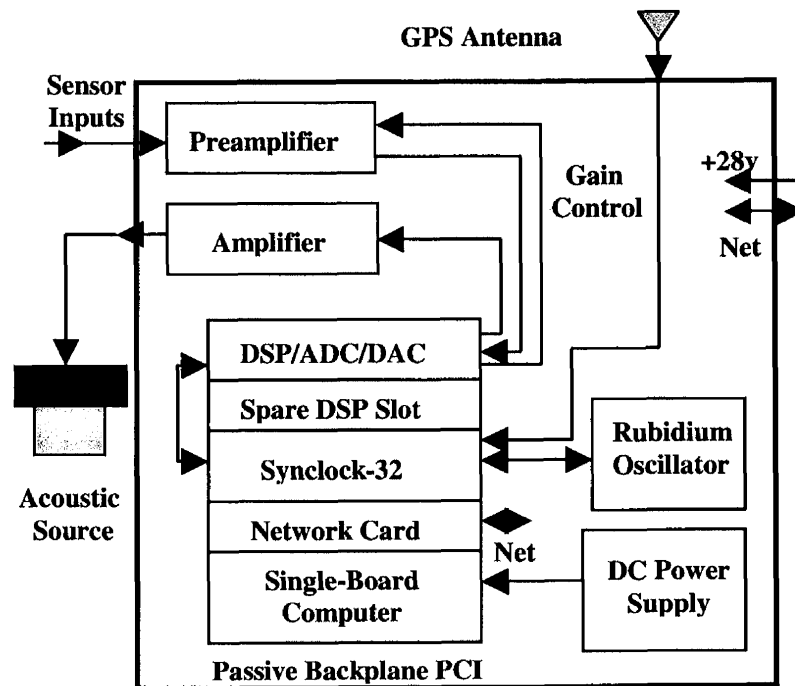


Figure 5-3: This figure shows the functional block diagram of the logical sonar sensor.

A functional block diagram of the sonar implementation is shown in 5-3.

5.3.1 Mechanical/Electrical

A standard Bluefin Robotics Corporation aluminum pressure vessel was used to house the sonar electronics. This pressure vessel is cylindrical with hemispherical end caps. It can be seen in Fig. 5-4 resting in a standard Odyssey-III payload section. This pressure vessel is capable of dissipating up to 150 watts of internally generated heat while maintaining a temperature below 45 degrees Celsius while underwater. The total power consumption of the payload is approximately 115 watts while in operation with the acoustic source transmitting three to five pings per second. The only connection between the sonar payload and the main vehicle consists of a 28-volt (nominal) power connection and a 10 Mbps Ethernet connection. Note the acoustic source mounted on a mechanism that allows it to be rotated between fully horizontal and fully vertical.

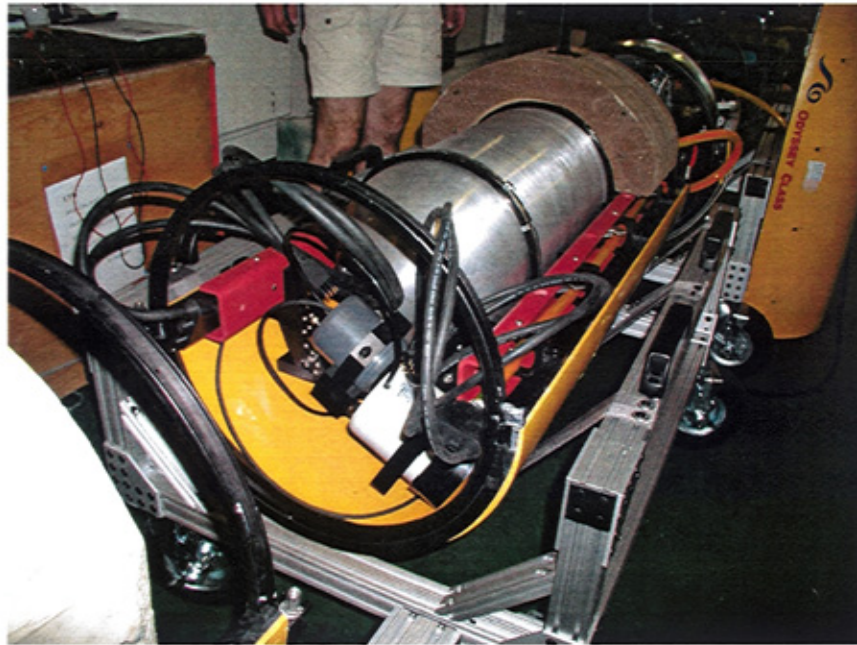


Figure 5-4: This figure shows the sonar payload in the Odyssey-III AUV payload section.

5.3.2 Receiving Array Integration

The sonar payload was configured to operate with both a 16-element single line array and 16-element dual line array. Fig. 5-2(a) shows the Odyssey vehicle with the single line array while Fig. 5-2(b) shows it with the dual line array.

5.3.3 Analog Processing Section

A high-performance preamplifier with low noise characteristics is vital to obtaining quality sonar data. The preamplifier used in the MIT sonar system was designed and built at the Woods Hole Oceanographic Institution. It is needed to boost the signal levels from the hydrophones up to the 4-volt p-p needed by the digitizers in the signal processing section. It is capable of three levels of gain, 0 dB, +20 dB, and +40 dB, selectable on the fly from the signal processing section via a control line. The preamplifier was designed to have a relatively flat pass band in the 4-24 kHz region.

5.3.4 Host Computer Architecture

The major issue driving the architecture of the host computer platform is that of having enough bandwidth on the host computer's data bus to successfully stream raw data samples to the hard disk from the data acquisition system. Sampling 16 channels at 100 kHz using 16-bit samples produces a data rate of 3.2 megabytes per second. The two computer architectures that were considered were the PC-104 and the Peripheral Component Interface (PCI) bus architectures. The advantages of the PC-104 architecture are in its compact size and low power consumption. However, one major flaw is its 16-bit data bus as opposed to the 32-bit bus of the PCI architecture. It was felt that the PC-104 data bus would be only marginally capable of performing the continuous data transfer without even taking into account use of the bus by other peripheral cards. A lack of available off-the-shelf, high-performance, 16-channel signal processing peripheral cards was also a factor.

Taking those factors into consideration, the PCI architecture was chosen for the sonar host computer. In order to minimize space requirements in the pressure vessel, a passive backplane construction was chosen utilizing a single-board computer. A single board computer is a complete computer system on a PCI-compatible board. The single-board computer card is inserted into the passive backplane PCI bus thereby giving other peripheral cards access to the CPU. The board chosen for this implementation was based on a 266 MHz Pentium processor with 128 megabytes of main memory. Since this board did not have an integrated network card, a stand-alone network card was used in one of the free PCI slots.

Another significant decision for the host computer architecture is the choice of operating system. Both Linux and Microsoft Windows were considered. In this case, Linux was chosen for

its cost (free), ease of programming (most graduate students are familiar with programming in Linux but not in Windows), and its speed. One drawback to the choice of Linux is the lack of availability of commercial device drivers for many peripheral cards of interest.

5.3.5 Data Acquisition Subsystem

The data acquisition subsystem is the most critical part of the sonar. High quality, low noise data is essential for detecting and classifying targets in the ocean environment, especially when the data analysis must be done in real-time without the benefit of powerful computers and hours of offline processing. The acquisition system chosen for the sonar payload is based on the Heron modular digital signal processing (DSP) system manufactured by Hunt Engineering. This DSP system is specifically designed for demanding, real-time applications. It utilizes a modular, extensible hardware architecture that accommodates multiple TMS320C6000 DSP processors, multiple Virtex-II floating point gate arrays (FPGAs), and fully integrated ultra-fast A/D, D/A and Digital I/O interfaces. For this system, the Heron HEPC8 Module Carrier Board was chosen. This PCI form factor module carrier board supports up to four Heron modules which can be multiple



(a) A sphere insonified at 3 kHz



(b) A cylinder insonified at 5 kHz.

Figure 5-5: This figure shows the Heron-4 DSP board and the rubidium oscillator used in the sonar payload.

combinations of DSP, FPGA, A/D, D/A, or I/O modules. The HEPC8 provides 32-bit first-in, first-out (FIFO) buffers between each module slot and the other modules slots on the board

for data transfer between Heron modules. One FIFO on the board is also connected to the board PCI interface for data transfer between a Heron DSP module and the host computer. The HEPC8 is shown in Fig. 5-5(a). The HEPC8 in the sonar payload contained four Heron modules; one Heron4-C6701 floating point DSP, one HEGD5 D/A converter, and two HEGD-12 A/D converters. A functional block diagram of the data acquisition subsystem utilizing the Heron modular system is shown in Fig. 5-6. The DSP in this system has a number of roles. First, it waits for commands from the host computer (sent over the PCI bus) to begin sampling. Second, it outputs the transmit waveform to the D/A converter module and triggers the power amplifier via an output control line. Third, it sends a TTL pulse trigger to the timing subsystem to generate a time tag via another control line. Fourth, it collects incoming data samples from the two A/D converters and transfers them to the host computer over the PCI bus. The DSP in this system was not used for signal processing but may be in future revisions. The Heron4-C6701 DSP module contains a 167 MHz Texas Instruments TMS320C6201 floating point DSP capable of 1 gigaflop of performance. The DSP module also provides numerous digital input and outputs that can be used for control of or communication with other system components.

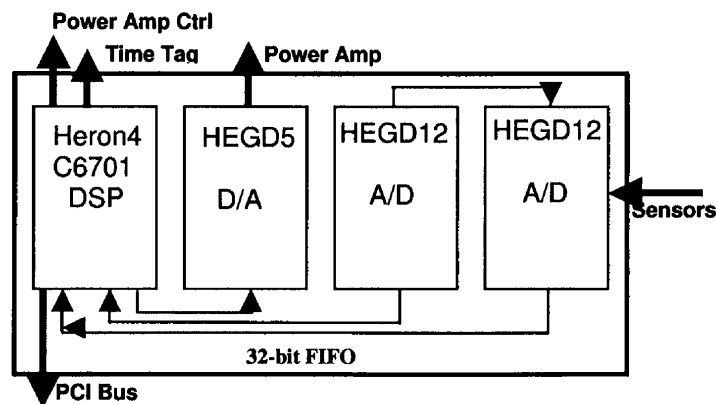


Figure 5-6: This figure shows the sonar payload in the Odyssey-III AUV payload section.

The HEGD5 is a four-channel, 16-bit D/A converter that is used to receive a digitized waveform from the DSP via a 32-bit FIFO and to output an analog signal to the power amplifier for the acoustic source. The sampling rate of the HEGD5 is 230 kHz. The HEGD12 is an eight-channel, 16-bit A/D converter that is used to digitize the input waveform received from the preamplifier. The HEGD12 provides simultaneous sampling on all eight channels and utilizes sigma-delta conversion on all channels. The sigma-delta conversion technique involves over-sampling the input

waveform by a factor of eight times, digitally filtering the over-sampled data, and then down-sampling the resulting data by a factor of eight. This technique eliminates the need for analog anti-aliasing filters. An onboard crystal oscillator provides the sample clock for the HEGD12. The sample clocks of the two HEGD12 modules are tied together in order to provide 16-channel simultaneous sampling. Digitized samples are pushed into a 32-bit FIFO where they are read by the DSP.

5.3.6 Time Reference Subsystem

The time reference subsystem is the key to being able to use multi-static sonar techniques for detection and classification due to the need for accurate travel times between waveform transmission by the acoustic source and reception of the scattered waveform by each of the receiving arrays on multiple vehicles. In order to accomplish this, the sonar payloads on each vehicle must be synchronized in time. This synchronization must be maintained for the duration of the mission. The solution to this problem is to synchronize each vehicle on the surface via GPS and to maintain this synchronization with an internal oscillator while the vehicles are underwater and unable to access the GPS satellites. This was accomplished in the MIT sonar payload by using a digital clock card. This card, the Synclock-32 manufactured by JXI2 incorporated, has an onboard GPS receiver which can be used to synchronize the clock on the card to GPS time to within 500 nanoseconds of UTC. Once synchronized, however, all clocks will drift (either fast or slow) and the Synclock-32 will begin to drift once the GPS input is gone. The rate of drift will depend on the quality of the oscillator that is used to keep time on the clock. Unfortunately, standard crystal oscillators do not have the capability to meet the stringent drift requirement of 1 microsecond per hour that is needed for this application. The solution to this problem is to use a rubidium oscillator to keep time. A rubidium oscillator is actually a small atomic clock. The rubidium oscillator chosen for the sonar payload is the RMO rubidium oscillator manufactured by Temex in Switzerland. This oscillator has a drift specification of 700 nanoseconds per hour and consumes 10 watts of power. See Fig. 5-5(b) for a picture of the RMO. Time tags can be generated by the Synclock-32 upon reception of a digital control input from the DSP. Once the time is latched, the Synclock-32 generates a PCI bus interrupt that is intercepted by a device driver that reads the time tag from the Synclock-32 and places it in a file. Since the payload on each vehicle can be synchronized within 500 nanoseconds of UTC, the maximum initial time

error between the transmitting vehicle and each and receiving vehicle is 1 microsecond.

5.3.7 Acoustic Source

The acoustic source used in the payload is the acoustic source from the SB-24 sub-bottom profiler tow sled manufactured by EdgeTech. It has a frequency range of 4-24 kHz and a beam width of 15-30 degrees depending on the transmit frequency. The power amplifier used to drive the source has a maximum output power of 200 watts.

5.3.8 Software Architecture

The software architecture of the sonar includes processes running on two different processors; the DSP in the data acquisition subsystem and the host computer. The software on the host computer is responsible for bi-directional communications with the main vehicle operating system, booting the DSP with an executable image, bi-directional communications with the process running on the DSP, transferring sample data from the DSP and saving it to the hard disk, and for running any data analysis algorithms such as detection and classification algorithms. A single multi-threaded process on the host computer accomplishes the first four of these tasks while the detection and classification algorithms are run simultaneously as individual processes. The main communications process is compiled with a library provided by the main vehicle operating system which makes communication with the operating system as simple as making a function call. An Application Programming Interface (API) provided by Hunt Engineering allows booting of the DSP with an executable image. Communications with the DSP is accomplished through a set of function calls provided by the API. All code on the host computer is written in C or C++ using the Linux compiler. The process running on the DSP is responsible for communicating with the host computer to send status or to read commands, reading samples from the A/D converters, transmitting the output waveform, triggering the time tag subsystem and for transferring sample data to the host computer. All code development for the DSP was done using the Texas Instruments Code Composer Studio. Communications between the DSP and host processor was accomplished by compiling the DSP code with an API provided by Hunt Engineering.

5.4 Experimental Results

The sonar payload was tested in three different international experiments. The GOATS 2000 experiment, conducted in conjunction with SACLANTCEN off of Elba Island Italy, tested only the time synchronization subsystem while the GOATS 2002 experiment, also held in conjunction with SACLANTCEN off the coast of Italy, tested the full sonar payload in a mono-static configuration with a single AUV. During the GOATS 2000 experiment, the time synchronization subsystem was successful in synchronizing the digital clock card to GPS time and in maintaining the time synchronization with the rubidium oscillator while the vehicle was underwater. The system was also successful in time-tagging sonar data collected by the AUV. During this experiment, the underwater targets were insonified by a tower-mounted acoustic source. The transmissions from the acoustic source were GPS time-tagged by the shore station that was controlling the transmissions. A major goal of the GOATS 2002 experiment was to test the complete sonar payload in an operational environment. This involved programming the AUV to conduct a sonar search of a patch of sea bottom where several buried and proud targets had been placed. A simple target detection algorithm, developed by several graduate students in the department, was run on the sonar host computer. This detection algorithm was fed a continuous stream of sonar data as the AUV conducted a search pattern in the area of interest. Detections were logged in a file for offline analysis. The online detection algorithm was successful in detecting a number of targets in real time. Fig. 5-7 shows the raw sonar data for one such online detection [60]. The sonar returns from the detected target can be seen inside the white box on the figure. Note the familiar hyperbolic shape of the target plot. This is exactly the pattern we expect to see from an AUV moving in a straight line past a point target. These extremely important results validate our sonar design and show that the data quality from the sonar is high enough to be used for real-time target detection. Although more extensive online target tracking and classification algorithms were not available at the time of the GOATS 2002 experiment, extensive offline analysis of the sonar data continues to show its superb quality and suitability. One of the challenges to online target detection is in trying to sort out spurious returns from those returns from actual point targets.

The purpose of the GOATS 2004 experiment was the investigation of bi-static and multi-static MCM scenarios. Fig. 5-8 shows a spectrogram of a multi-static ping reception. This figure shows the spectrogram of the reception of two sonar pings transmitted in a multi-static MCM

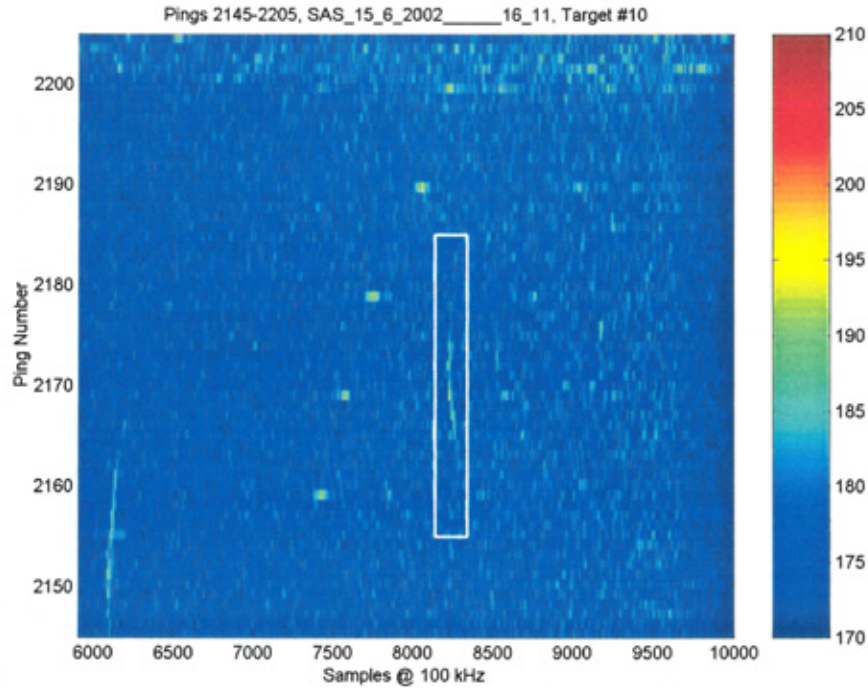


Figure 5-7: This figure shows an online detection of a fixed target with the sonar payload in MCM mode. Note the classic hyperbolic shape of the sonar returns over time.

scenario. Two CHIRP signals are clearly seen, one increasing in frequency, the other decreasing. Each CHIRP signal was transmitted by a separate sensor platform.

Fig. 5-9 and Fig. 5-10 show experimental results from the FAF '05 experiment where the sonar was run in a passive acquisition mode. In this mission, the AUV was run in a rectangular box with the source at a fixed location. The goal was to compute bearings to an acoustic source at a fixed location. Fig. 5-9 shows absolute target bearings computed both in real-time and then subsequently in post-processing after tuning parameters and modifying the beam tracking algorithm. This was done in order to increase performance and correct for an incorrect left/right ambiguity decision for $t > 500$ s. Fig. 5-10 shows bearing lines originating from logged AUV navigation position data. This demonstrates consistency of bearing line triangulation with the logged GPS locations of the ship towing the source.

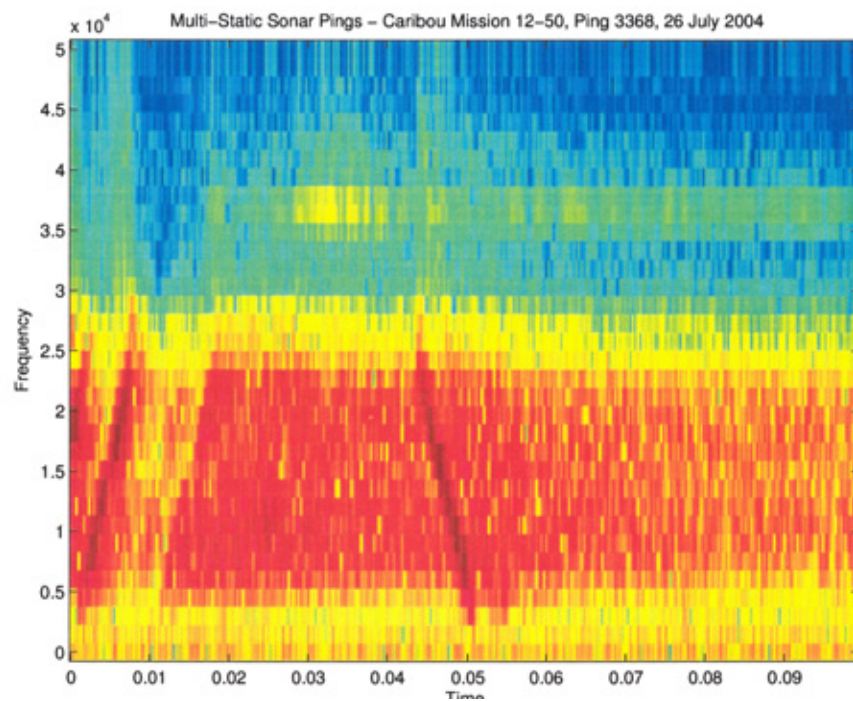


Figure 5-8: A spectrogram of a multi-static ping reception. This figure shows the spectrogram of the reception of two sonar pings transmitted in a multi-static MCM scenario. Two CHIRP signals are clearly seen, one increasing in frequency, the other decreasing. Each CHIRP signal was transmitted by a separate sensor platform.

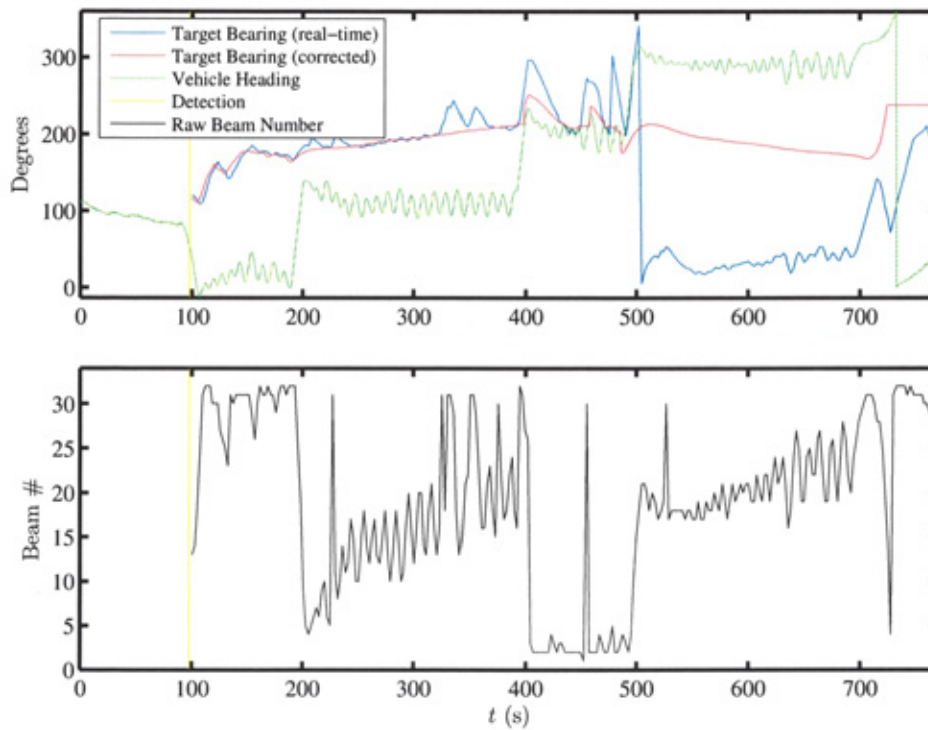


Figure 5-9: Absolute target bearings computed both in real-time and then subsequently in post-processing after tuning parameters and modifying the beam tracking algorithm. This was done in order to increase performance and correct for an incorrect left/right ambiguity decision for $t > 500$ s.

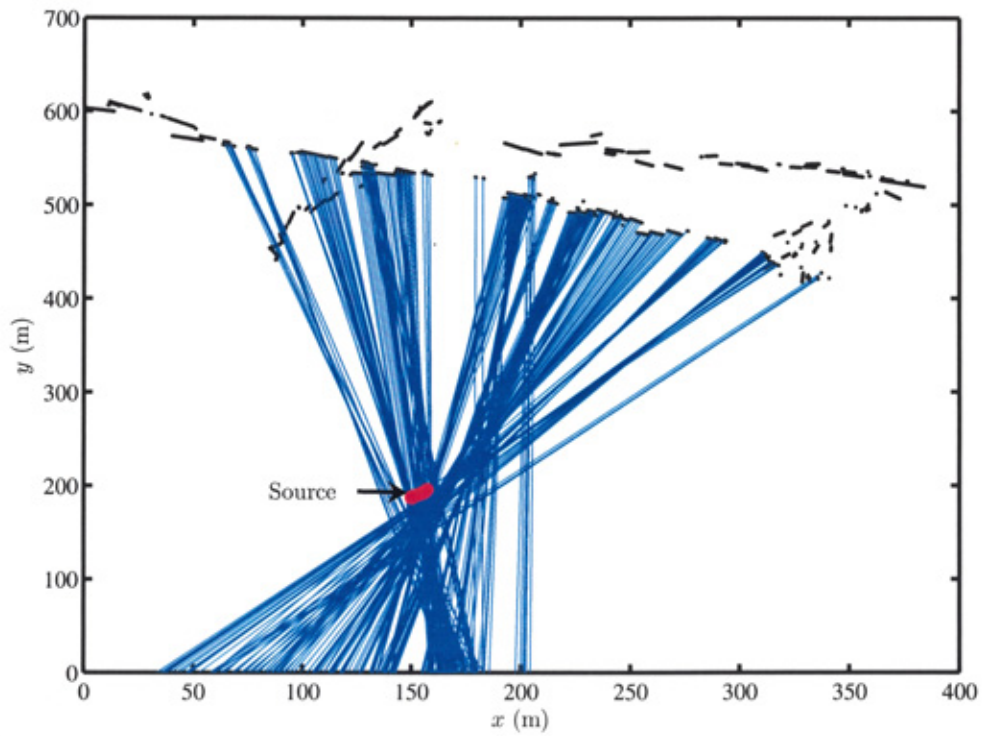


Figure 5-10: Bearing lines originating from logged AUV navigation position data. This demonstrates consistency of bearing line triangulation with the logged GPS locations of the ship towing the source.

Chapter 6

Example One: Adaptive Track and Classify

6.1 Introduction

We are motivated by the following scenario: two heterogeneous vehicles are in operation, the first is fitted with a passive, bearings-only towed sensor array and takes on the role of tracking other moving underwater objects of unknown trajectory and type.

The second vehicle is fitted with a different sensor more appropriate for detecting acoustic signatures of underwater objects and takes on the role of classifying other underwater objects. The two vehicles work together to track and classify underwater objects by communicating track solution information from the tracking vehicle to the classify vehicle via acoustic modem. The latter vehicle uses the track information to close its position on the object of interest to the benefit of its classification sensors. Each vehicle optimizes its trajectory to balance their sensing responsibilities alongside mutual relative position responsibilities.

In this chapter, we will first derive the mathematical basis for target tracking with a single mobile bearing sensor which will allow us to design the proper behaviors for the vehicle motion. Next, we will derive the target localization and tracking algorithms which reside on the intelligent sensor. Finally, we will present experimental validation of these concepts using three autonomous surface craft.

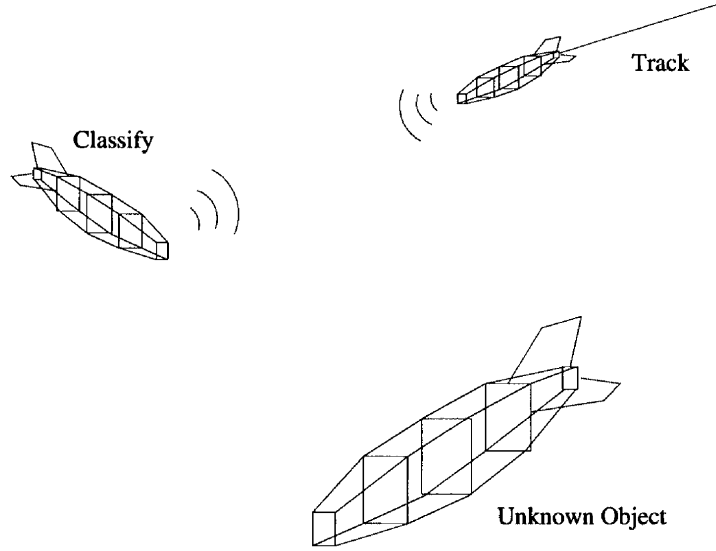


Figure 6-1: Two heterogeneous unmanned marine vehicles are in operation together. The first tracks the position and trajectory of unknown underwater objects using a towed linear array, and communicates track solution information via acoustic modem to a second vehicle with different sensors more suitable for classifying underwater objects.

6.2 Target Tracking with a Single Bearing Sensor

In order to track a moving object from a set of discrete sensor observations, one must first decide on the kinematic model used to describe the object's motion. In this work, a constant-velocity model was chosen because it is one of the simplest to describe mathematically and because estimating the motion of a constant velocity target using a bearings-only sensor is a classical problem in target motion analysis. Also termed "passive localization" or "passive ranging" this problem arises, for example, when trying to estimate the motion of a submarine moving at constant velocity from another submarine observing the target using a linear towed array sensor.

6.2.1 State Estimator Derivation

In formulating this problem, we follow a classical analysis as given in [61]. Consider a Cartesian coordinate frame having an object with position $[x_t[n] \ y_t[n]]^T$ and constant velocity $[\dot{x}_t \ \dot{y}_t]^T$ being tracked by a bearing sensor on a sensor platform with position $[x_p[n] \ y_p[n]]^T$ moving in the same plane with measurement observations taken at the discrete time intervals $n = 0, 1, \dots, N$. The

state equations for the target motion can be written in discrete time as

$$x[n] = x[0] + \dot{x}_t t_n \quad (6.1)$$

$$y[n] = y[0] + \dot{y}_t t_n \quad (6.2)$$

Given (6.1) and (6.2) we define the state parameter vector

$$\hat{x} \triangleq [x[0] \ y[0] \ \dot{x}_t \ \dot{y}_t]^T \triangleq [x_0 \ x_1 \ x_2 \ x_3]^T \quad (6.3)$$

All of the parameters in the state parameter vector are assumed to be statistically independent. The measurements are target bearings relative to the sensor platform given by

$$z[n] = h[n, x] + w[n] \quad (6.4)$$

where

$$h[n, x] \triangleq \tan^{-1} \frac{y_t[n] - y_p[n]}{x_t[n] - x_p[n]} \quad (6.5)$$

and $w[n]$ is the measurement noise assumed to be a Gaussian white noise sequence with variance q . Our sensor makes a sequence of bearing measurements which we combine into a single measurement vector Z .

Given our assumption of a constant velocity target, estimating the parameters in 6.3 from a sequence of observations will completely define the target motion. A number of different estimation techniques can be used to estimate the state parameter vector. These include the use of extended Kalman filters (EKF) and maximum likelihood (nonlinear least squares) estimators. An advantage of using a recursive estimator such as the EKF is that only the current measurement is needed to form the estimate, allowing the estimator to run in constant time with respect to the number of observations made. A significant disadvantage of the EKF is that it is a suboptimal estimator based on the linear Kalman filter with a modification that linearizes the nonlinear measurement equation about the latest estimate. In general, the EKF works well if the initial estimates and measurement noise are not “too large”, given a particular problem. The biggest advantage of the maximum likelihood estimator is that the estimates are optimal in a least squares sense. Disadvantages of the maximum likelihood estimator include the need to compute the minimum of the likelihood equation and the need to use the complete observation

vector to compute each estimate. The latter disadvantage leads to increasing computational load as the number of observations increases. Because computational load is not an issue in either our simulations or in our experimental apparatus, we will use a maximum likelihood estimator to estimate the state parameter vector in order to form the optimal estimate.

6.2.2 The Likelihood Function

Given the Gaussian noise assumption for our measurement, we define the negative log-likelihood function as

$$\lambda(x) = \frac{1}{2q} \sum_{n=1}^N [z[n] - h[n, x]]^2 \quad (6.6)$$

The maximum likelihood estimate is then formed by

$$\hat{x} = \arg \min_x \lambda(x) \quad (6.7)$$

The state parameter vector which satisfies (6.7) is the *maximum likelihood estimate*. The minimization required to satisfy (6.7) can be accomplished using a number of numerical techniques including Newton-Raphson, quasi-Newton, and simplex methods. Newton-Raphson and quasi-Newton methods require knowledge of the first derivatives of (6.6) with respect to the state parameters. In the following work, The minimization required to satisfy (6.7) was accomplished using the Broyden-Fletcher-Goldfarb-Shanno algorithm, a quasi-Newton method The derivatives (with irrelevant constants removed) of equation 6.6 are

$$\frac{\partial \lambda(x)}{\partial x_0} = \sum_{n=1}^N \frac{2(z[n] - h[x, n])(y_t[n, x] - y_p[n])}{(x_t[n, x] - x_p[n])^2 + (y_t[n, x] - y_p[n])^2} \quad (6.8)$$

$$\frac{\partial \lambda(x)}{\partial x_1} = \sum_{n=1}^N \frac{-2(z[n] - h[x, n])(x_t[n, x] - x_p[n])}{(x_t[n, x] - x_p[n])^2 + (y_t[n, x] - y_p[n])^2} \quad (6.9)$$

$$\frac{\partial \lambda(x)}{\partial x_2} = \sum_{n=1}^N -t[n] \frac{\partial \lambda(x)}{\partial x_0} \quad (6.10)$$

$$\frac{\partial \lambda(x)}{\partial x_3} = \sum_{n=1}^N -t[n] \frac{\partial \lambda(x)}{\partial x_1} \quad (6.11)$$

Any search method requires an initial starting point for the search. Ideally, the starting point should be carefully chosen to lie near enough to the actual solution in order to find the global minimum. In the following work, the minimization algorithm was initialized with a starting state

parameter vector $[x[0] \ y[0] \ \dot{x}_t \ \dot{y}_t]^T$ as follows:

1. \dot{x}_t and \dot{y}_t were initialized to 1.0 m/s
2. $x[0]$ and $y[0]$ are initialized by using an initial “range guess” along with the first bearing measurement to compute a starting point for the search. For each subsequent measurement, the state estimate from the previous measurement is used as the starting point for the search. For this work, the initial range guess value was 200 m.

6.2.3 The Cramer-Rao Lower Bound

The Cramer-Rao lower bound (CRLB) stipulates that the variance of our parameter estimates cannot be lower on average than a certain value determined by the shape of the likelihood function. The derivation and proof of the CRLB can be found in a number of textbooks on estimation theory including [61]. Formally, we say that

$$E [(\hat{x}(Z) - x)^2] \geq I_y(x)^{-1} \quad (6.12)$$

where $I_y(x)$ is known as the Fisher information matrix (FIM). The elements of the FIM are measures of the amount of “information” available about each parameter. Given our measurement vector Z and the Gaussian noise assumption, the diagonal elements of the FIM for this problem are

$$h_{x0}[n, x] = \sum_{n=1}^N \left[\frac{-(y_t[n] - y_p[n])}{(x_t[n] - x_p[n])^2 + (y_t[n] - y_p[n])^2} \right]^2 \quad (6.13)$$

$$h_{x1}[n, x] = \sum_{n=1}^N \left[\frac{(x_t[n] - x_p[n])}{(x_t[n] - x_p[n])^2 + (y_t[n] - y_p[n])^2} \right]^2 \quad (6.14)$$

$$h_{x2}[n, x] = \sum_{n=1}^N [t[n] h_{x0}]^2 \quad (6.15)$$

$$h_{x3}[n, x] = \sum_{n=1}^N [t[n] h_{x1}]^2 \quad (6.16)$$

The Cramer-Rao lower bound on the variance of each of our parameters is then found by inverting the FIM. By examining the elements of the FIM, several important issues can be noted. First, it is readily apparent that the number of observations N in our observation vector Z is a critical parameter determining the variance of our parameter estimates. Second, it is also

apparent that the relative positions of the sensor and target over time also play a critical role as is explored in section 6.2.4.

6.2.4 Parameter Observability

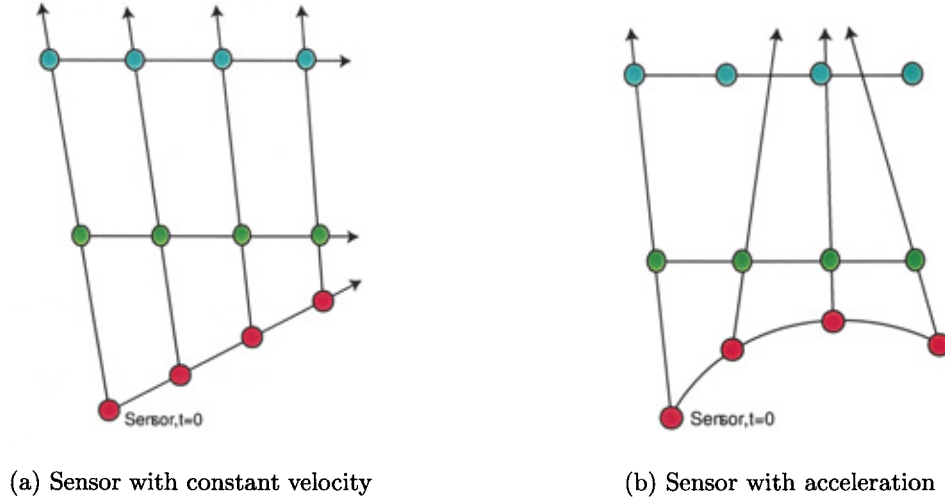


Figure 6-2: Parameter observability based on sensor motion. In 6-2(a), both the sensor platform and the target have constant velocity while in 6-2(b), the sensor platform has acceleration with respect to the target motion.

A well known constraint in tracking a constant-velocity target from a moving sensor platform is that, if the sensor platform also moves with constant velocity, the target motion parameters are unobservable. Therefore, the sensor platform must undergo an acceleration with respect to the target. A simple change of course can satisfy this condition. In 6-2(a), both the sensor platform and the target have constant velocity while in 6-2(b), the sensor platform has acceleration with respect to the target motion. As can be seen in the figures, the observations made with the sensor platform with constant velocity are parallel and , hence, redundant while the observations from the platform with acceleration are not. Redundant observations lead to an under-constrained system of equations when trying to solve for the target state parameters and produce a FIM which is not invertible. The degree to which the sensor motion improves the observability and, hence, the variance of the parameter estimates can be quantified by the condition number J of the FIM [61]. If J is too large, the FIM is ill-conditioned and the parameters are unobservable. Even if the FIM is invertible, the parameters may be marginally observable depending on the

actual value of J . The vehicle behaviors described in Section 4.2 are designed with the goal of producing a well-conditioned FIM.

6.2.5 Covariance of the Target Position Estimate

In many tracking applications, it is very useful to know at any time t_n what the estimated position of the target is and what our confidence in that estimate is (in a statistical sense). From the state equations for the target motion (6.1) and (6.2), we can develop the state transition matrix F_p which, when multiplied by the state parameter vector \hat{x} , will give us the estimated target position at any time t_n as follows

$$\hat{x}_p(t_n) = \begin{bmatrix} 1 & 0 & t_n & 0 \\ 0 & 1 & 0 & t_n \end{bmatrix} \hat{x} \triangleq F_p(t_n) \hat{x}_p \quad (6.17)$$

It is then a well known procedure to find the covariance matrix for a set of position estimates at any time $t_n \geq 0$ from the parameter estimate covariance matrix and the state transition matrix as follows

$$C_p(t_n) = F_p(t_n) C(t_n) F_p(t_n)^T \quad (6.18)$$

where $C_p(t_n)$ is the covariance matrix for the state parameter estimates at time t_n . It only remains then to determine the covariance matrix for the state parameter estimates. If the maximum likelihood estimator for this problem is efficient then the CRLB can be used as the covariance matrix for the state parameters. In [61], Bar-Shalom's analysis shows that the maximum likelihood estimator is indeed efficient for this problem. Therefore, we can calculate the covariance of our target position estimates in (6.18) using the CRLB ($C = I_y(x)^{-1}$).

From this covariance matrix (and the knowledge that the variance of our position estimate at any time t_n is a multivariate Gaussian distribution over x and y given our Gaussian noise assumptions on the parameter estimates), we can then plot the confidence region for the target position estimate at time t_n as the ellipse

$$[x - \hat{x}_p(t_n)]^T C_p^{-1}(t_n) [x - \hat{x}_p(t_n)] = g_p^2 \quad (6.19)$$

where g_p is the probability associated with the required confidence region (e.g. 90% or 0.9).

6.3 Experimental Setup

Experimental validation of the architecture and algorithms for autonomous bearings-only tracking, was conducted using two autonomous kayaks as the tracking and classify vehicles, and a third kayak as a moving object to be tracked and classified. The kayaks are proxies for autonomous underwater vehicles (AUV) used in upcoming follow-on experiments.

6.3.1 Simplifying Assumptions

Three simplifying assumptions were made. First, as a proxy for the towed array bearings-only sensor, the GPS position of the sensed vehicle was communicated over an 802.11b wireless connection to the sensing vehicle. The sensor vehicles converted (diminished) this information into bearings-only sensor data using a simulator which provided bearing data to the MOOS database just as the intelligent sensor currently in use on the AUVs would do. Although a bearing simulator of this nature does not have the same characteristics as a real acoustic array, the performance is acceptable within the ranges used in this experiment. Fig 6-3 shows the uncertainty of the simulated bearing sensor as a function of the position uncertainty of the target vehicle and range. The second simplification was the use of the 802.11b wireless connection as a proxy for communications via acoustic modem between the sensor vehicles. Given that acoustic communications is much slower than the wireless system used in this experiment, the simplification allowed the compression of the experiment in time in order to fit within the allowed physical boundaries of the test range.

6.3.2 The Marine Vehicle Platforms

The autonomous surface crafts used in this experiment are based on a kayak platform (Fig. 6-4). Each is equipped with a Garmin 18 GPS unit providing position and trajectory updates at 1 Hz. The vehicles are also equipped with a compass but the GPS provides more accurate heading information, and is preferred, at speeds greater than 0.2 m/s. Each vehicle is powered by 5 lead-acid batteries and a Minn Kota motor providing both propulsion and steerage. The vehicles have a top speed of roughly 2.5 meters per second. See [62] for more details on this platform.

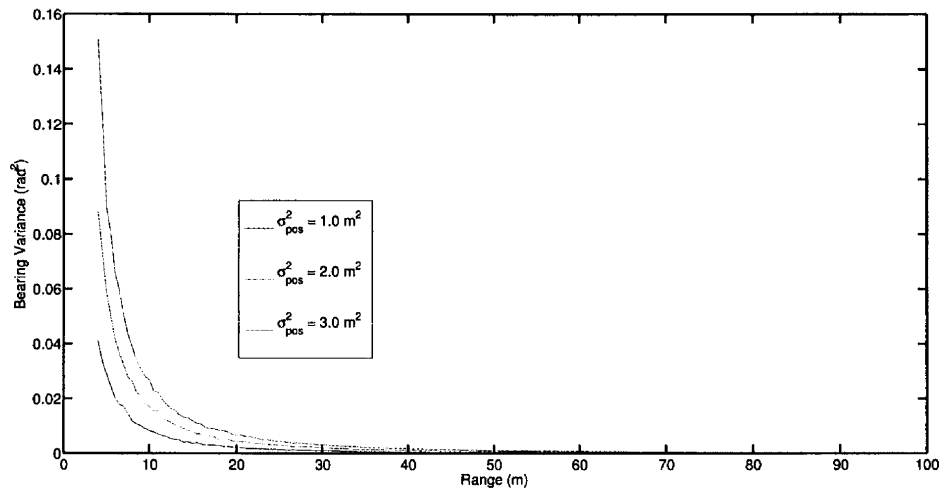


Figure 6-3: Bearing simulator uncertainty. This figure shows the variance of the bearing simulator used in the experiment as a function of target position variance and range.

6.3.3 Scenario

The experimental scenario begins with the deployment of the sensor and classification vehicles into separate patrol orbits where they will remain until a target detection occurs. At some point, the target kayak will begin its motion into the target area. When it enters into the target area, it will begin broadcasting its GPS location to the sensor vehicle whose sensor simulator will convert the position information into a target bearing. After clearing the left/right ambiguity on the simulated array, the sensor vehicle will begin tracking the target and broadcasting the target track information to the classification vehicle. While the sensor vehicle continues to track the target, the classification vehicle will simulate a classification run by closing range with the target estimate. When the classification vehicle closes to within a predetermined range from the target estimate, it will return to its patrol orbit. After a predetermined amount of tracking time, tracking will be declared over and the sensor vehicle will return to its patrol orbit to await another target. The target vehicle will return to its starting location.

6.3.4 Behavior Configurations

The three vehicles were configured with the following behaviors and preconditions. A condition is a “variable=value” pair in the MOOS Database. Details of the individual behaviors are given in section 4.2. A mission is started by broadcasting “deploy=true” to all vehicles and ended when the “return=true” message is broadcast. A broadcast is over 802.11b and changes a particular

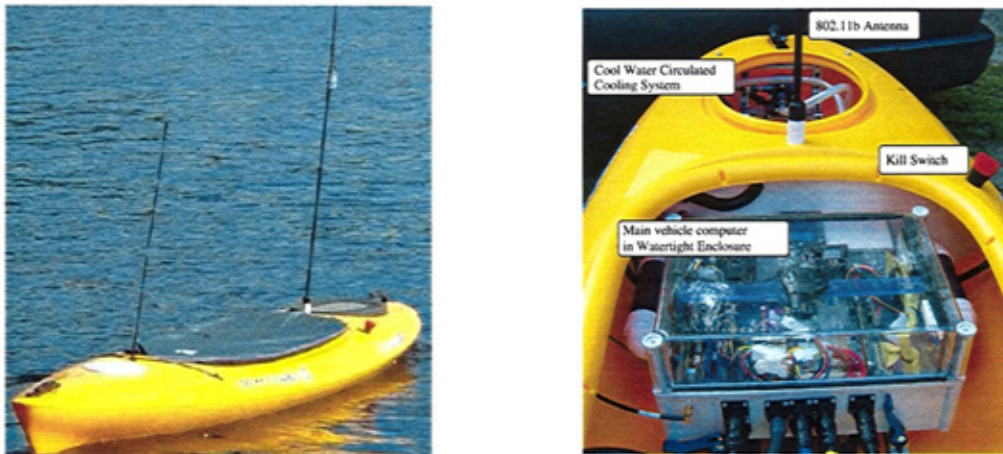


Figure 6-4: The kayak-based autonomous surface craft.

MOOS variable in the database resident on the vehicle. The broadcast could also be made via acoustic modem. All vehicle helms were configured with the OpRegion behavior as a safety measure. This behavior is active upon mission startup indicated by “deploy=true”.

The tracking vehicle helm was configured with an Orbit behavior which is active immediately upon mission startup indicated by “deploy=true”. The Orbit behavior is conditioned on not receiving bearing sensor data, i.e., “sensor_data=inactive”. It was also configured with ArrayTurn, ArrayAngle, and CloseRange behaviors. These three behaviors are conditioned on the vehicle receiving bearings-only sensor data, indicated by “sensor_data=active” in the MOOS Database.

The classify vehicle helm was also configured with an Orbit behavior that activated at mission startup. Additionally, the helm on this vehicle was configured with a Classify behavior which went active when target track solution data was received from the tracking vehicle. The Classify behavior was configured to deactivate itself when the CPA to the target vehicle reached 30 meters.

The target vehicle was configured to follow a simple set of waypoints, and was further configured to communicate its GPS position to the sensor vehicle. This communication only occurred when the target vehicle was within a certain specified region referred to as the “Sensor” region (Fig. 6-5). Deployment of the target vehicle was done via human command over wireless link when the other two vehicles had been on-station for an arbitrary sufficient time.

6.4 Experimental Results

6.4.1 Mission 1507

Fig. 6-5 shows the vehicle motion for an experimental track and classify mission with autonomous kayaks (see Fig. 6-4) with one tracking kayak, one classify kayak, and one target kayak. The objective of this mission is for the tracking vehicle to acquire and track the target vehicle while relaying target track solutions to the classify vehicle which then executes a simulated classification run.

In (a) the track vehicle and classify vehicle are deployed and executing their Orbit behavior to loiter in two separate regions. In (b) the target vehicle is deployed and has just entered the sensor region where it begins to transmit its position data to the track vehicle. The track vehicle has just activated its ArrayTurn behavior for determining which side of the sensor array the target is on. In (c) the track vehicle has just sufficiently resolved the left-right ambiguity and has begun transmitting track solutions to the classify vehicle. The classify vehicle has begun its CloseRange behavior to facilitate classification of the target. The track vehicle has activated its CloseRange and ArrayAngle behaviors. In (d) both the track and classify vehicle are dominated by CloseRange behaviors to the target. In (e), the classify vehicle has performed the classification of the target and both vehicles are returning a back to their loiter regions. In (f) both vehicles are back on-station and awaiting any further unknown objects or vehicles to come through its sensor field. The target vehicle has returned to the dock.

Fig. 6-6 depicts the target position estimates produced by the MOOS process pTracker overlaid onto the actual target track. It is readily seen in the figure that the initial estimates were poor due to a small value for N as discussed in section 6.2.4. As the number of observations increases, a convergence of the estimate near to the actual track can be seen. Of special note is the large increase in convergence labeled “Vehicle Turn” in the figure. This is the point at which the sensor vehicle’s CloseRange behavior became active and made a sharp course change between the positions shown in Fig. 5-8(c) and 5-8(d). Some increasing error can be seen in the estimates near the end of the experiment for two primary reasons. First, this highlights the difficulty in trying to use a single bearings-only sensor to track a target of nearly the same or faster speed. In this configuration, the target is ahead of and moving away from the sensor and it is difficult to position the sensor to produce a better FIM as discussed in section 6.2.4. Second, this error is due to a need to further optimize the vehicle behavior parameters to produce a better FIM.

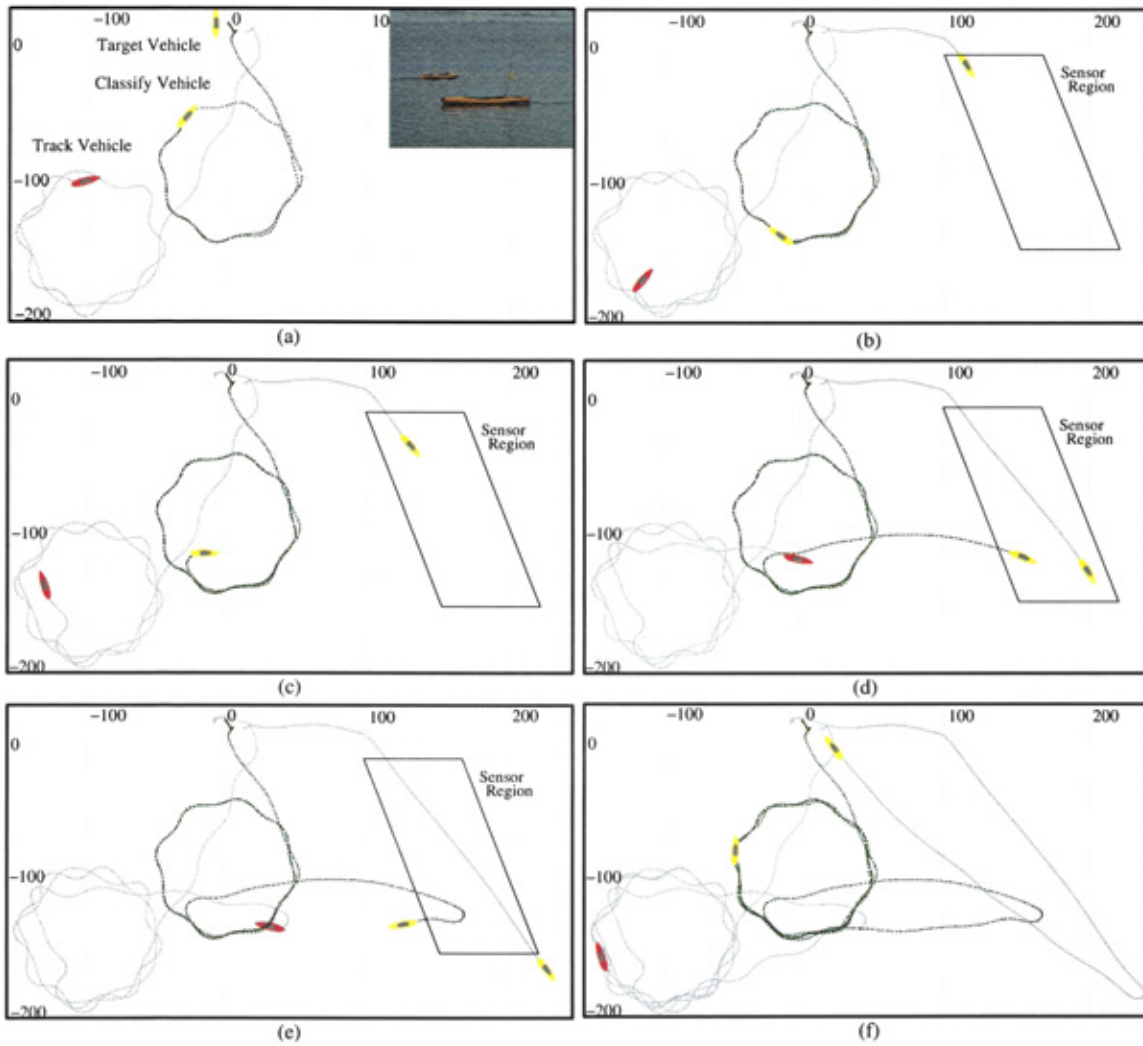


Figure 6-12: In (a) the track vehicle and classify vehicle (both autonomous kayaks, see Fig. 6-4) are deployed and executing their Orbit behavior to loiter in two separate regions. In (b) the target vehicle is deployed and has just entered the sensor region where it begins to transmit its position data to the track vehicle. The track vehicle has just activated its ArrayTurn behavior for determining which side of the sensor array the target is on. In (c) the track vehicle has just sufficiently resolved the left-right ambiguity and has begun transmitting track solutions to the classify vehicle. The classify vehicle has begun its CloseRange behavior to facilitate classification of the target. The track vehicle has activated its CloseRange and ArrayAngle behaviors. In (d) both the track and classify vehicle are dominated by CloseRange behaviors to the target. In (e), the classify vehicle has performed the classification of the target and both vehicles are returning back to their loiter regions. In (f) both vehicles are back on-station and awaiting any further unknown objects or vehicles to come through its sensor field. The target vehicle has returned to the dock.

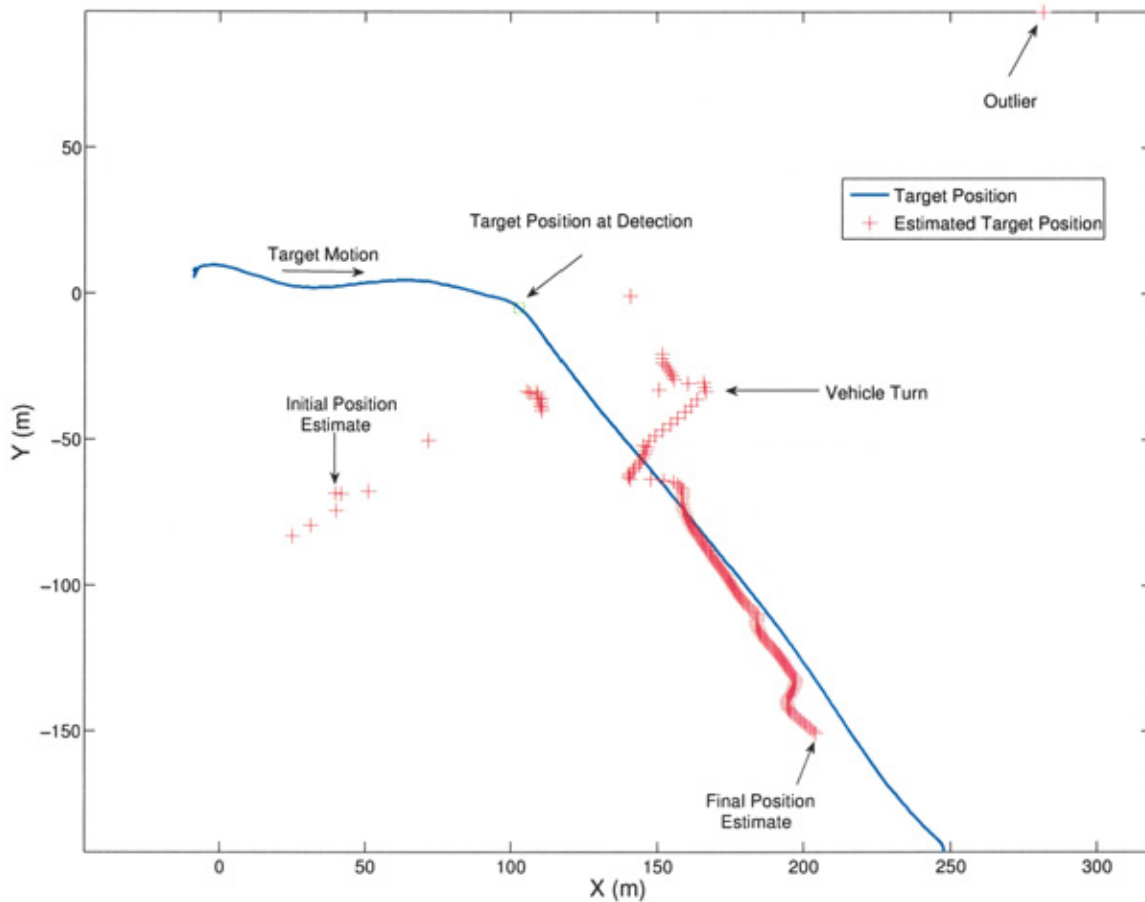


Figure 6-6: Target track solution results. This figure depicts the target position estimates produced by the MOOS process pTracker overlaid onto the actual target track for mission 1507. It is readily seen in the figure that the initial estimates were poor due to a small value for N as discussed in section 6.2.4. As the number of observations increases, a convergence of the estimate near to the actual track can be seen. Of special note is the large increase in convergence labeled “Vehicle Turn” in the figure. This is the point at which the sensor vehicle’s CloseRange behavior became active and made a sharp course change between the positions shown in Fig. 5-8(c) and 5-8(d). Some bias can be seen in the estimates near the end of the experiment due to a need to further optimize the vehicle behavior parameters to produce a better FIM as discussed in section 6.2.4

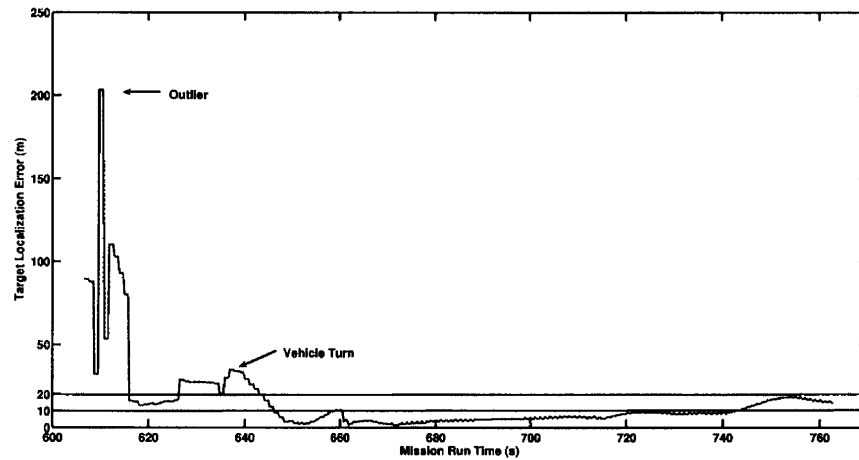


Figure 6-7: Target localization error. This figure shows the error between the target position estimates and the actual target location as a function of mission run time for mission 1507.

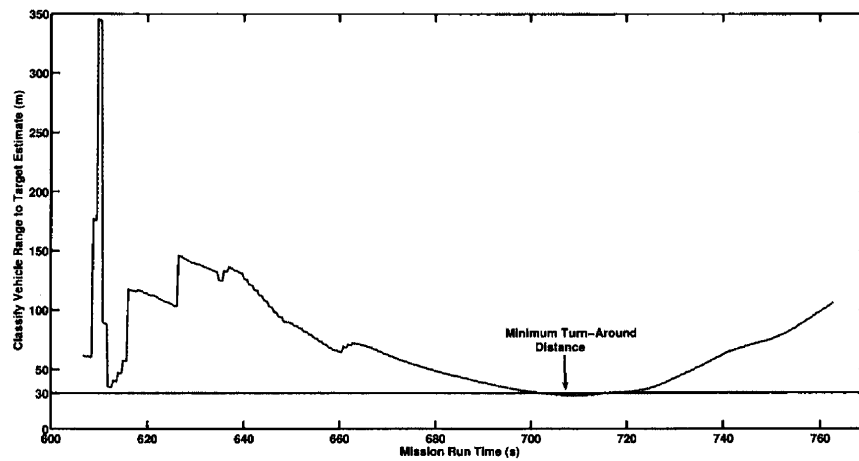


Figure 6-8: Classification vehicle distance from target position estimate. This figure shows the distance between the position of the classification vehicle and the estimated target position as a function of mission run time for mission 1507. The classification vehicle steadily closes range with the target until it reaches its predetermined minimum turnaround distance.

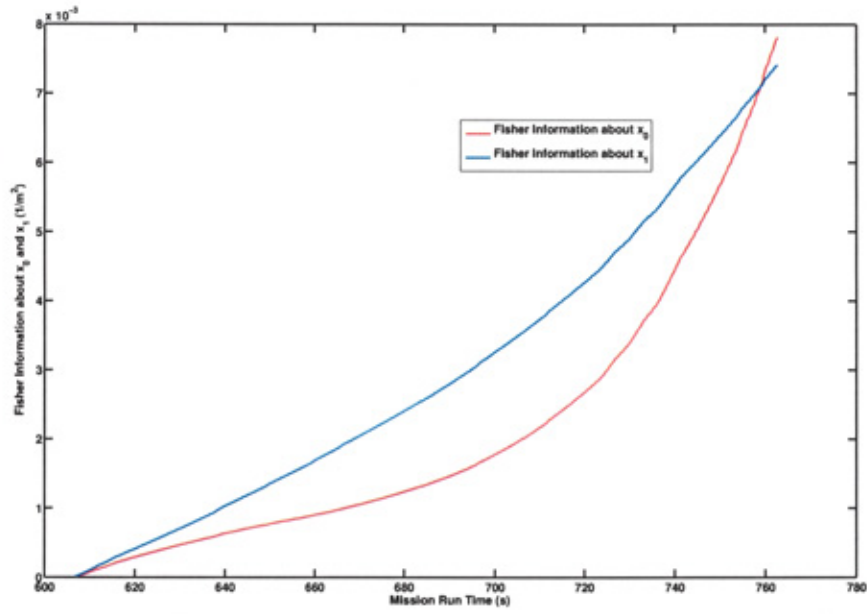


Figure 6-9: Fisher information. This figure shows Fisher information value for state parameters x_0 and x_1 for mission 1507. The Fisher information steadily increase as the number of observations increases and the vehicle closes range with the target.

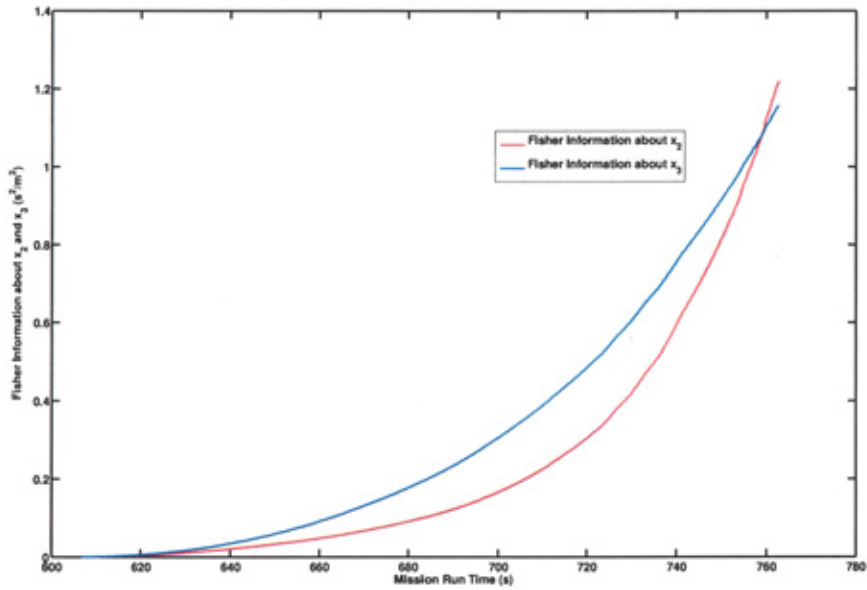


Figure 6-10: Fisher information. This figure shows Fisher information value for state parameters x_2 and x_3 for mission 1507. The Fisher information steadily increase as the number of observations increase and the vehicle closes range with the target.

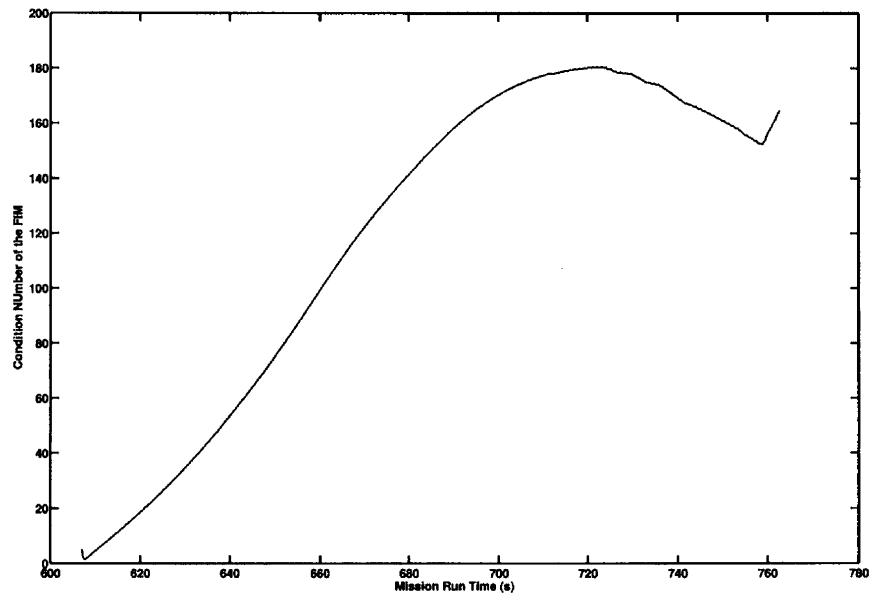


Figure 6-11: Condition number of the Fisher information matrix. This figure shows the condition number of the Fisher information matrix as a function of mission run time for mission 1507. The condition number of the FIM initially increases as the vehicle's CloseRange behavior dominates the vehicle motion. As the vehicle closes with the target, the ArrayAngle behavior becomes more dominant and the vehicle motion starts to reduce the condition number of the FIM.

6.4.2 Mission 1444

Fig. 6-12 shows the vehicle motion for an experimental track and classify mission with autonomous kayaks (see Fig. 6-4) with one tracking kayak, one classify kayak, and one target kayak. The objective of this mission is for the tracking vehicle to acquire and track the target vehicle while relaying target track solutions to the classify vehicle which then executes a simulated classification run.

In (a) the track vehicle and classify vehicle (both autonomous kayaks, see Fig. 6-4) are deployed and executing their Orbit behavior to loiter in two separate regions. In (b) the target vehicle is deployed and has just entered the sensor region where it begins to transmit its position data to the track vehicle. The track vehicle has just activated its ArrayTurn behavior for determining which side of the sensor array the target is on. In (c) the track vehicle has just sufficiently resolved the left-right ambiguity and has begun transmitting track solutions to the classify vehicle. The classify vehicle has begun its CloseRange behavior to facilitate classification of the target. The track vehicle has activated its CloseRange and ArrayAngle behaviors. In (d) both the track and classify vehicle are dominated by CloseRange behaviors to the target. In (e), the classify vehicle has performed the classification of the target and both vehicles are returning back to their loiter regions. In (f) both vehicles are back on-station and awaiting any further unknown objects or vehicles to come through its sensor field. The target vehicle has returned to the dock.

Fig. 6-13 depicts the target position estimates produced by the MOOS process pTracker overlaid onto the actual target track. It is readily seen in the figure that the initial estimates were poor due to a small value for N as discussed in section 6.2.4. As the number of observations increases, a convergence of the estimate near to the actual track can be seen. Some bias can be seen in the estimates near the end of the experiment due to a need to further optimize the vehicle behavior parameters to produce a better FIM as discussed in section 6.2.4.

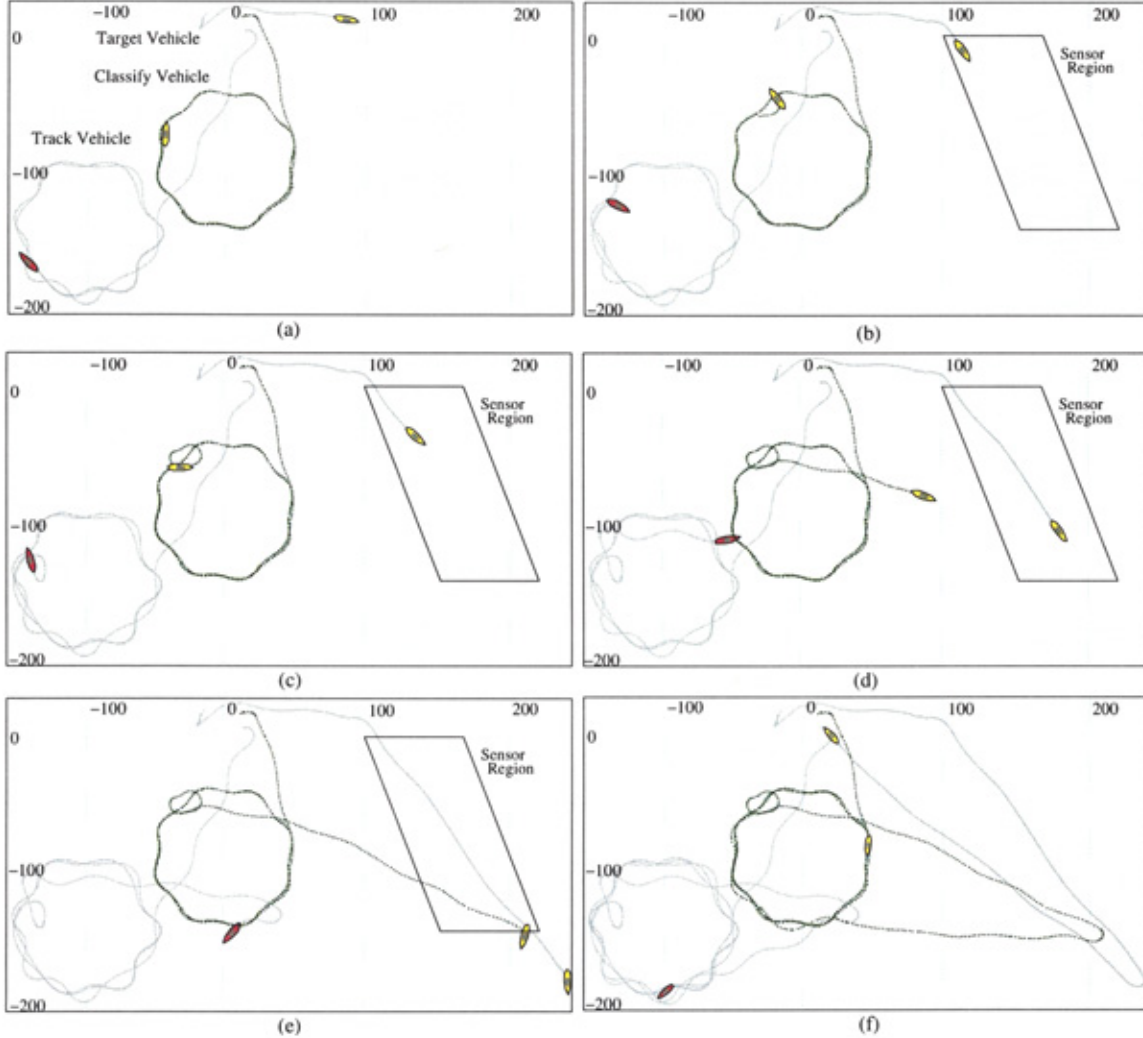


Figure 6-12: In (a) the track vehicle and classify vehicle (both autonomous kayaks, see Fig. 6-4) are deployed and executing their Orbit behavior to loiter in two separate regions. In (b) the target vehicle is deployed and has just entered the sensor region where it begins to transmit its position data to the track vehicle. The track vehicle has just activated its ArrayTurn behavior for determining which side of the sensor array the target is on. In (c) the track vehicle has just sufficiently resolved the left-right ambiguity and has begun transmitting track solutions to the classify vehicle. The classify vehicle has begun its CloseRange behavior to facilitate classification of the target. The track vehicle has activated its CloseRange and ArrayAngle behaviors. In (d) both the track and classify vehicle are dominated by CloseRange behaviors to the target. In (e), the classify vehicle has performed the classification of the target and both vehicles are returning back to their loiter regions. In (f) both vehicles are back on-station and awaiting any further unknown objects or vehicles to come through its sensor field. The target vehicle has returned to the dock.

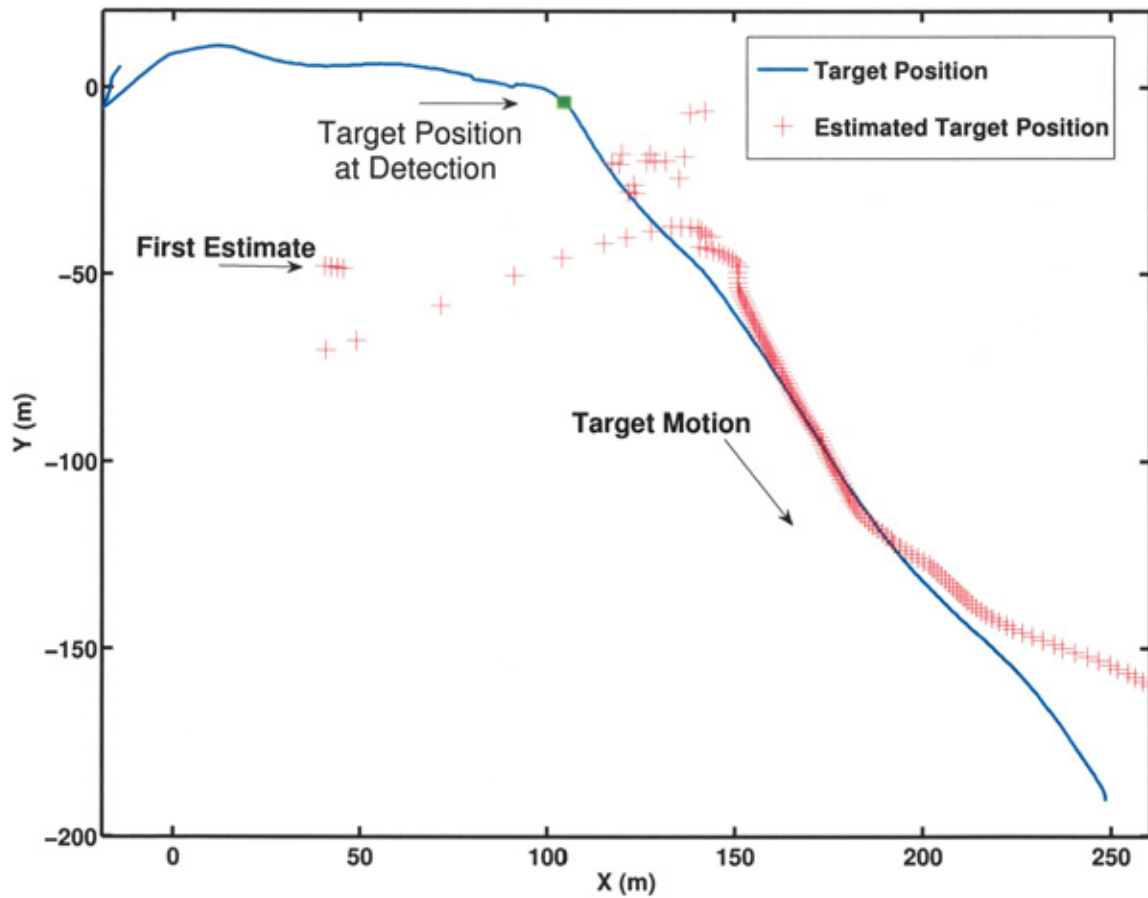


Figure 6-13: Target track solution results. This figure depicts the target position estimates produced by the MOOS process pTracker overlaid onto the actual target track for mission 1444. It is readily seen in the figure that the initial estimates were poor due to a small value for N as discussed in section 6.2.4. As the number of observations increases, a convergence of the estimate near to the actual track can be seen. Some bias can be seen in the estimates near the end of the experiment due to a need to further optimize the vehicle behavior parameters to produce a better FIM as discussed in section 6.2.4

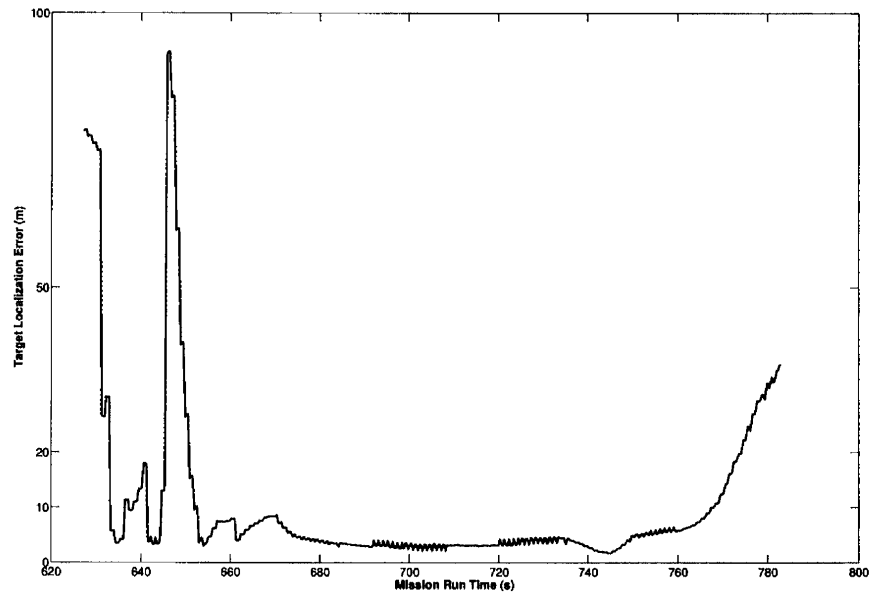


Figure 6-14: Target localization error. This figure shows the error between the target position estimates and the actual target location as a function of mission run time for mission 1444.

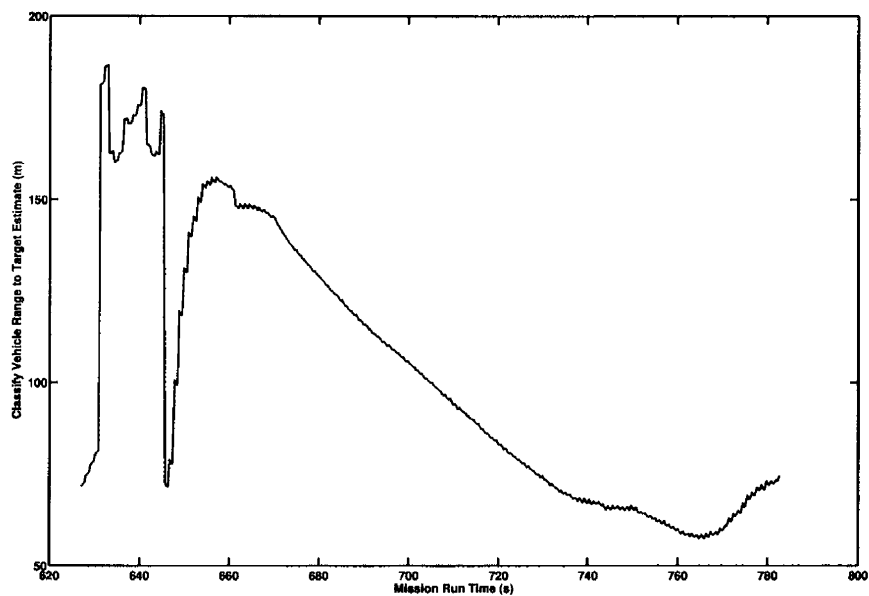


Figure 6-15: Classification vehicle distance from target position estimate. This figure shows the distance between the position of the classification vehicle and the estimated target position as a function of mission run time for mission 1444. The classification vehicle steadily closes range with the target until it reaches its predetermined minimum turnaround distance.

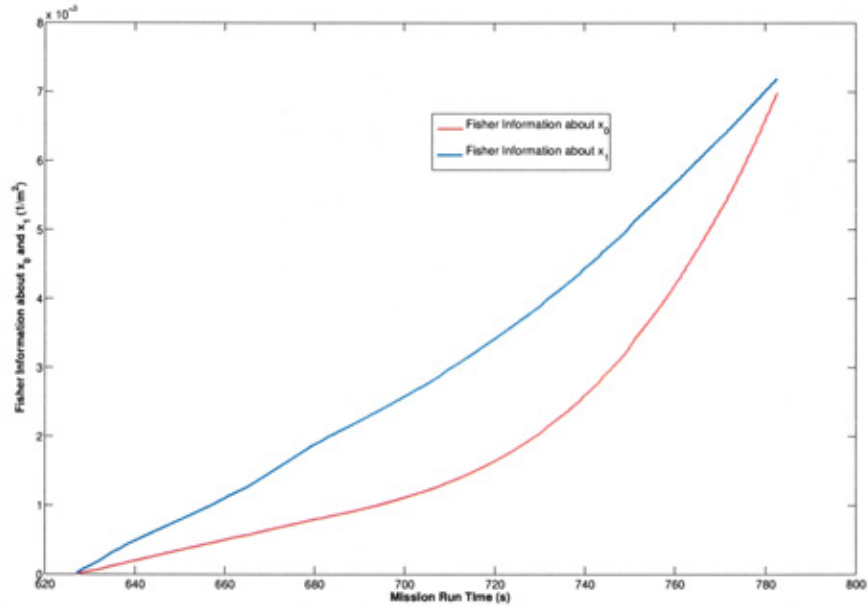


Figure 6-16: Fisher information. This figure shows Fisher information value for state parameters x_0 and x_1 for mission 1444. The Fisher information steadily increase as the number of observations increases and the vehicle closes range with the target.

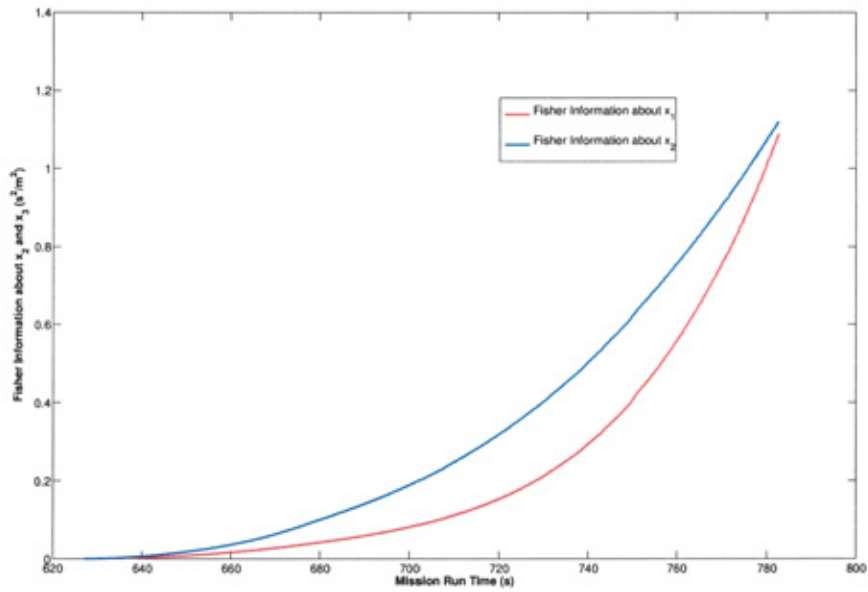


Figure 6-17: Fisher information. This figure shows Fisher information value for state parameters x_2 and x_3 for mission 1444. The Fisher information steadily increase as the number of observations increases and the vehicle closes range with the target.

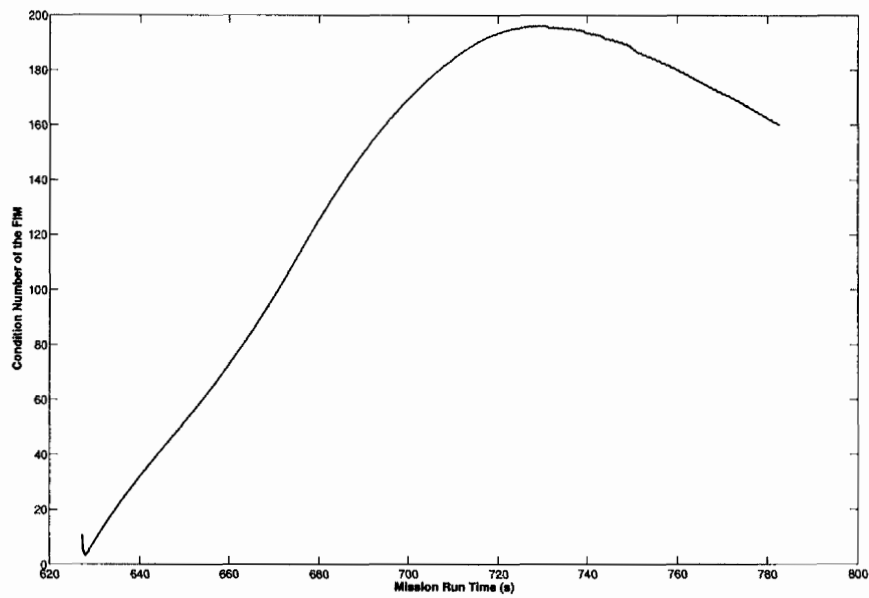


Figure 6-18: Condition number of the Fisher information matrix. This figure shows the condition number of the Fisher information matrix as a function of mission run time for mission 1444. The condition number of the FIM initially increases as the vehicle's CloseRange behavior dominates the vehicle motion. As the vehicle closes with the target, the ArrayAngle behavior becomes more dominant and the vehicle motion starts to reduce the condition number of the FIM.

6.4.3 Mission 1422

Fig. 6-19 shows the vehicle motion for an experimental track and classify mission with autonomous kayaks (see Fig. 6-4) with one tracking kayak, one classify kayak, and one target kayak. The objective of this mission is for the tracking vehicle to acquire and track the target vehicle while relaying target track solutions to the classify vehicle which then executes a simulated classification run.

In (a) the track vehicle and classify vehicle (both autonomous kayaks, see Fig. 6-4) are deployed and executing their Orbit behavior to loiter in two separate regions. In (b) the target vehicle is deployed and has just entered the sensor region where it begins to transmit its position data to the track vehicle. The track vehicle has just activated its ArrayTurn behavior for determining which side of the sensor array the target is on. In (c) the track vehicle has just sufficiently resolved the left-right ambiguity and has begun transmitting track solutions to the classify vehicle. The classify vehicle has begun its CloseRange behavior to facilitate classification of the target. The track vehicle has activated its CloseRange and ArrayAngle behaviors. In (d) both the track and classify vehicle are dominated by CloseRange behaviors to the target. In (e), the classify vehicle has performed the classification of the target and both vehicles are returning back to their loiter regions. In (f) both vehicles are back on-station and awaiting any further unknown objects or vehicles to come through its sensor field. The target vehicle has returned to the dock.

Fig. 6-20 depicts the target position estimates produced by the MOOS process pTracker overlaid onto the actual target track. It is readily seen in the figure that the initial estimates were poor due to a small value for N as discussed in section 6.2.4. As the number of observations increases, a convergence of the estimate near to the actual track can be seen. Of special note is the large increase in convergence labeled “Vehicle Turn” in the figure. This is the point at which the sensor vehicle’s CloseRange behavior became active and made a sharp course change between the positions shown in Fig. 4(c) and 4(d). Some bias can be seen in the estimates near the end of the experiment due to a need to further optimize the vehicle behavior parameters to produce a better FIM as discussed in section 6.2.4.

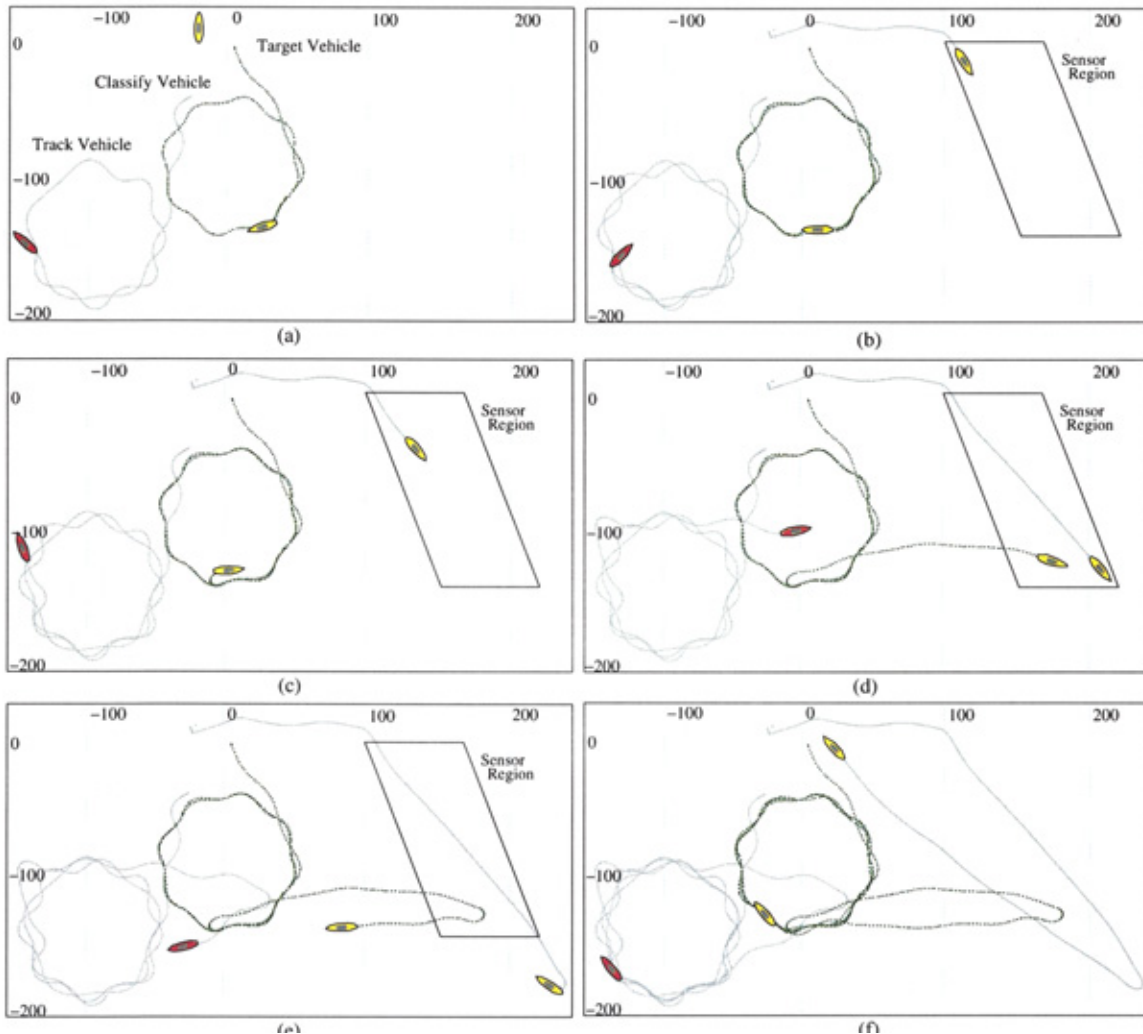


Figure 6-19: In (a) the track vehicle and classify vehicle (both autonomous kayaks, see Fig. 6-4) are deployed and executing their Orbit behavior to loiter in two separate regions. In (b) the target vehicle is deployed and has just entered the sensor region where it begins to transmit its position data to the track vehicle. The track vehicle has just activated its ArrayTurn behavior for determining which side of the sensor array the target is on. In (c) the track vehicle has just sufficiently resolved the left-right ambiguity and has begun transmitting track solutions to the classify vehicle. The classify vehicle has begun its CloseRange behavior to facilitate classification of the target. The track vehicle has activated its CloseRange and ArrayAngle behaviors. In (d) both the track and classify vehicle are dominated by CloseRange behaviors to the target. In (e), the classify vehicle has performed the classification of the target and both vehicles are returning back to their loiter regions. In (f) both vehicles are back on-station and awaiting any further unknown objects or vehicles to come through its sensor field. The target vehicle has returned to the dock.

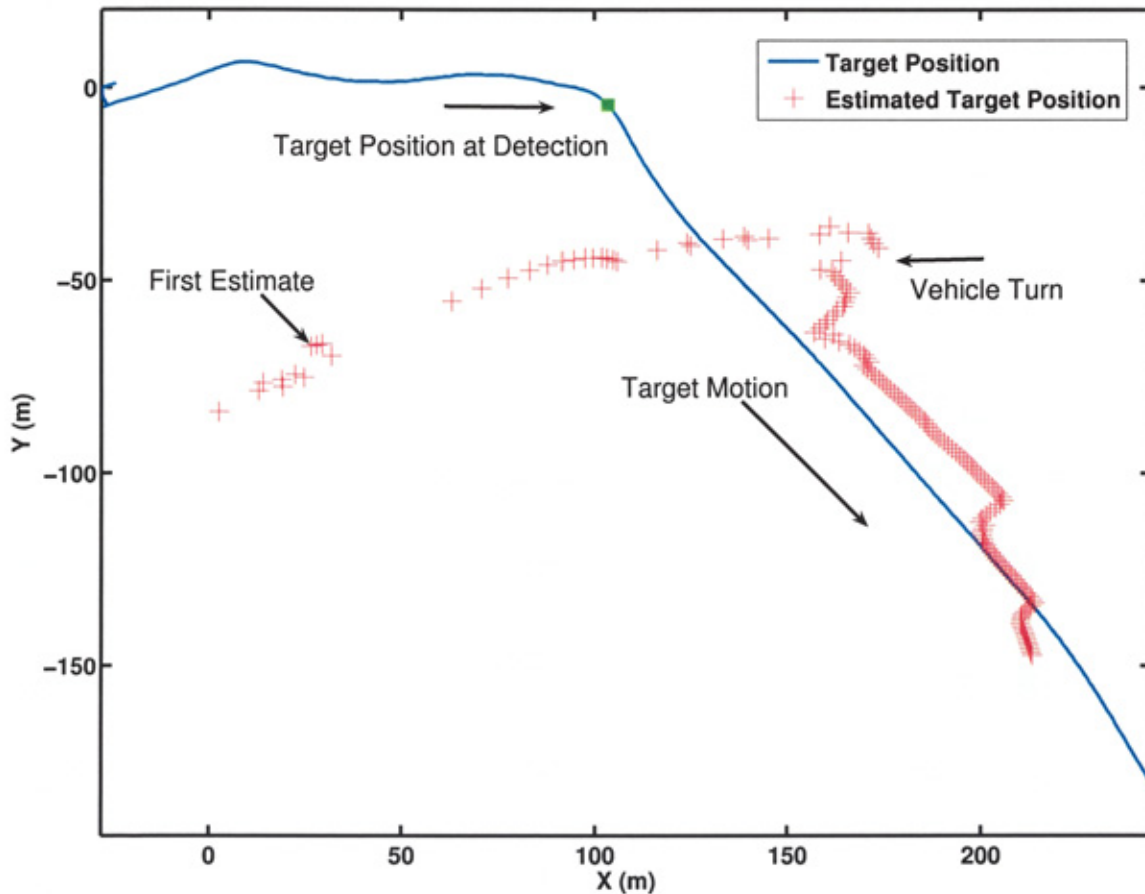


Figure 6-20: Target track solution results. This figure depicts the target position estimates produced by the MOOS process pTracker overlaid onto the actual target track for mission 1422. It is readily seen in the figure that the initial estimates were poor due to a small value for N as discussed in section 6.2.4. As the number of observations increases, a convergence of the estimate near to the actual track can be seen. Of special note is the large increase in convergence labeled “Vehicle Turn” in the figure. This is the point at which the sensor vehicle’s CloseRange behavior became active and made a sharp course change between the positions shown in Fig. 5-22(c) and 5-22(d). Some bias can be seen in the estimates near the end of the experiment due to a need to further optimize the vehicle behavior parameters to produce a better FIM as discussed in section 6.2.4

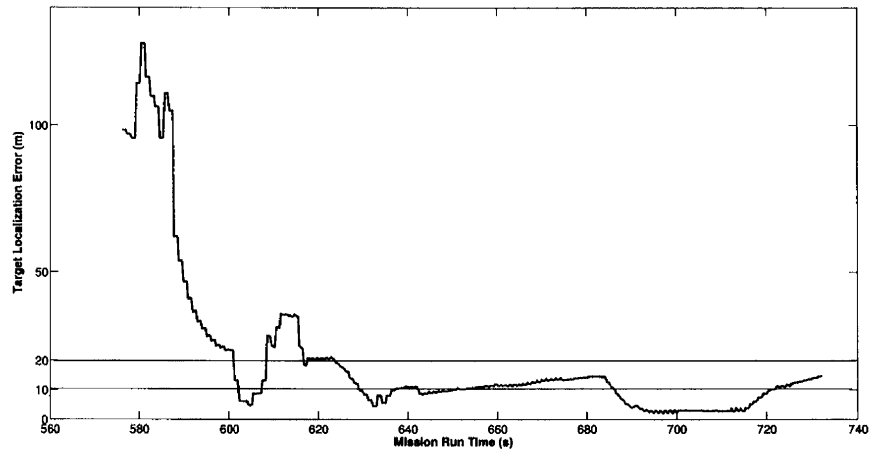


Figure 6-21: Target localization error. This figure shows the error between the target position estimates and the actual target location as a function of mission run time for mission 1422.

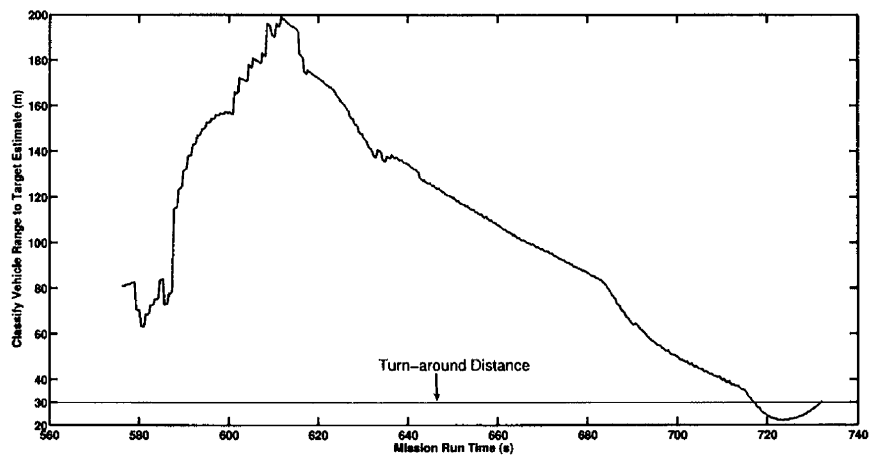


Figure 6-22: Classification vehicle distance from target position estimate. This figure shows the distance between the position of the classification vehicle and the estimated target position as a function of mission run time for mission 1422. The classification vehicle steadily closes range with the target until it reaches its predetermined minimum turnaround distance.

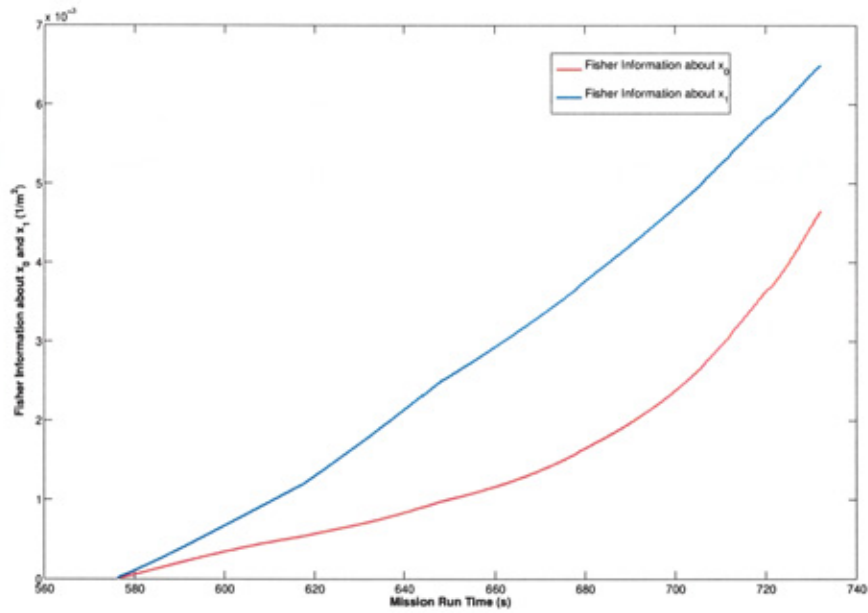


Figure 6-23: Fisher information. This figure shows Fisher information value for state parameters x_0 and x_1 for mission 1422. The Fisher information steadily increase as the number of observations increases and the vehicle closes range with the target.

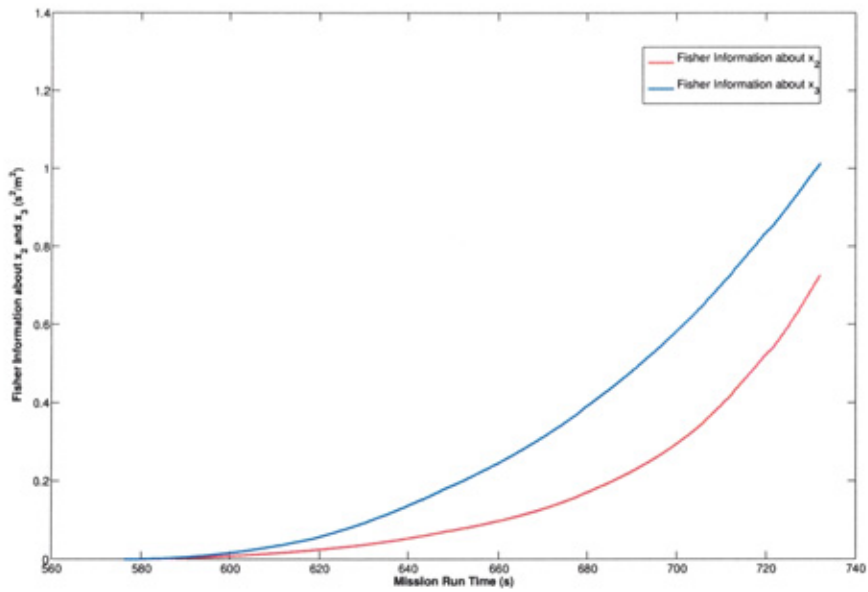


Figure 6-24: Fisher information. This figure shows Fisher information value for state parameters x_2 and x_3 for mission 1422. The Fisher information steadily increase as the number of observations increases and the vehicle closes range with the target.

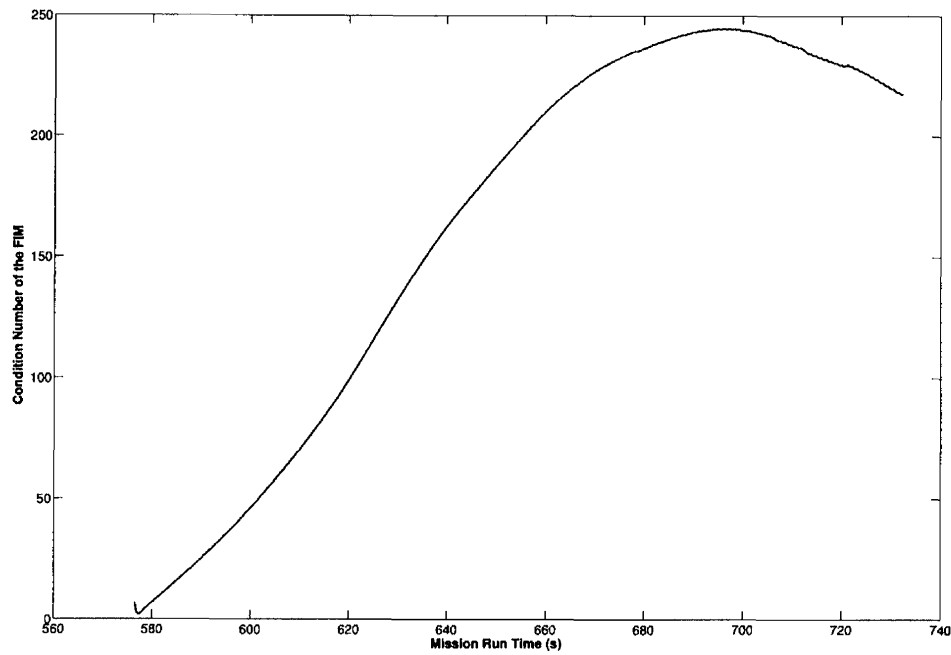


Figure 6-25: Condition number of the Fisher information matrix. This figure shows the condition number of the Fisher information matrix as a function of mission run time for mission 1422. The condition number of the FIM initially increases as the vehicle's CloseRange behavior dominates the vehicle motion. As the vehicle closes with the target, the ArrayAngle behavior becomes more dominant and the vehicle motion starts to reduce the condition number of the FIM.

Chapter 7

Example Two: Adaptive Tracking with Multiple Sensors

7.1 Introduction

We are motivated by the following scenario (see Fig. 7-1: two networked sensor vehicles are in operation, both fitted with passive, towed, acoustic sensor arrays. Both vehicles will detect and cooperatively track moving targets of unknown trajectory and type. Both vehicles begin in

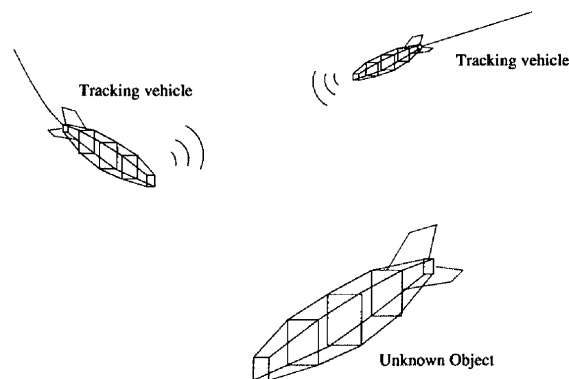


Figure 7-1: Two unmanned marine vehicles are in operation together in a marine sensor network. Both vehicles use linear towed arrays to produce simultaneous bearing estimates to an acoustic target which are combined to form a target track estimate. The track estimate is used to maneuver both vehicles into a formation designed to minimize the target track estimate uncertainty

patrol mode in separate portions of the operating area in order to optimize their sensor coverage.

The two vehicles work together to track underwater objects by communicating target bearing and track estimate information between themselves via acoustic modem. The vehicles will then position themselves with respect to the target in a track and trail formation designed to minimize the uncertainty in the target track estimate.

In this chapter, we will follow a similar technical approach to that followed in Chapter 6 for target tracking with a single sensor platform. First we will derive the mathematical basis for target tracking with two distributed bearing sensors which will allow us to design the proper behaviors for vehicle motion. Next, we will derive the target localization and tracking algorithms which reside on the intelligent sensor. Finally, we will present experimental validation of these concepts using three autonomous surface craft. By comparing the tracking results obtained with two distributed sensor platforms with those obtained in Chapter 6 using a single sensor platform, it will be clear that spatially distributed sensors have the potential to offer significant advantages.

7.2 Bearings-Only Target Tracking with Two Sensors

In order to track a moving object from a set of discrete sensor observations, one must first decide on the kinematic model used to describe the object's motion. In this work, a constant-velocity model was chosen because it is one of the simplest to describe mathematically and because estimating the motion of a constant velocity target using a bearings-only sensor is a classical problem in target motion analysis. Also termed "passive localization" or "passive ranging" this problem arises, for example, when trying to estimate the motion of a submarine moving at constant velocity from another submarine observing the target using a linear towed array sensor. In typical passive ranging applications, however, the state parameters for the target track are estimated using a set of observations from a single moving sensor platform. With only one sensor, both temporal and spatial diversity in the sensor measurements are needed to estimate the target track. In this work, we will estimate the target track parameters using simultaneous measurements from two spatially distributed sensors from which an immediate solution of the target position can be formed. Successive position estimates will then be used to estimate the target's velocity components.

7.2.1 2D Target Position Triangulation

Triangulating the position of an object using passive angle measurements is common in a number of fields including optics. Most analysis, however, assume fixed sensors triangulating fixed or moving targets or moving sensors estimating the position of a fixed target [63]. In this work we now consider the position estimation for a moving target from a moving sensor platform. In this section, we will follow the analysis as developed in [63] for the 2D target position estimation and the subsequent error analysis. Given the coordinate frame shown in Fig. 7-2 with target location

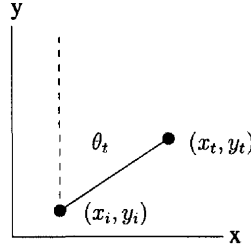


Figure 7-2: Coordinate frame for 2D multi-sensor tracking.

$(x_t[n], y_t[n])$ and sensor positions $(x_i[n], y_i[n])$ for the discrete time interval $n = 0, 1, \dots, N$, the relationship between the position of the i^{th} sensor and its measured target bearing θ_i at time n is given by

$$\tan \theta_i[n] = \frac{x_t[n] - x_i[n]}{y_t[n] - y_i[n]} \quad (7.1)$$

The solution to (7.1) for the general case of I sensors can be written in matrix form as

$$\begin{bmatrix} \cdot \\ x_i[n] - y_i[n] \tan \theta_i \\ \cdot \end{bmatrix} = \begin{bmatrix} \cdot & \cdot \\ 1 & -\tan \theta_i[n] \\ \cdot & \cdot \end{bmatrix} \begin{bmatrix} \hat{x}_t[n] \\ \hat{y}_t[n] \end{bmatrix} \quad (7.2)$$

This system of nonlinear equations can be solved using general least-squares methods such as Gauss-Newton and Levenberg-Marquardt. For the problem under consideration in this work, we limit ourself to the case of two sensors for which the exact solution at any time step n can be written as

$$\hat{x}_t = \frac{x_2 \tan \theta_1 - x_1 \tan \theta_2 + (y_1 - y_2) \tan \theta_1 \tan \theta_2}{\tan \theta_1 - \tan \theta_2} \quad (7.3)$$

$$\hat{y}_t = \frac{y_1 \tan \theta_1 - y_2 \tan \theta_2 + x_2 - x_1}{\tan \theta_1 - \tan \theta_2} \quad (7.4)$$

7.2.2 Variance of the Target Position Estimate

One of the most important pieces of information needed to develop the proper behaviors for a sensor-adaptive system is the relationship between the target motion and the variance of the parameter estimates for the process under observation. From (7.3) and (7.4) it is apparent that the uncertainty in the target position estimates will be influenced by three factors:

1. The uncertainty of the sensor positions $(x_i[n], y_i[n])$
2. The uncertainty of the bearing measurements $\theta_i[n]$
3. The positions of the sensors with respect to the target

The sensor position uncertainties we model as Gaussian distributions with variance σ_{pos}^2 equal and uncorrelated in both the x and y directions. The bearing measurement uncertainties we also model as Gaussian distributions with variance σ_θ^2 equal and independent of sensor platform. The usual method for finding the variances of (7.3) and (7.4) would be to take the expectation

$$var(\hat{x}) = E[(\hat{x} - x)^2] \quad (7.5)$$

Given the complexity of the functional forms for (7.3) and (7.4) however, no closed form solution for (7.5) can be calculated. In this case, one can derive the error propagation equations by performing Taylor series expansions of (7.3) and (7.4) as given in detail for this application in [63]. Using the above assumptions with regards to the uncertainties for sensor position and bearing measurements, a first-order approximation to the target position uncertainties can be given as

$$\sigma_{x_t}^2 \approx C_1 \sigma_{pos}^2 + C_2 \sigma_\theta^2 \quad (7.6)$$

$$\sigma_{y_t}^2 \approx C_3 \sigma_{pos}^2 + C_4 \sigma_\theta^2 \quad (7.7)$$

where C_1 , C_2 , C_3 , and C_4 are coefficients given as

$$C_1 = \left(\frac{\partial x_t}{\partial x_1}\right)^2 + \left(\frac{\partial x_t}{\partial x_2}\right)^2 + \left(\frac{\partial x_t}{\partial y_1}\right)^2 + \left(\frac{\partial x_t}{\partial y_2}\right)^2 \quad (7.8)$$

$$C_2 = \left(\frac{\partial x_t}{\partial \theta_1}\right)^2 + \left(\frac{\partial x_t}{\partial \theta_2}\right)^2 \quad (7.9)$$

$$C_3 = \left(\frac{\partial y_t}{\partial x_1}\right)^2 + \left(\frac{\partial y_t}{\partial x_2}\right)^2 + \left(\frac{\partial y_t}{\partial y_1}\right)^2 + \left(\frac{\partial y_t}{\partial y_2}\right)^2 \quad (7.10)$$

$$C_4 = \left(\frac{\partial y_t}{\partial \theta_1}\right)^2 + \left(\frac{\partial y_t}{\partial \theta_2}\right)^2 \quad (7.11)$$

The derivatives needed to calculate (7.8) through (7.11) are derived in [63] and are also listed in Appendix A. Coefficients C_1 and C_3 measure the contribution of the sensor position error to the target location error while coefficients C_2 and C_4 measure the contribution of the bearing measurement error to the target location error. Coefficients C_1 , C_2 , C_3 , and C_4 are plotted in Fig (7-3) through (7-7). Fig. (7-5) is a plot of coefficient C_2 with a sensor to target range of 20 meters versus the range of 10 meters used in Fig. (7-4). From an analysis of these plots, the following observations can be made with regard to the effect of sensor platform motion on the variance of the target position estimates:

1. The largest influence on $\sigma_{x_t}^2$ and $\sigma_{y_t}^2$ is the sensor separation angle $(\theta_1 - \theta_2)$ with minimum variance at a separation angle of 90 degrees rising to infinity at separation angles of 0 degrees and 180 degrees.
2. The influence of the bearing measurement error rises linearly with the sensor to target range. The bearing measurement error will also rise with the sensor to target range due to the reduction in the received signal to noise ratio when using a real acoustic array.
3. The 90 degree rotation between the plots of the coefficients for the variances of \hat{x}_t and \hat{y}_t indicate that uncertainty in one spatial direction can be minimized with a corresponding increase in uncertainty in the other spatial direction.

These observations will be used in Section 4.2 to develop the autonomous vehicle behaviors designed to cooperatively track a moving target with two sensor platforms with a goal of minimizing the target localization errors subject to other constraints on the platform motion.

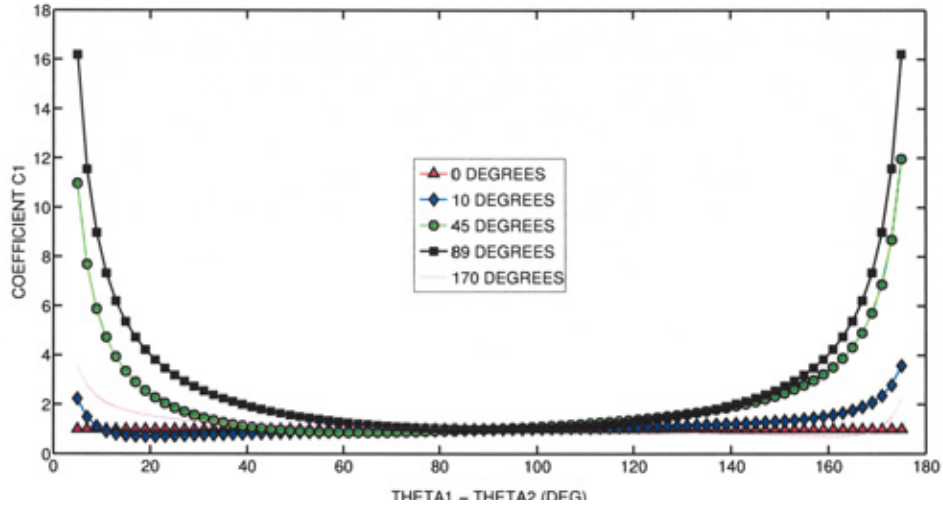


Figure 7-3: Coefficient C_1 . This plot shows shows coefficient C_1 in (7.8) as a function of θ_1 and $(\theta_1 - \theta_2)$. It is clearly seen that C_1 is minimized for $(\theta_1 - \theta_2) = 90$ degrees.

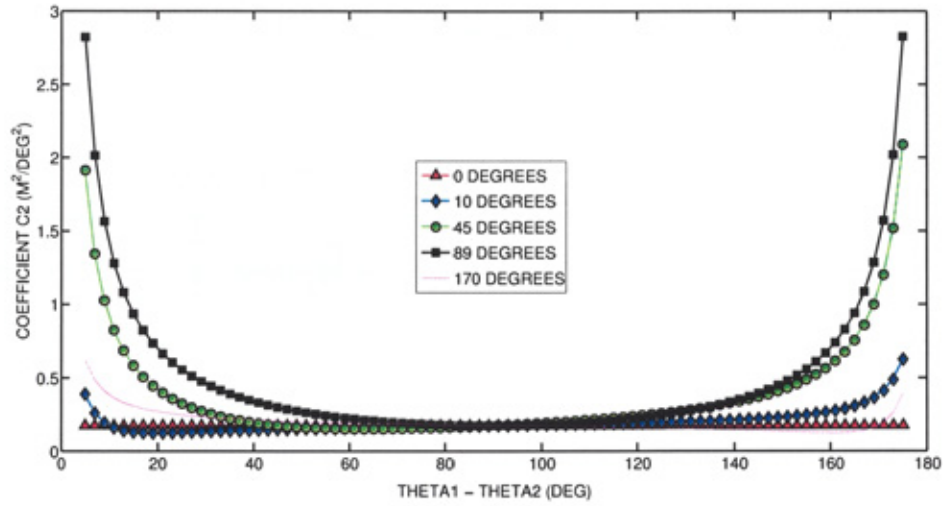


Figure 7-4: Coefficient C_2 (10m). This plot shows shows coefficient C_2 in (7.9) as a function of θ_1 and $(\theta_1 - \theta_2)$ for a sensor to target range of 10 meters. It is clearly seen that C_2 is minimized for $(\theta_1 - \theta_2) = 90$ degrees.

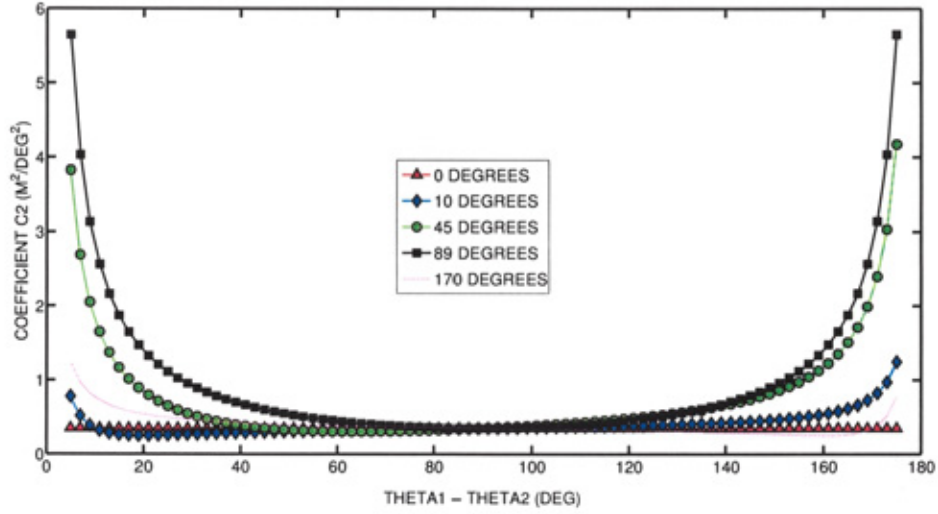


Figure 7-5: Coefficient C_2 (20m). This plot shows shows coefficient C_2 in (7.9) as a function of θ_1 and $(\theta_1 - \theta_2)$ for a sensor to target range of 20 meters. By comparison with Fig. 7-4, it is clearly seen that C_2 is linearly dependent on the sensor to target range.

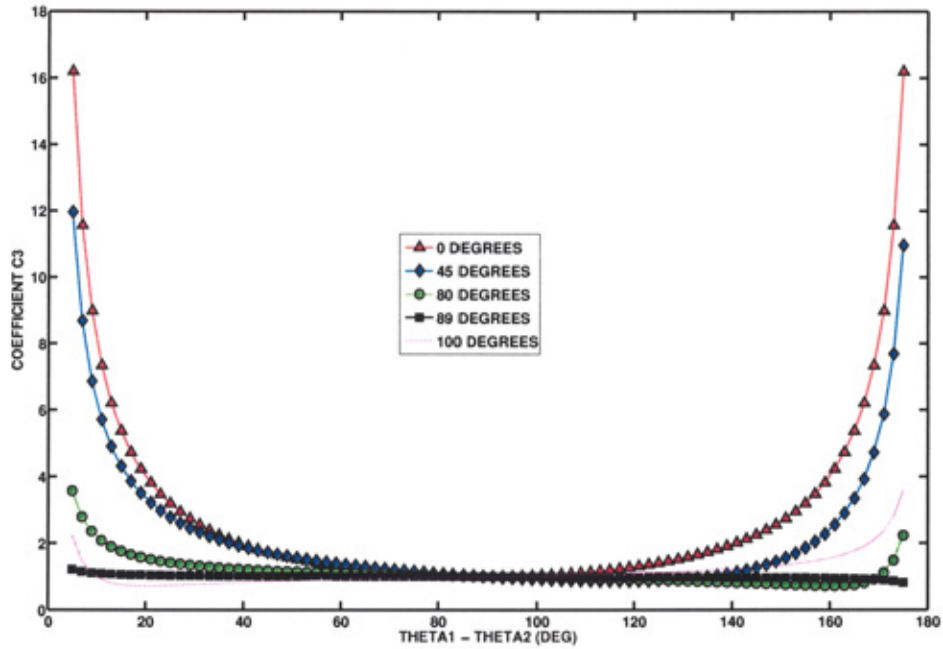


Figure 7-6: Coefficient C_3 (10m). This plot shows shows coefficient C_3 in (7.10) as a function of θ_1 and $(\theta_1 - \theta_2)$ for a sensor to target range of 10 meters. It is clearly seen that C_3 is minimized for $(\theta_1 - \theta_2) = 90$ degrees.

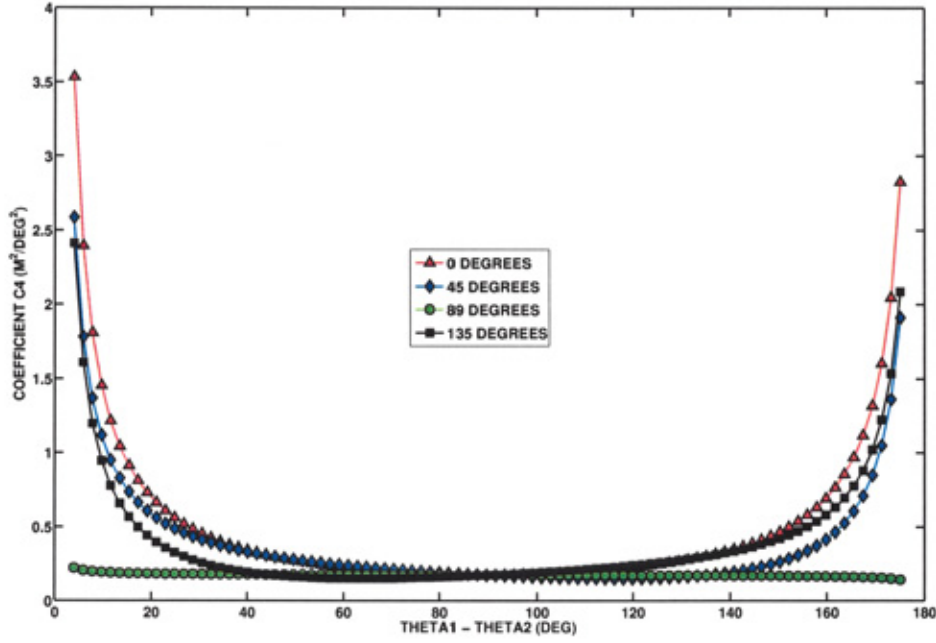


Figure 7-7: Coefficient C_4 (10m). This plot shows shows coefficient C_4 in (7.11) as a function of θ_1 and $(\theta_1 - \theta_2)$ for a sensor to target range of 10 meters. It is clearly seen that C_4 is minimized for $(\theta_1 - \theta_2) = 90$ degrees.

7.2.3 Target Velocity Component Estimation

Having derived the necessary analysis to be able to estimate the instantaneous position of a target from two simultaneous bearing measurements, we would like to filter these noisy measurements as well as estimate the target's velocity components from successive position estimates. A number of techniques are available to do this but the extended Kalman filter was chosen for its speed, with available CPU cycles being limited on small, autonomous platforms. Even though this is a non-optimal estimation technique, good performance was obtained as shown in section 7.3. We start by modeling the target motion with the discrete time state equation

$$\mathbf{x}_{k+1} = \mathbf{F}_k \mathbf{x}_k + \mathbf{w}_k \quad (7.12)$$

where \mathbf{x}_k is the state vector for the target motion given by

$$\mathbf{x}_k \triangleq [\dot{x}_t \ x_t \ \dot{y}_t \ y_t]_k^T \quad (7.13)$$

with

$$x_{tk} = x_{tk-1} + \dot{x}_t dt \quad y_{tk} = y_{tk-1} + \dot{y}_t dt \quad (7.14)$$

and \mathbf{w}_k the process noise vector given as $[qx \ 0 \ qy \ 0]^T$ where qx and qy are independent and equally distributed, zero mean, Gaussian random variables. Given the assumption of a constant velocity target, the state transition matrix \mathbf{F}_k is computed as the Jacobian of (7.13).

$$\mathbf{F}_k = \begin{bmatrix} 1 & 0 & 0 & 0 \\ dt & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & dt & 1 \end{bmatrix} \quad (7.15)$$

\mathbf{Q}_k is the covariance matrix of the process noise \mathbf{w}_k given by

$$\mathbf{Q}_k = E\{\mathbf{w}_k^2\} = E \left[\int_0^{dt} \mathbf{F}_k \mathbf{w}_k d(\epsilon) \int_0^{dt} (\mathbf{F}_k \mathbf{w}_k)^T d(\eta) \right] \quad (7.16)$$

resulting in

$$\mathbf{Q}_k = qq \begin{bmatrix} dt^2 & \frac{dt^3}{2} & 0 & 0 \\ \frac{dt^3}{2} & \frac{dt^4}{4} & 0 & 0 \\ 0 & 0 & dt^2 & \frac{dt^3}{2} \\ 0 & 0 & \frac{dt^3}{2} & \frac{dt^4}{4} \end{bmatrix} \quad (7.17)$$

where qq is the variance of the process noise. At each time step k , an observation \mathbf{z}_k of \mathbf{x}_k is made according to

$$\mathbf{z}_k = \mathbf{H}_k \mathbf{x}_k + \mathbf{v}_k \quad (7.18)$$

where \mathbf{H}_k is the Jacobian of the observation model

$$\mathbf{h} = \left[\text{atan} \left(\frac{y_t - y_1}{x_t - x_1} \right) \text{atan} \left(\frac{y_t - y_2}{x_t - x_2} \right) \right]^T \quad (7.19)$$

and \mathbf{v}_k is the measurement noise vector given as $r[1 \ 1]^T$ with r a zero mean normally distributed random variable. In this application, \mathbf{z}_k corresponds to a pair of simultaneous bearing measure-

ments $\mathbf{z}_k = [z_1 \ z_2]_k^T$ and the resulting matrix \mathbf{H}_k is

$$\mathbf{H}_k = \begin{bmatrix} 0 & \frac{-(y_t - y_1)}{d_1} & 0 & \frac{(x_t - x_1)}{d_1} \\ 0 & \frac{-(y_t - y_2)}{d_2} & 0 & \frac{(x_t - x_2)}{d_2} \end{bmatrix} \quad (7.20)$$

where d_i is the squared distance from sensor i to the target position estimate given by $(y_t - y_i)^2 + (x_t - x_i)^2$. We consider the measurement noise of the bearing measurements from each sensor platform to be equal and independent of platform, therefore the covariance matrix of the measurement noise \mathbf{v}_k is given as

$$\mathbf{R}_k = rr \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad (7.21)$$

where rr is the variance of our bearing measurements. Given these definitions of our estimation model, our estimation proceeds in classical fashion in two steps. In the first step, we calculate the predicted state $\mathbf{x}_{k|k-1}$ and the predicted state covariance $\mathbf{P}_{k|k-1}$ for the current time step k given the information from the previous time step $k - 1$ as follows:

$$\mathbf{x}_{k|k-1} = \mathbf{F}_k \mathbf{x}_k \quad (7.22)$$

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k|k-1} \mathbf{F}_k^T + \mathbf{Q}_k \quad (7.23)$$

In the second step we refine this prediction using our observations. We proceed by first calculating the measurement residual $\tilde{\mathbf{y}}_k$ and the covariance residual \mathbf{S}_k as

$$\tilde{\mathbf{y}}_k = \mathbf{z}_k - \mathbf{h}_k \quad (7.24)$$

$$\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^T + \mathbf{R}_k \quad (7.25)$$

The Kalman gain is then computed as

$$\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^T \mathbf{S}_k^{-1} \quad (7.26)$$

The Kalman gain is then used with the measurement residual to update the current state estimate $\mathbf{x}_{k|k}$ and the state covariance matrix $\mathbf{P}_{k|k}$ as follows

$$\mathbf{x}_{k|k} = \mathbf{x}_{k|k-1} + \mathbf{K}_k \tilde{\mathbf{y}}_k \quad (7.27)$$

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_{k|k-1} \quad (7.28)$$

7.2.4 Scenario

The experimental scenario begins with the deployment of the two sensor vehicles into separate patrol orbits where they will remain until a target detection occurs. At some point, the target kayak will begin its motion into the target area. When it enters into the target area, it will begin broadcasting its GPS location to the sensor vehicles whose sensor simulators will convert the position information into target bearings. Vehicle two's bearing data will then be transmitted to vehicle one where it will be combined with vehicle one's bearing information to form the target track. The target track information will then be broadcast back to vehicle two and both vehicles will use the track information to position themselves with respect to the target using the formation described in Section 4.2.8. After a predetermined amount of tracking time, tracking will be declared over and the sensor vehicles will return to their patrol orbits to await another target. The target vehicle will return to its starting location.

7.2.5 Behavior Configurations

The sensor vehicles were configured with the following behaviors and preconditions. Details on the individual behaviors is given in section 4.2. A condition is a “variable=value” pair in the MOOS Database. A mission is started by broadcasting “deploy=true” to all vehicles and ended when the “return=true” message is broadcast. A broadcast is over 802.11b and changes a particular MOOS variable in the database resident on the vehicle. The broadcast could also be made via acoustic modem. All vehicle helms were configured with the OpRegion behavior as a safety measure. This behavior is active upon mission startup indicated by “deploy=true”.

The helms on the sensor vehicles were configured with Orbit behaviors which are active immediately upon mission startup indicated by “deploy=true”. The Orbit behavior is conditioned on not receiving bearing sensor data, i.e., “sensor_data=inactive”. It was also configured with the ArrayTurn, ArrayAngle, and CloseRange behaviors described in Section 4.2. These three behaviors are conditioned on the vehicle receiving bearings-only sensor data, indicated by “sen-

sor_data=active” in the MOOS Database.

The target vehicle was configured to follow a simple set of waypoints. Deployment of the target vehicle was done via human command over wireless link when the other two vehicles had been on-station for an arbitrarily sufficient time.

7.2.6 Kalman Filter Initialization

Before the first measurement is processed, the state covariance matrix P_0 , the measurement noise variance rr and the process noise variance qq must be initialized. In all missions described in this work, P_0 , rr , and qq were initialized to the following values:

$$\mathbf{P}_0 = \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 20,000 & 0 & 0 \\ 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 20,000 \end{bmatrix} \quad (7.29)$$

$$rr = 0.01 \text{ rad}^2 \quad qq = 0.002 \text{ m}^2 \quad (7.30)$$

7.3 Experimental Results

7.3.1 Mission 1448

Fig. 7-8 shows the vehicle motion for an experimental tracking mission with autonomous kayaks (see Fig. 6-4) with two tracking vehicles and one target vehicle. The objective of this mission is to execute the scenario described in Section 7.2.4 where two sensor vehicles cooperatively track a target vehicle moving with constant velocity. This mission took place in the Charles River test range on December 1st, 2005. In (a) two tracking vehicles are deployed and executing their Orbit behaviors to patrol in two separate regions. Note that tracking vehicle two, on the right, is exhibiting signs of a rudder control problem. In (b) the target vehicle is deployed and has just entered the sensor region where it begins to transmit its position data to the tracking vehicles for use in the bearing simulators. The tracking vehicles have just activated their ArrayTurn behaviors for determining which side of the sensor array the target is on and the sensor vehicles have begun

their turns. In (c) the tracking vehicles have just sufficiently resolved the left-right ambiguity and have begun executing their Formation behaviors using the target position estimate as a virtual leader. In (d) both the tracking vehicles have moved into formation behind the target to begin the track and trail configuration.. In (e), both sensor vehicles are still in formation trailing the target. In (f) tracking is complete and both vehicles are back on-station and awaiting any further contacts to enter their sensor fields. The target vehicle has returned to the dock.

Fig. 7-9 depicts the target position estimates produced by the MOOS process pTracker overlaid onto the actual target track for the period in which the target vehicle was operating in a constant velocity scenario. As can be seen, excellent position estimates were obtained, especially compared with the tracking results obtained using a single sensor platform to track a constant velocity target as shown in [64]. The gaps in the estimates as seen in the figure were due to communications breaks when no bearing estimates from vehicle two were received by vehicle one. Fig. 7-10 shows the error in the target position estimate as a function of mission run time. As can be seen, even with the communications breaks, position estimation results were generally very good, with an error of approximately 2 *meters* once steady state was reached.

Fig. 7-11 shows the angle between the two sensor vehicles as a function of mission run time. Given the other constraints on the vehicle control, the optimal formation angle was not fully obtained but the angle was well within acceptable limits as shown in Fig. 7-3 and 7-4.

Fig. 7-12 and 7-13 show the sensor vehicle to target estimate ranges. Neither platform was able to fully maintain the programmed distance from the target position estimate although both vehicles were actually closer to the target position estimate at steady state than the programmed distance.

Fig. 7-14 depicts the actual speed of the target versus the speed estimate produced by pTracker for mission 1448. As can be seen, the speed estimate is very good, generally within 0.2 *m/s* of the actual speed of the target by mission time 900. The target speed was obtained from GPS.

Fig. 7-15 depicts the actual heading of the target versus the heading estimate produced by pTracker for mission 1448. As can be seen, the heading estimate is also very good, generally within 5 to 10 *degrees* of the actual heading of the target. The target heading was obtained from GPS.

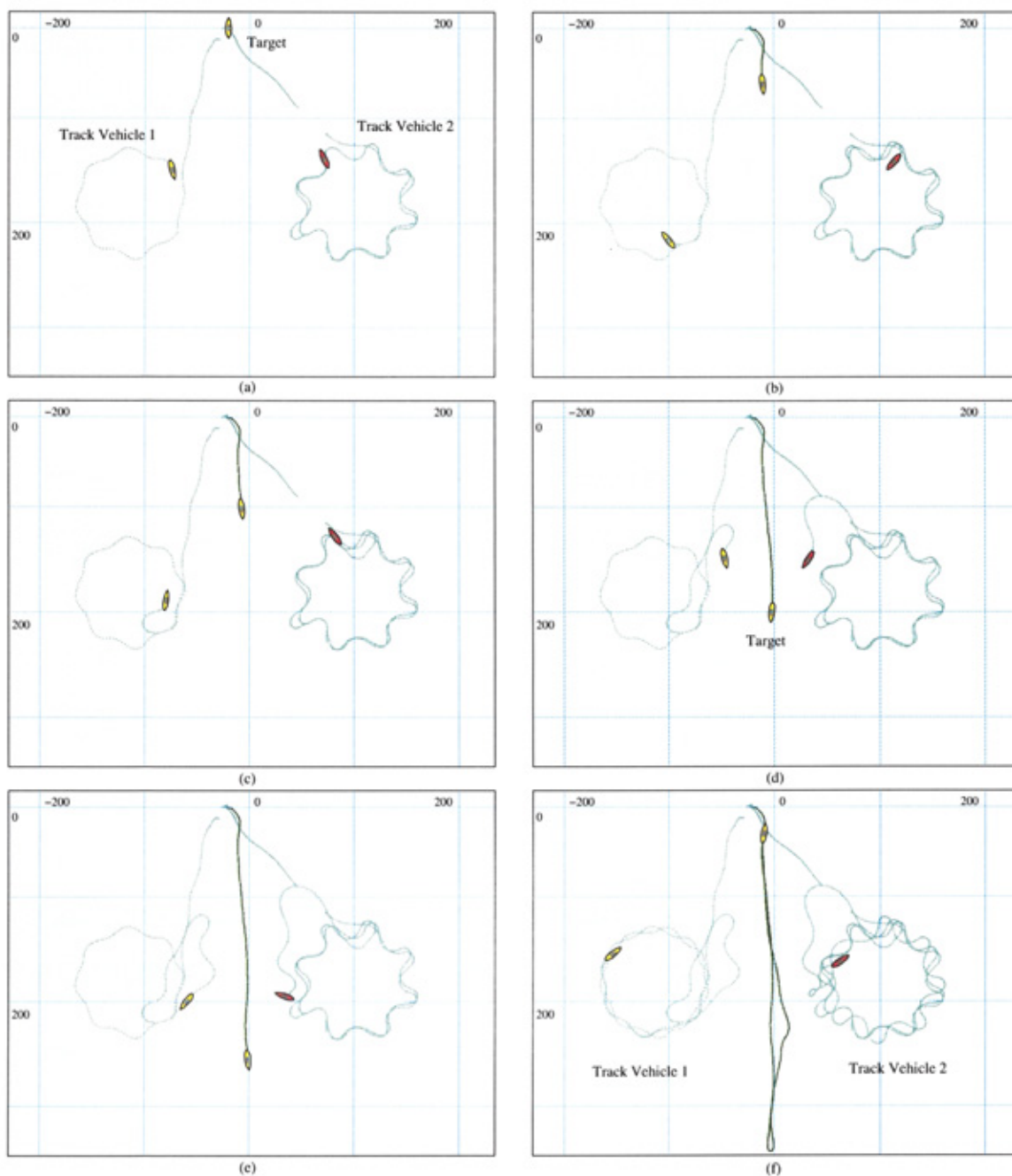


Figure 7-8: This figure shows the vehicle motion for the two sensor vehicles and the target vehicle for Mission 1448. The vehicles are executing the scenario described in Section 7.2.4 using three autonomous kayaks maneuvering within the Charles River test area.

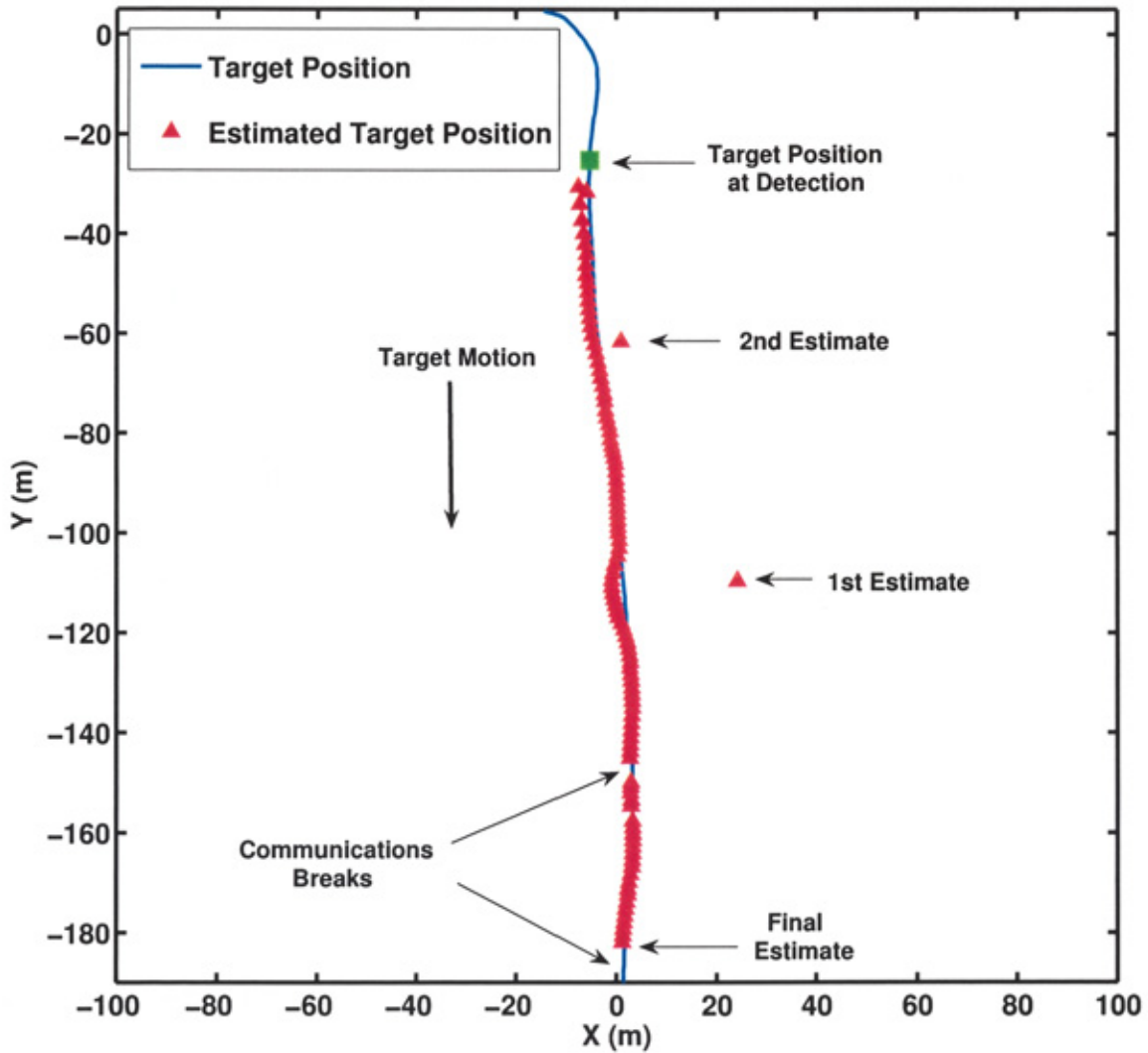


Figure 7-9: Target track solution results. This figure depicts the target position estimates produced by the MOOS process pTracker overlaid onto the actual target track for Mission 1448 for the period in which the target vehicle was operating in a constant velocity scenario. As can be seen, excellent position estimates were obtained, especially compared with the tracking results obtained using a single sensor platform to track a constant velocity target as shown in [64]. The gaps in the estimates as seen in the figure were due to communications breaks when no bearing estimates from vehicle two were received by vehicle one.

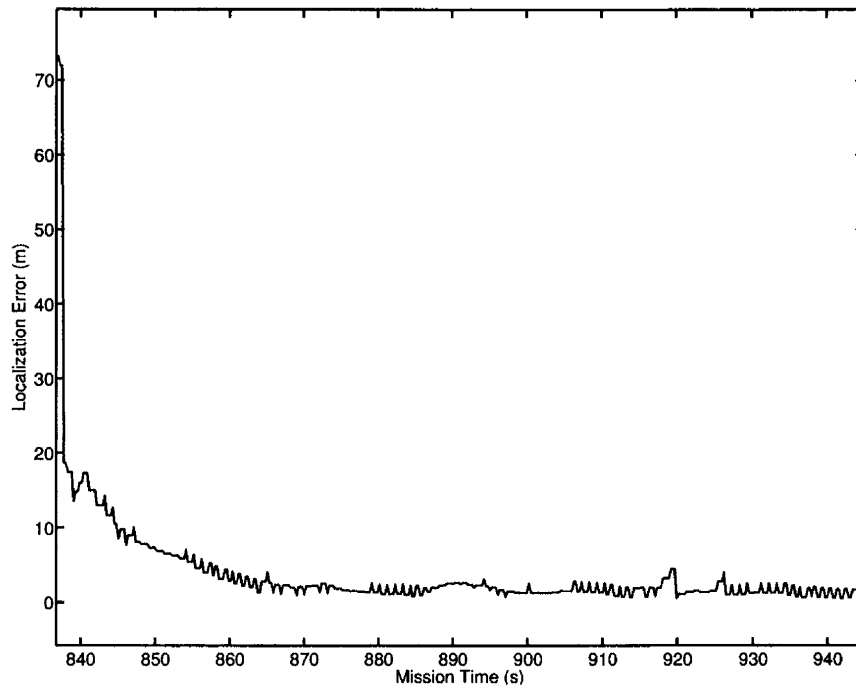


Figure 7-10: Target Track Error. This figure shows the target track error as a function of mission time for Mission 1448. As can be seen, even with the communications breaks, position estimation results were generally very good, with an error of approximately 2 meters once steady state was reached.

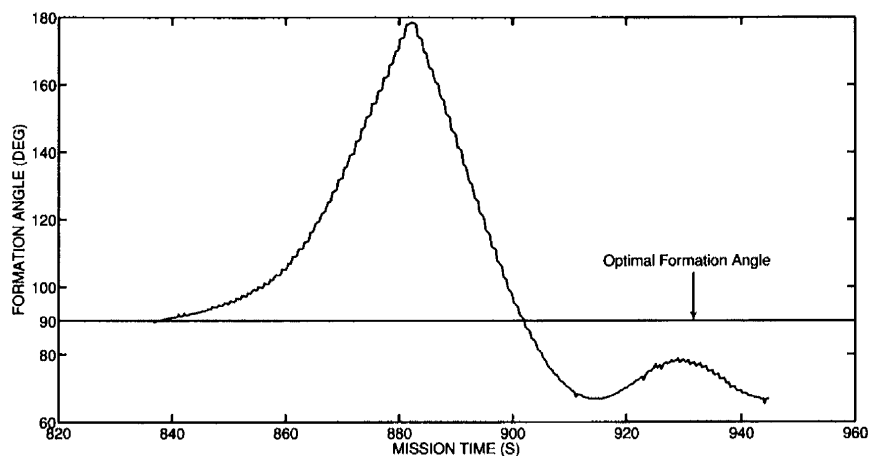


Figure 7-11: Formation angle. This figure shows the formation angle between the two sensor platforms as a function of mission time for mission 1448. Given the other constraints on the vehicle control, the optimal formation angle was not fully obtained but the angle was well within acceptable limits as shown in Fig. 7-3 and 7-4.

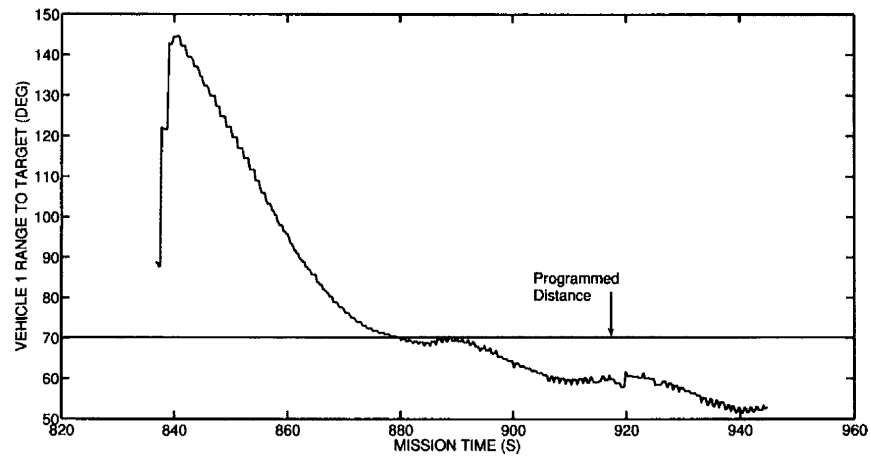


Figure 7-12: Target range for sensor platform one. This figure shows the range between sensor platform one and the target estimate as a function of mission run time for mission 1448.

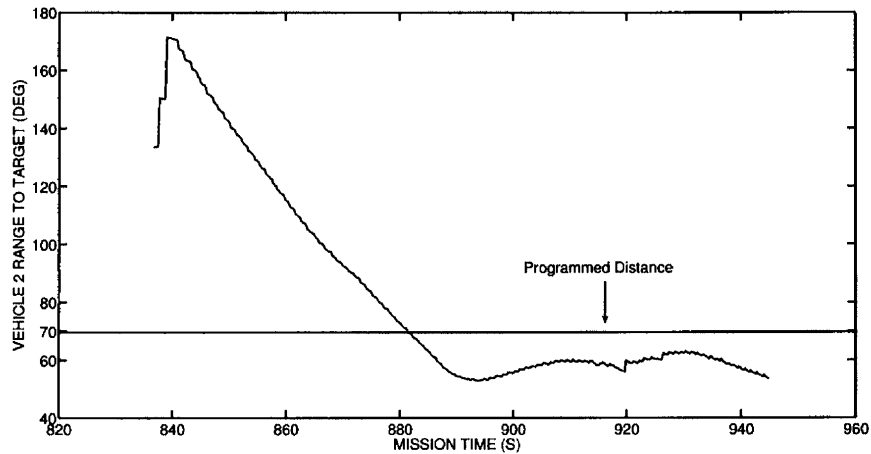


Figure 7-13: Target range for sensor platform two. This figure shows the range between sensor platform two and the target estimate as a function of mission run time for mission 1448.

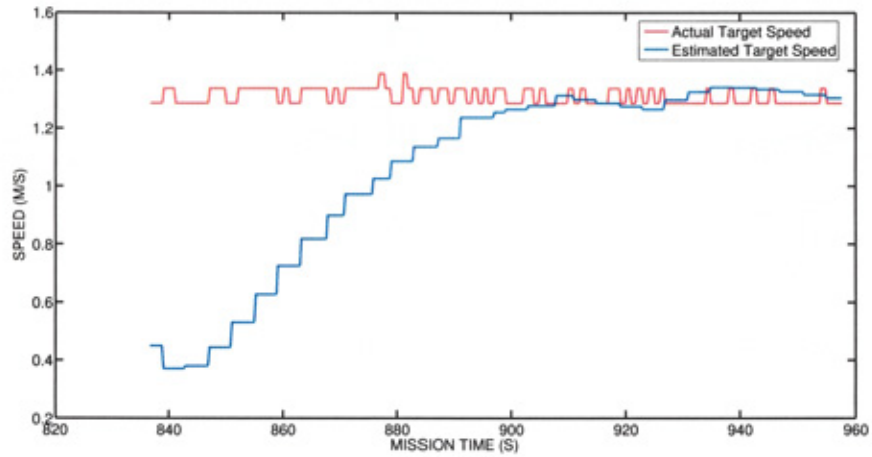


Figure 7-14: Speed estimate error. This figure depicts the actual speed of the target versus the speed estimate produced by pTracker for mission 1448. As can be seen, the speed estimate is very good, generally within 0.2 m/s of the actual speed of the target by mission time 900. The target speed was obtained from GPS.

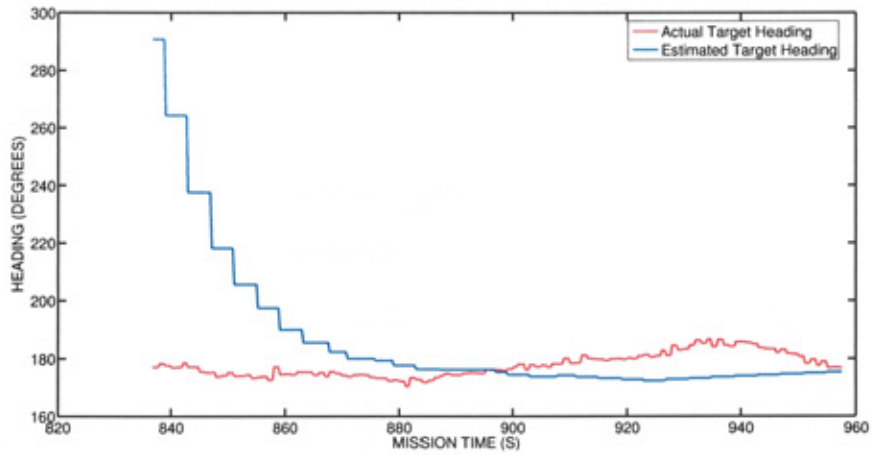


Figure 7-15: Heading estimate error. This figure depicts the actual heading of the target versus the heading estimate produced by pTracker for mission 1448. As can be seen, the heading estimate is very good, generally within $5\text{ to }10\text{ degrees}$ of the actual heading of the target by mission time 900. The target heading was obtained from GPS.

7.3.2 Mission 1144

Fig. 7-16 shows the vehicle motion for an experimental tracking mission with autonomous kayaks (see Fig. 6-4) with two tracking vehicles and one target vehicle. The objective of this mission is to execute the scenario described in Section 7.2.4 where two sensor vehicles cooperatively track a target vehicle moving with constant velocity. This mission took place in the Charles River test range on December 1st, 2005. In (a) two tracking vehicles are deployed and executing their Orbit behaviors to patrol in two separate regions. Note that tracking vehicle two, on the right, is exhibiting signs of a rudder control problem. In (b) the target vehicle is deployed and has just entered the sensor region where it begins to transmit its position data to the tracking vehicles for use in the bearing simulators. The tracking vehicles have just activated their ArrayTurn behaviors for determining which side of the sensor array the target is on and the sensor vehicles have begun their turns. In (c) the tracking vehicles have just sufficiently resolved the left-right ambiguity and have begun executing their Formation behaviors using the target position estimate as a virtual leader. In (d) both the tracking vehicles have moved into formation behind the target to begin the track and trail configuration.. In (e), the target vehicle has turned around before tracking is complete, violating the constant velocity assumption and confusing the sensor vehicles. In (f) tracking is complete and both vehicles are back on-station and awaiting any further contacts to enter their sensor fields. The target vehicle has returned to the dock.

Fig. 7-17 depicts the target position estimates produced by the MOOS process pTracker overlaid onto the actual target track for the period in which the target vehicle was operating in a constant velocity scenario. As can be seen, excellent position estimates were obtained, especially compared with the tracking results obtained using a single sensor platform to track a constant velocity target as shown in [64]. The gaps in the estimates as seen in the figure were due to communications breaks when no bearing estimates from vehicle two were received by vehicle one. Fig. 7-18 shows the error in the target position estimate as a function of mission run time. As can be seen, even with the communications breaks, position estimation results were generally very good, with an error of approximately 2 *meters* once steady state was reached.

Fig. 7-19 shows the angle between the two sensor vehicles as a function of mission run time. Given the other constraints on the vehicle control, the optimal formation angle was not fully obtained but the angle was well within acceptable limits as shown in Fig. 7-3 and 7-4.

Fig. 7-20 and 7-21 show the sensor vehicle to target estimate ranges. Sensor vehicle one

was able to obtain the programmed distance while sensor vehicle two was not, possibly due to problems with the rudder.

Fig. 7-22 depicts the actual speed of the target versus the speed estimate produced by pTracker for mission 1448. As can be seen, the speed estimate is very good, generally within 0.2 m/s of the actual speed of the target. The target speed was obtained from GPS.

Fig. 7-23 depicts the actual heading of the target versus the heading estimate produced by pTracker for mission 1448. As can be seen, the heading estimate is also very good, generally within 5 degrees of the actual heading of the target. The target heading was obtained from GPS.

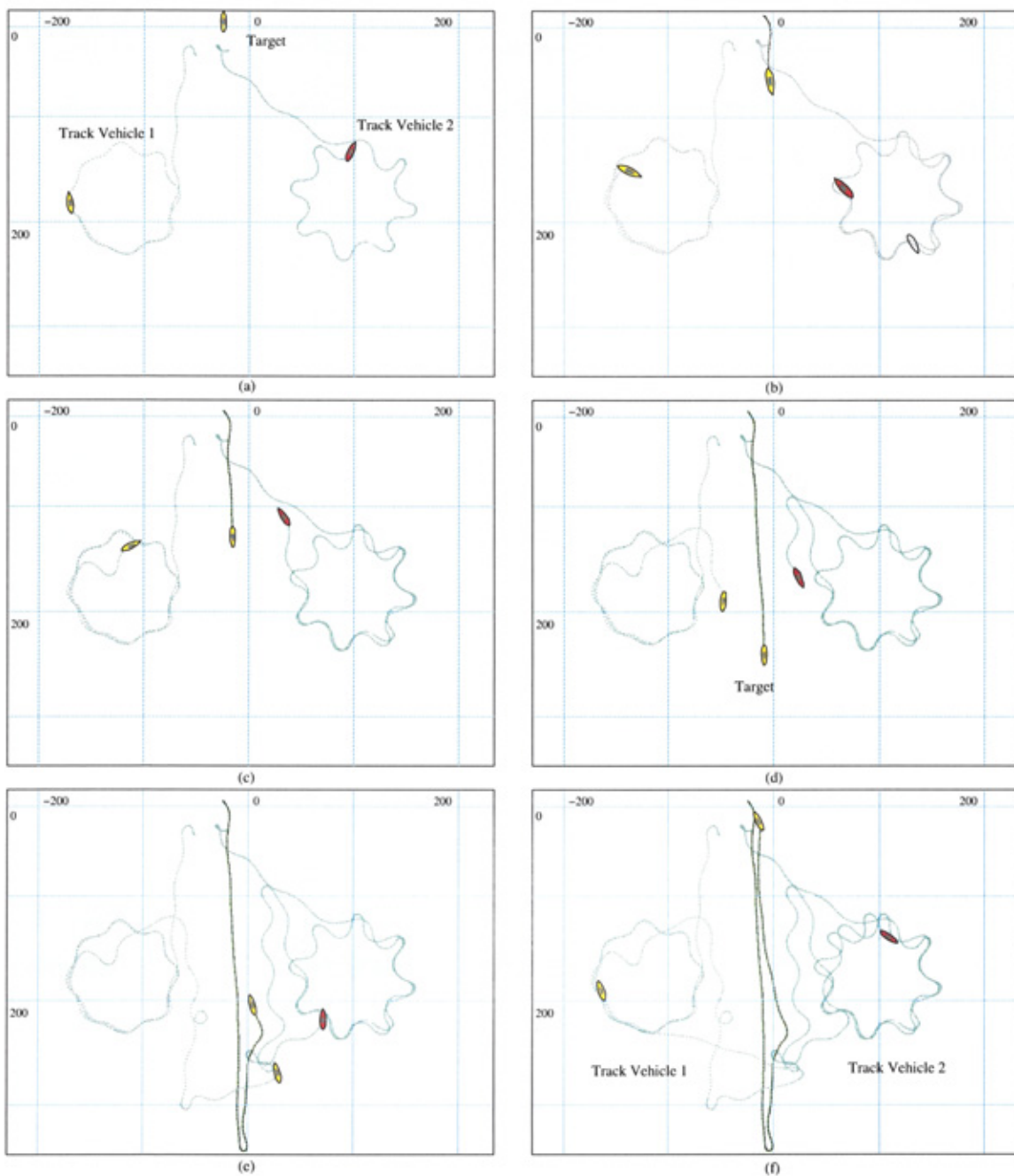


Figure 7-16: This figure shows the vehicle motion for the two sensor vehicles and the target vehicle for Mission 1144. The vehicles are executing the scenario described in Section 7.2.4 using three autonomous kayaks maneuvering within the Charles River test area.

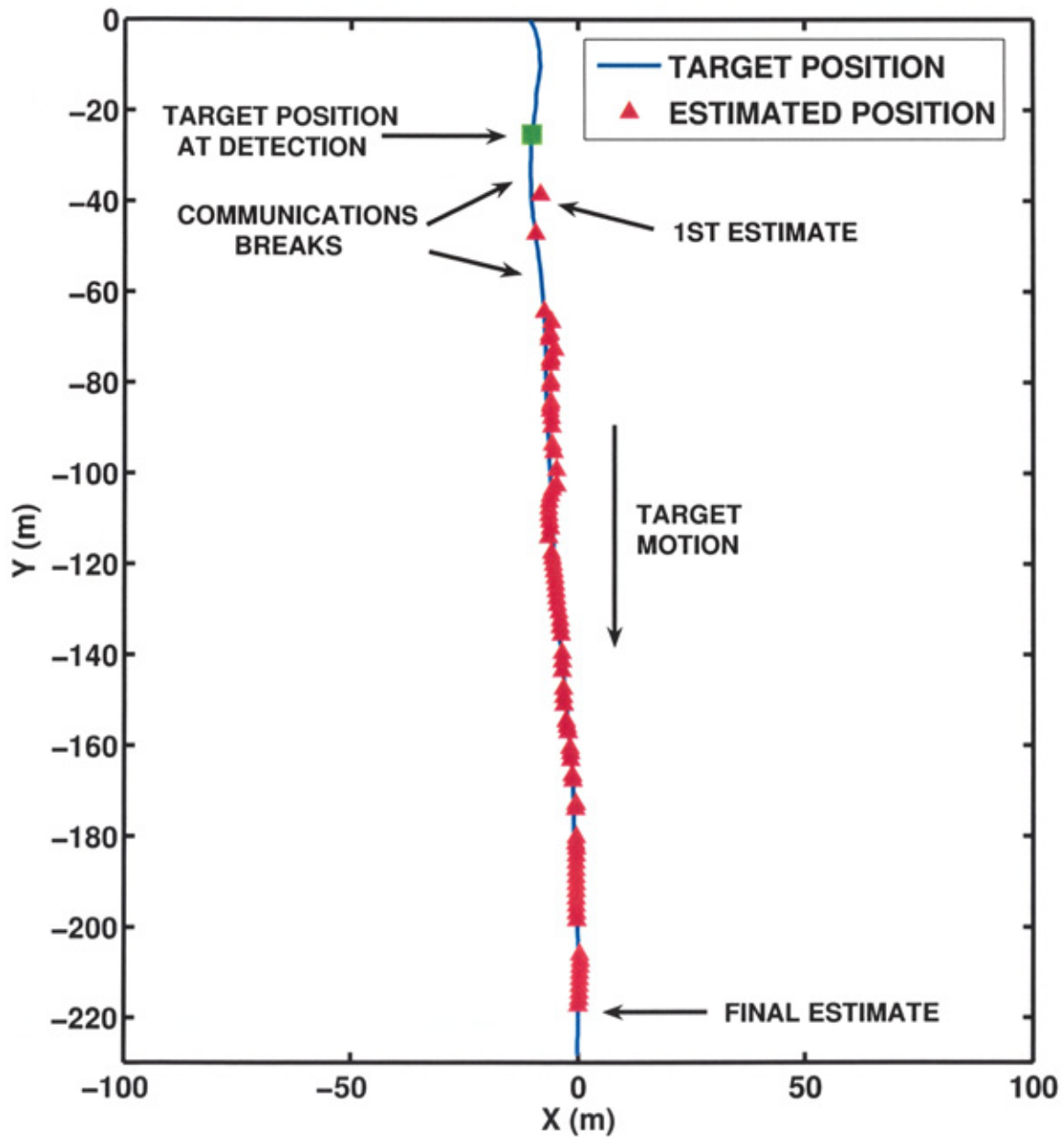


Figure 7-17: Target track solution results. This figure depicts the target position estimates produced by the MOOS process pTracker overlaid onto the actual target track for Mission 1144 for the period in which the target vehicle was operating in a constant velocity scenario. As can be seen, excellent position estimates were obtained, especially compared with the tracking results obtained using a single sensor platform to track a constant velocity target as shown in [64]. The gaps in the estimates as seen in the figure were due to communications breaks when no bearing estimates from vehicle two were received by vehicle one.

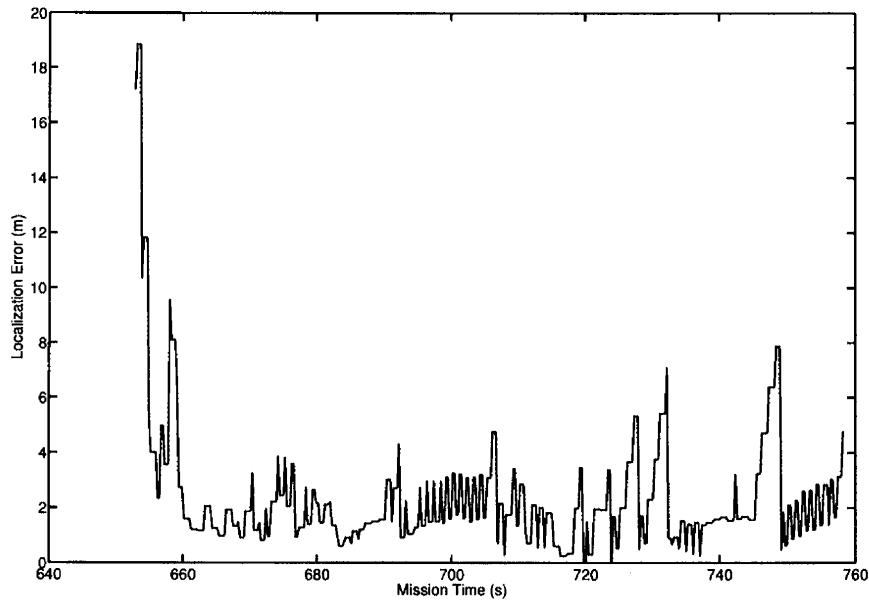


Figure 7-18: Target Track Error. This figure shows the target track error as a function of mission time for Mission 1144. As can be seen, even with the communications breaks, position estimation results were generally very good, with an error of approximately 2 meters once steady state was reached.

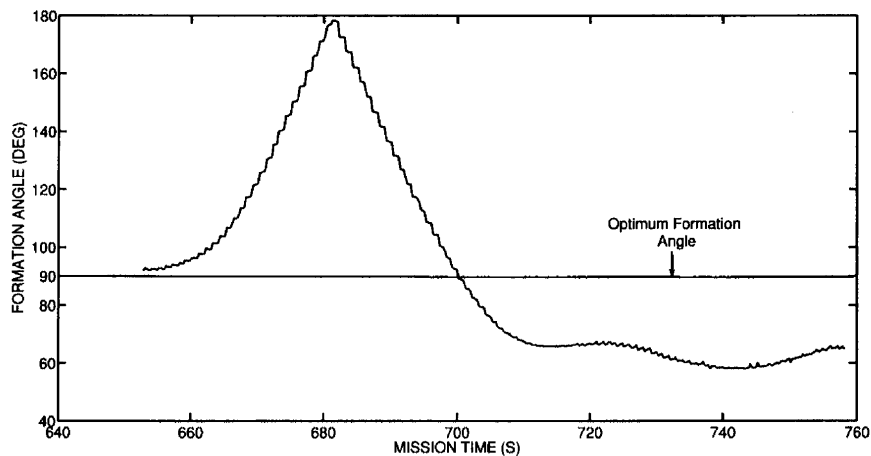


Figure 7-19: Formation angle. This figure shows the formation angle between the two sensor platforms as a function of mission time for mission 1144. Given the other constraints on the vehicle control, the optimal formation angle was not fully obtained but the angle was well within acceptable limits as shown in Fig. 7-3 and 7-4.

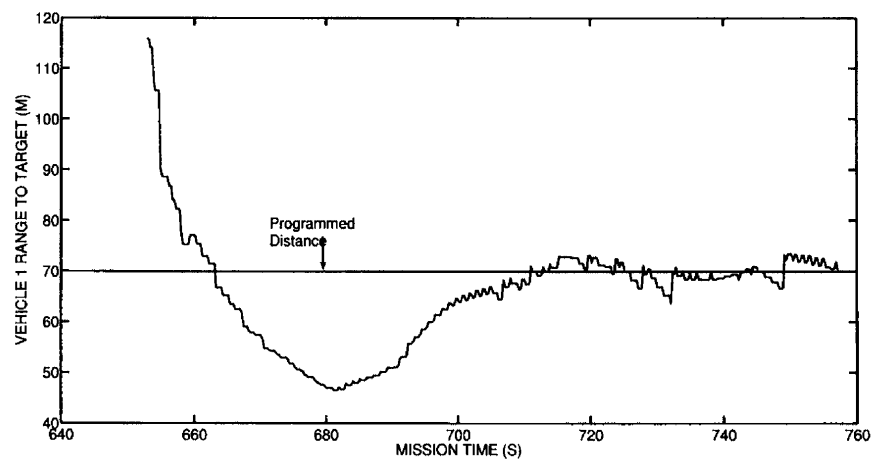


Figure 7-20: Target range for sensor platform one. This figure shows the range between sensor platform one and the target estimate as a function of mission run time for Mission 1144.

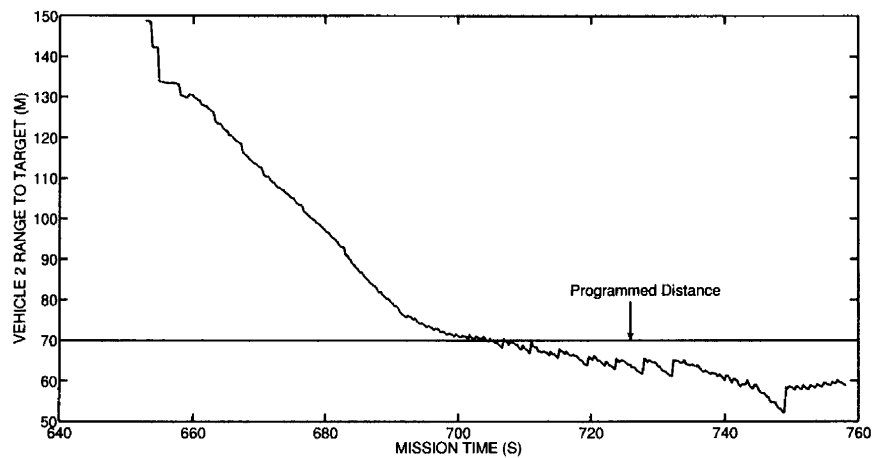


Figure 7-21: Target range for sensor platform two. This figure shows the range between sensor platform two and the target estimate as a function of mission run time for mission 1144.

7.3.3 Mission 1121

Fig. 7-24 shows the vehicle motion for an experimental tracking mission with autonomous kayaks (see Fig. 6-4) with two tracking vehicles and one target vehicle. The objective of this mission is to execute the scenario described in Section 7.2.4 where two sensor vehicles cooperatively track a target vehicle moving with constant velocity. This mission took place in the Charles River test range on December 1st, 2005. In (a) two tracking vehicles are deployed and executing their Orbit behaviors to patrol in two separate regions. Note that tracking vehicle two, on the right, is exhibiting signs of a rudder control problem. In (b) the target vehicle is deployed and has just entered the sensor region where it begins to transmit its position data to the tracking vehicles for use in the bearing simulators. The tracking vehicles have just activated their ArrayTurn behaviors for determining which side of the sensor array the target is on and the sensor vehicles have begun their turns. In (c) the tracking vehicles have just sufficiently resolved the left-right ambiguity and have begun executing their Formation behaviors using the target position estimate as a virtual leader. In (d) both the tracking vehicles have moved into formation behind the target to begin the track and trail configuration.. In (e), both sensor vehicles are still in formation trailing the target. In (f) tracking is complete and both vehicles are back on-station and awaiting any further contacts to enter their sensor fields. The target vehicle has returned to the dock.

Fig. 7-25 depicts the target position estimates produced by the MOOS process pTracker overlaid onto the actual target track for the period in which the target vehicle was operating in a constant velocity scenario. As can be seen, excellent position estimates were obtained, especially compared with the tracking results obtained using a single sensor platform to track a constant velocity target as shown in [64]. The gaps in the estimates as seen in the figure were due to communications breaks when no bearing estimates from vehicle two were received by vehicle one. Fig. 7-26 shows the error in the target position estimate as a function of mission run time. As can be seen, even with the communications breaks, position estimation results were generally very good, with an error of approximately 2 *meters* once steady state was reached.

Fig. 7-27 shows the angle between the two sensor vehicles as a function of mission run time. Given the other constraints on the vehicle control, the optimal formation angle was not fully obtained but the angle was well within acceptable limits as shown in Fig. 7-3 and 7-4.

Fig. 7-28 and 7-29 show the sensor vehicle to target estimate ranges. Both sensor vehicles were able to obtain the programmed distance from the target position estimate in steady state.

Fig. 7-30 depicts the actual speed of the target versus the speed estimate produced by pTracker for mission 1448. As can be seen, the speed estimate is very good, generally within 0.2 m/s of the actual speed of the target. The target speed was obtained from GPS.

Fig. 7-31 depicts the actual heading of the target versus the heading estimate produced by pTracker for mission 1448. As can be seen, the heading estimate is also very good, generally within 5 degrees of the actual heading of the target. The target heading was obtained from GPS.

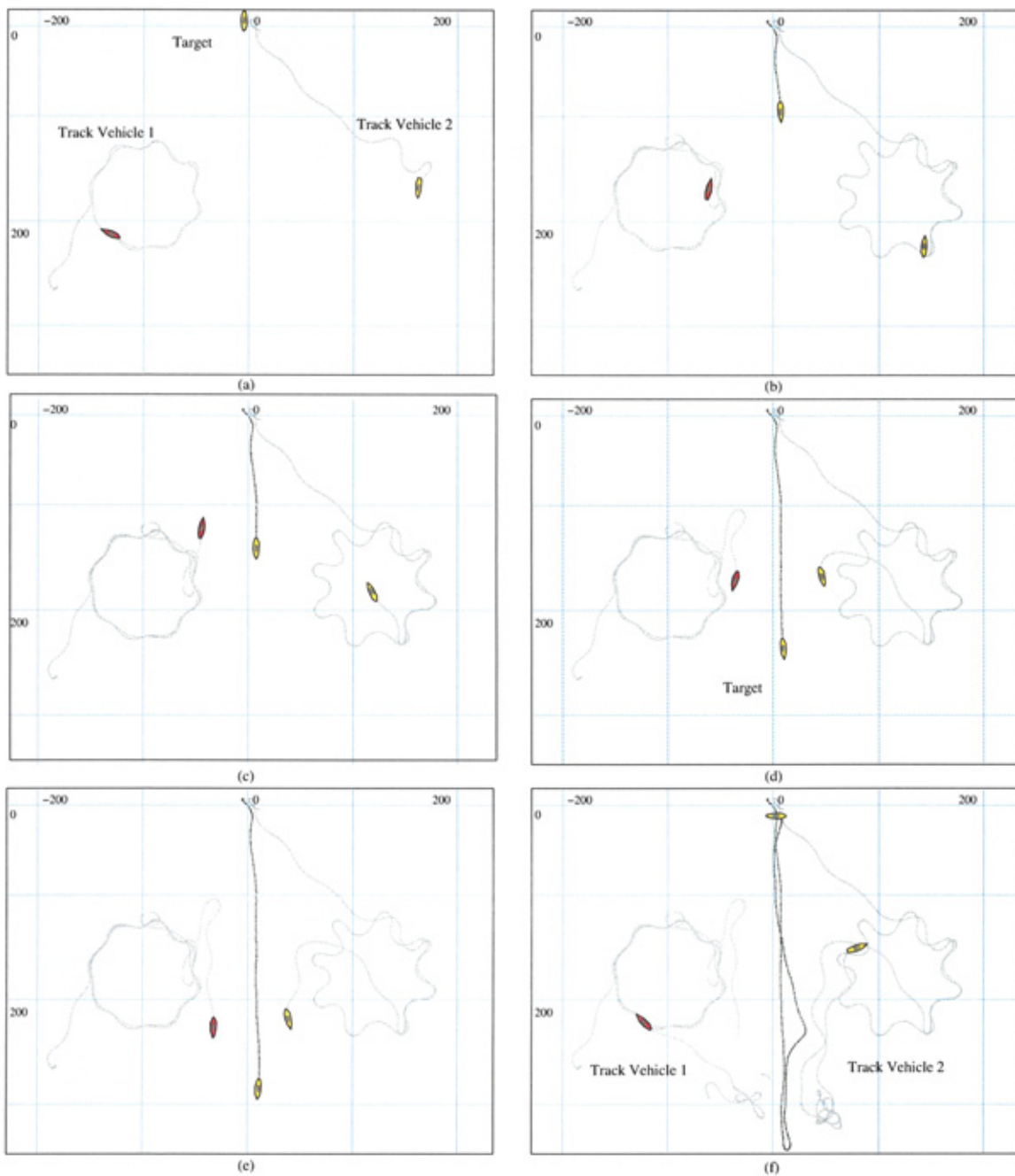


Figure 7-24: This figure shows the vehicle motion for the two sensor vehicles and the target vehicle for Mission 1121. The vehicles are executing the scenario described in Section 7.2.4 using three autonomous kayaks maneuvering within the Charles River test area.

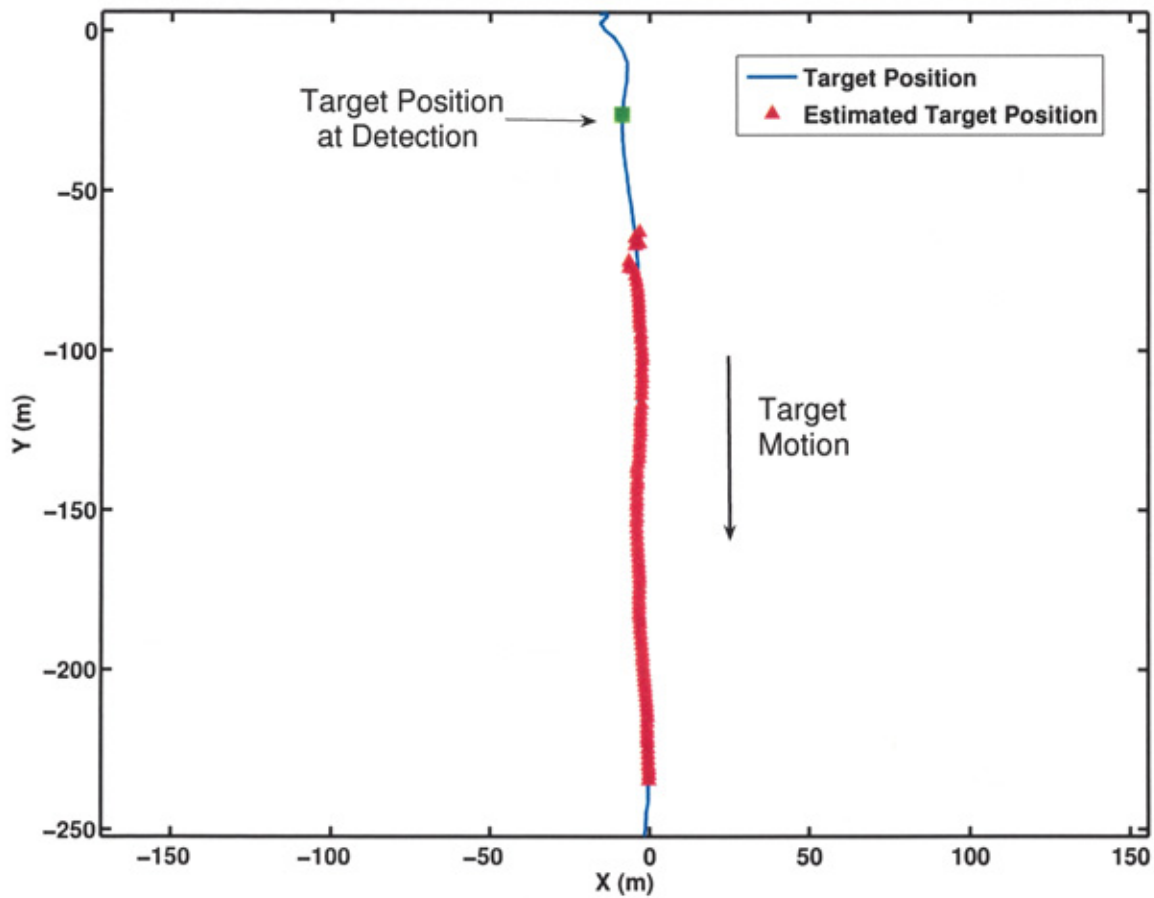


Figure 7-25: Target track solution results. This figure depicts the target position estimates produced by the MOOS process pTracker overlaid onto the actual target track for Mission 1121 for the period in which the target vehicle was operating in a constant velocity scenario. As can be seen, excellent position estimates were obtained, especially compared with the tracking results obtained using a single sensor platform to track a constant velocity target as shown in [64]. The gaps in the estimates as seen in the figure were due to communications breaks when no bearing estimates from vehicle two were received by vehicle one.

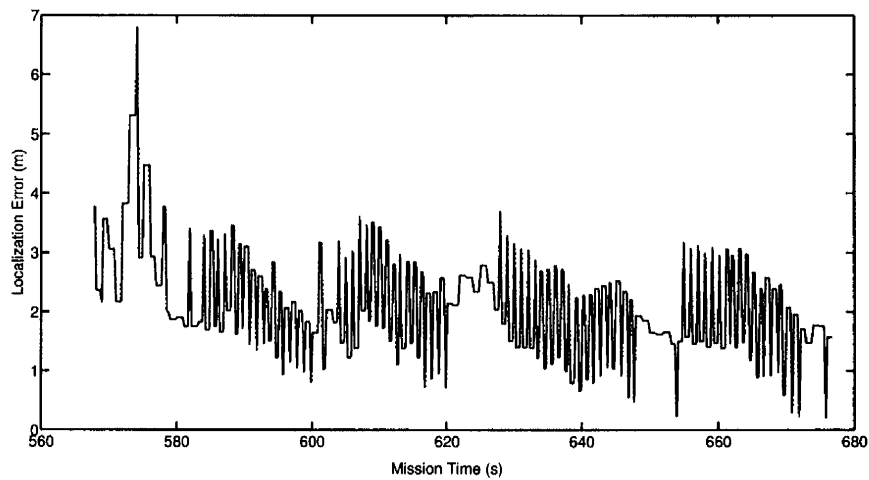


Figure 7-26: Target Track Error. This figure shows the target track error as a function of mission time for Mission 1121. As can be seen, even with the communications breaks, position estimation results were generally very good, with an error of approximately 2 meters once steady state was reached.

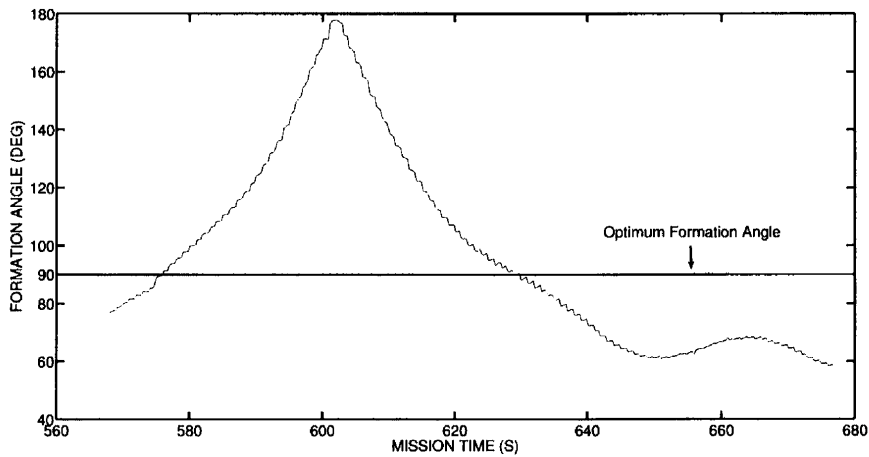


Figure 7-27: Formation angle. This figure shows the formation angle between the two sensor platforms as a function of mission time for mission 1121. Given the other constraints on the vehicle control, the optimal formation angle was not fully obtained but the angle was well within acceptable limits as shown in Fig. 7-3 and 7-4.

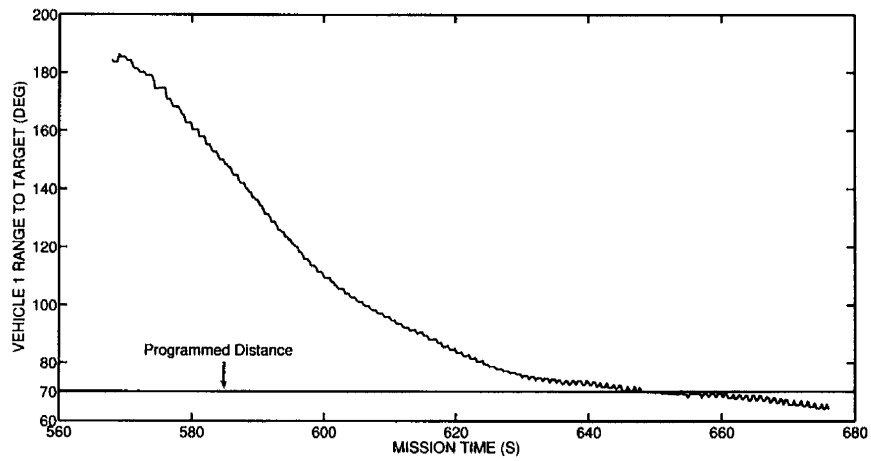


Figure 7-28: Target range for sensor platform one. This figure shows the range between sensor platform one and the target estimate as a function of mission run time for mission 1121.

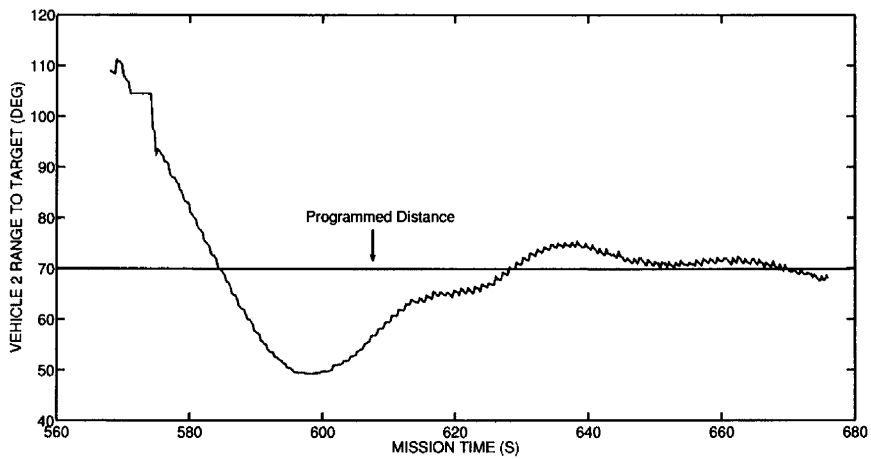


Figure 7-29: Target range for sensor platform two. This figure shows the range between sensor platform two and the target estimate as a function of mission run time for mission 1121.

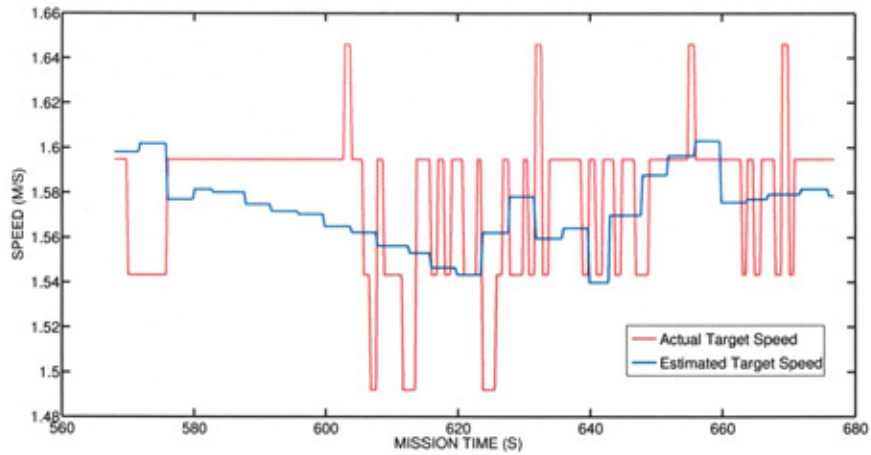


Figure 7-30: Speed estimate error. This figure depicts the actual speed of the target versus the speed estimate produced by pTracker for mission 1121. As can be seen, the speed estimate is very good, generally within 0.2 m/s of the actual speed of the target. The target speed was obtained from GPS.

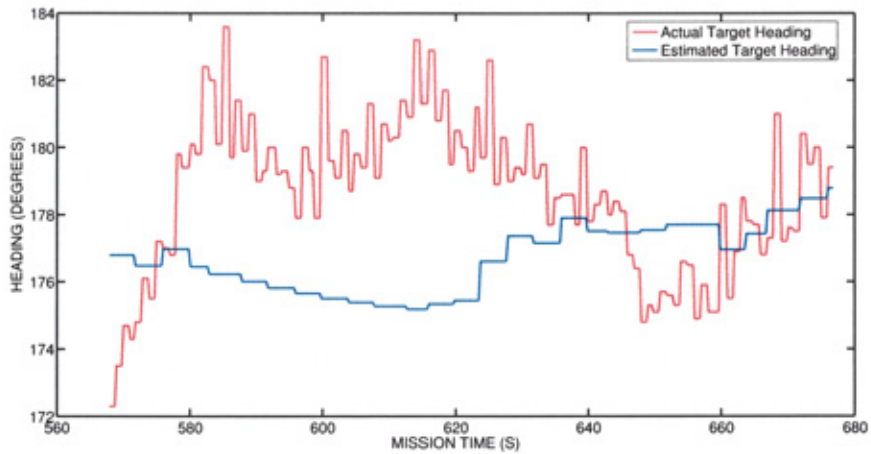


Figure 7-31: Heading estimate error. This figure depicts the actual heading of the target versus the heading estimate produced by pTracker for mission 1121. As can be seen, the heading estimate is very good, generally within 5 degrees of the actual heading of the target. The target heading was obtained from GPS.

Chapter 8

Summary and Conclusions

8.1 Summary of Contributions

This thesis makes a number of contributions toward an understanding of adaptive sampling in marine sensor networks. One of the significant contributions made is an innovative architecture for sensor-adaptive control of autonomous sensor platforms in marine sensor networks. This architecture consists of:

- One or more logical sensors that are designed to provide high-level environmental state data to a behavior-based control system on an autonomous sensor platform. These logical sensors are designed to be application-specific and to hide the sensor implementation details from the control system. A major advantage of the logical sensor approach is that sensing algorithms can be modified without having to modify the vehicle control behaviors. For example, in the logical targeting sensor used in the two experimental examples, the target detection and/or tracking algorithms could be modified without changing any of the tracking behaviors. The logical sensor approach also allows multiple physical sensors on the same sensor platform or multiple distributed sensors to be integrated into a single sensor from the viewpoint of the control system. In the two-bearing tracking example in Chapter 7 for example, the logical targeting sensor fused the data from two distributed sensors into a single target track.
- A behavior-based control system which utilizes objective functions for action selection. The use of objective functions in the behavior-based control system allows for compromise be-

tween the needs of individual behaviors. This allows much more flexibility than in schemes that only pick the output of a single behavior (e.g. Brooks' subsumption architecture) and those that average the output of individual behaviors (e.g. motor schema). The computational intensiveness of the multi-function optimization approach is greatly reduced by using the Interval Programming Method developed by Benjamin [3].

- A hybrid multi-robot cooperation scheme utilizing the behavior-based control approach but with some of the behaviors designed to cooperate with other autonomous sensor platforms. This cooperation is achieved by the sharing of state data among the distributed sensor platforms. This approach keeps the reactive control ability of behavior-based schemes but does away with the explicit task negotiation common to typical "intentional cooperation" approaches to cooperative control.

Another significant contribution was the development of a targeting sensor for the Odyssey-III AUV used in the department to support a number of research programs investigating sensor-adaptive target localization and tracking with multiple, cooperating sensor platforms (described in detail in Chapter 5). This sensor was developed with the following major characteristics:

- Active or passive acoustic sampling utilizing the Odyssey-III acoustic nose array.
- Integration with the sensor platform control system as a logical targeting sensor.
- Time synchronization with other sensor platforms with microsecond accuracy.

This sensor has been used to support four major international experiments (GOATS 2002, GOATS 2004, FAF 2005 and the upcoming Monterey Bay 2006 experiment), four Ph.D. theses including this report, and a number of papers [60] [65]. The algorithms and behaviors used in the experiments detailed in Chapters 6 and 7 were developed for this targeting sensor.

The third significant contribution was the execution of two experimental examples illustrating the concepts developed in the thesis. The two examples illustrated the following:

- The relationship between the sensor platform motion and the uncertainty of the parameter estimates for target tracking with both one and two sensor platforms.
- The development of sensor platform control behaviors designed to reduce the uncertainty of the parameter estimates in both the one and two sensor tracking cases.

- The integration and operation of a logical targeting sensor with a behavior-based control system that utilizes an objective function approach to action selection.
- The effectiveness of the hybrid approach to multi-robot cooperation. Two sensor platforms were able to maintain relative formation with each other while cooperatively tracking a moving target.
- The effectiveness of a distributed, multi-sensor approach to the estimation of the parameters of a process. Although the single sensor platform was able to track a moving target in the experiment detailed in Chapter 6, the distributed tracking platforms were able to converge to the target track much faster and were able to achieve a much lower estimate error.

8.2 Conclusions

While the architecture and methods for adaptive sampling in marine sensor networks described in this report were shown to have promise, there are some areas that need more attention:

- The proper weighting of the behaviors is a difficult task since the weights can be dynamic and change with the state of the environment. For example, in the single sensor platform tracking experiment in Chapter 6, both the `ArrayAngle` and the `CloseRange` behaviors were active simultaneously. The goal was to have the weight of the `ArrayAngle` behavior increasingly dominate as the sensor platform got closer to the target. However, it was not clear what the optimal weighting should be as a function of range in order to optimize both the Fisher information and the parameter observability. This task is made more difficult by the fact that the behavior weightings may be different in various scenarios such as a fast target moving away or for slow targets. One solution to this issue would be to use a Monte Carlo simulator to be able to efficiently test thousands of scenarios in a probabilistic manner. The simulator could be paired with some sort of search mechanism like a genetic algorithm to optimize the behavior weights.
- Accurate sensor models are critical to understanding how the variance of our parameter estimates varies with platform motion and therefore how to design appropriate behaviors to reduce this variance. In this report, extremely simple sensor models were used. However, to realistically model the platform behavior, more complex models are needed. For the bearing

sensor for example, the probability of target detection and the measurement variance change as a function of target range. The measurement variance also changes with the angle of the array with respect to the target. It is not really possible to use more complex sensor models when utilizing the bearing simulator with the autonomous surface craft but more complex sensor models could easily be incorporated into a Monte Carlo simulator.

8.3 Future Work

The target tracking problem is an excellent problem to use to develop adaptive sampling methods for marine sensor networks. The problem is sufficiently difficult, relevant, and can benefit from the application of distributed sensors. In Fig. 8-1(a) is a depiction of the example experiment discussed in Chapter 7 where two sensor platforms track a moving target. Although it is shown that two sensors are much better at tracking a target than one sensor, it is possible that two distributed sensors may also have bad geometry with respect to the target. As you increase the number of sensors, it becomes more likely that any *pair* of sensors has good geometry as depicted in Fig. 8-1(b). This N-sensor approach brings up several difficult questions such as how do we fuse N bearings into a target track, and what is the optimal sensor placement for N sensors? The latter question is related to the so-called “paparazzi problem” formulated by Jenkin and Dudek in [66]. Solutions to these questions are a topic of current investigation.

An even more difficult problem is that of attempting to track multiple targets with distributed sensors as depicted in Fig. 8-2. Even though target two is occluded by target one with respect to sensor one, target two is clearly seen by sensor two. The same is also true of target three which is occluded by target two with respect to sensor two but is clearly seen by sensor one. However, the use of distributed sensors introduces an addition problem in that “ghost” targets can appear as seen in Fig. 8-2 where no actual target exists [67]. This issue, along with the typical data association problems associated with multiple target tracking are very difficult but, a very significant advantage exist when using mobile sensor platforms in that we may be able to maneuver the sensor platforms in such a way as to be able to resolve some of the ghosting and data association problems. This is also an area of current investigation.

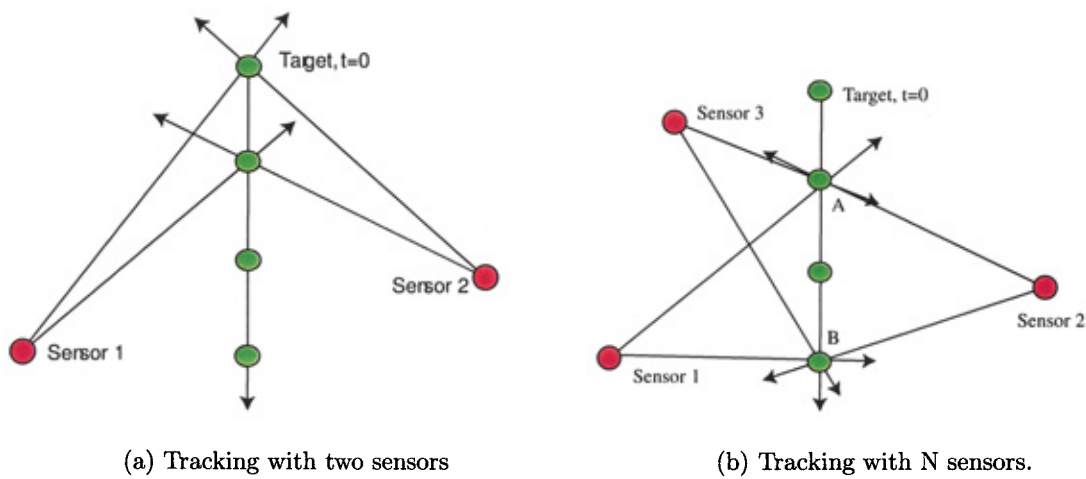


Figure 8-1: In (a), two sensors are able to track the target but may not always be able to maintain the most effective placement while in (b) it is seen that it is more likely that given *pairs* of sensors may always be able to have good geometry with respect to the target.

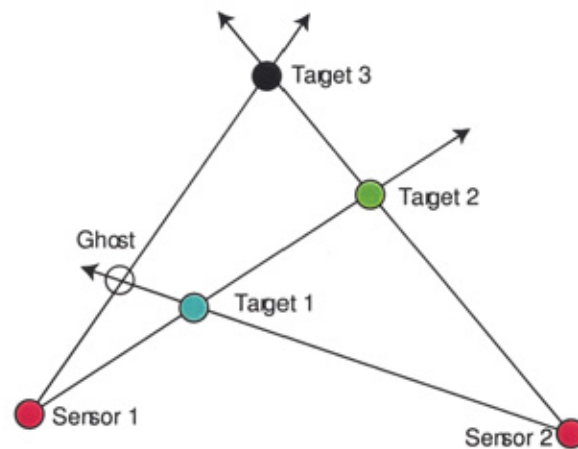


Figure 8-2: This figure depicts the multiple target tracking scenario with multiple sensors. As can be seen in the figure, distributed sensors can solve the occlusion problem but there are still issues with “ghost” targets.

Appendix A

Supporting Equations

The partial derivatives used in equations 7.6 and 7.7 to compute the estimated variances for the target state vector were derived in [63] and are given as:

$$\frac{\partial x_t}{\partial x_1} = \frac{-\tan \theta_2}{\tan \theta_1 - \tan \theta_2} \quad \frac{\partial x_t}{\partial x_2} = \frac{\tan \theta_1}{\tan \theta_1 - \tan \theta_2} \quad (\text{A.1})$$

$$\frac{\partial x_t}{\partial y_1} = \frac{\tan \theta_1 \tan \theta_2}{\tan \theta_1 - \tan \theta_2} \quad \frac{\partial x_t}{\partial y_2} = \frac{-\tan \theta_1 \tan \theta_2}{\tan \theta_1 - \tan \theta_2} \quad (\text{A.2})$$

$$\frac{\partial y_t}{\partial y_1} = \frac{\tan \theta_1}{\tan \theta_1 - \tan \theta_2} \quad \frac{\partial y_t}{\partial y_2} = \frac{\tan \theta_2}{\tan \theta_1 - \tan \theta_2} \quad (\text{A.3})$$

$$\frac{\partial y_t}{\partial x_1} = \frac{-1}{\tan \theta_1 - \tan \theta_2} \quad \frac{\partial y_t}{\partial x_2} = \frac{1}{\tan \theta_1 - \tan \theta_2} \quad (\text{A.4})$$

$$\frac{\partial x_t}{\partial \theta_1} = \left[\frac{x_2 + (y_1 - y_2) \tan \theta_2}{\tan \theta_1 - \tan \theta_2} - \frac{x_2 \tan \theta_1 - x_1 \tan \theta_2 + (y_1 - y_2) \tan \theta_1 \tan \theta_2}{(\tan \theta_1 - \tan \theta_2)^2} \right] \sec^2 \theta_1 \quad (\text{A.5})$$

$$\frac{\partial x_t}{\partial \theta_2} = \left[\frac{-x_1 + (y_1 - y_2) \tan \theta_1}{\tan \theta_1 - \tan \theta_2} - \frac{x_2 \tan \theta_1 - x_1 \tan \theta_2 + (y_1 - y_2) \tan \theta_1 \tan \theta_2}{(\tan \theta_1 - \tan \theta_2)^2} \right] \sec^2 \theta_2 \quad (\text{A.6})$$

$$\frac{\partial y_t}{\partial \theta_1} = \left[\frac{y_1}{\tan \theta_1 - \tan \theta_2} - \frac{y_1 \tan \theta_1 - y_2 \tan \theta_2 + x_2 - x_1}{(\tan \theta_1 - \tan \theta_2)^2} \right] \sec^2 \theta_1 \quad (\text{A.7})$$

$$\frac{\partial y_t}{\partial \theta_2} = \left[\frac{-y_2}{\tan \theta_1 - \tan \theta_2} - \frac{y_1 \tan \theta_1 - y_2 \tan \theta_2 + x_2 - x_1}{(\tan \theta_1 - \tan \theta_2)^2} \right] \sec^2 \theta_2 \quad (\text{A.8})$$

Appendix B

Behavior Code

B.1 ArrayTurn Behavior

B.1.1 BHV_ArrayTurn.h

```
/* ***** */
/*  NAME: Don Eickstedt                               */
/*  FILE: BHV_ArrayTurn.h                               */
/* ***** */
#ifndef BHV_ARRAYTURN_HEADER
#define BHV_ARRAYTURN_HEADER

#include "IvPBehavior.h"

using namespace std;

class IvPDomain;
class BHV_ArrayTurn : public IvPBehavior {
public:
    BHV_ArrayTurn(IvPDomain);
    ~BHV_ArrayTurn() {};

    IvPFunction* produceOF();
    bool setParam(std::string, std::string);

protected:

    double course_fix;

    bool course_fixed;
};
#endif
```

B.1.2 BHV_ArrayTurn.cpp

```

/*****
/*   NAME: Don Eickstedt
/*   FILE: BHV_ArrayTurn.cpp
/*   DATE: July 03 2005
*****/

#include <string>
#include <iostream>
#include <math.h>
#include "BHV_ArrayTurn.h"
#include "MBUtils.h"
#include "AOF_ArrayTurn.h"
#include "OF_Reflector.h"

using namespace std;

//-----
// Procedure: Constructor

BHV_ArrayTurn::BHV_ArrayTurn(IvPDomain gdomain) :
    IvPBehavior(gdomain)
{

    this->setParam("descriptor", "(d)bhv_1BTrack");
    this->setParam("unifbox", "course=3");
    this->setParam("gridbox", "course=9");
    course_fixed = false;

    info_vars.push_back("NAV_HEADING");
}

//-----
// Procedure: setParam

bool BHV_ArrayTurn::setParam(string param, string val)
{

    IvPBehavior::setParamCommon(param, val);

    return true;
}

//-----
// Procedure: produceOF
```



```

IvPFunction *BHV_ArrayTurn::produceOF()
{
    messages.clear();

    // Need to know the name of ownship to query position
    if(us_name == "") {
        postEMessage("error,BHV_ArrayTurn: ownship name not known.");
        return(0);
    }

    bool ok1;

    //get current course
    double osCourse = info_buffer->dQuery(us_name, "NAV_HEADING", &ok1);

    if(!ok1){
        postEMessage("error,BHV_ArrayTurn: ownship data not available");
        return (0);
    }

    //check to see if behavior just activated - if so, save current course
    if (!course_fixed){
        course_fix = osCourse;
        course_fixed = true;
    }

    AOF_ArrayTurn aof_track(domain,course_fix,osCourse);

    OF_Reflector *ofr_track = new OF_Reflector(&aof_track,1);

    ofr_track->create_uniform(unif_box,grid_box);

    IvPFunction *of = ofr_track->extractOF();

    of->getPDMap()->normalize(0.0,100.0);

    delete(ofr_track);

    of->setDomainName(0, "course");

    of->setPWT(priority_wt);

    return(of);
}

```

B.1.3 AOF_ArrayTurn.h

```

/*****
/*   NAME: Don Eickstedt
/*   FILE: AOF_ArrayTurn.h
/*   DATE: 23 July 05
*****/
#ifndef AOF_ARRAYTURN_HEADER
#define AOF_ARRAYTURN_HEADER

#include "AOF.h"

class IvPDomain;
class AOF_ArrayTurn: public AOF
{
public:

    AOF_ArrayTurn(IvPDomain, double,double);
    ~AOF_ArrayTurn() {};

public:

    double evalBox(const IvPBox*) const;    // virtual defined
    double lrmetric(double) const;

protected:

    double leftabs,rightabs,hwidth,osCourse;
    double crsBase;
    double crsDelta;
};
#endif
```

B.1.4 AOF_ArrayTurn.cpp

```

/*****
/*   NAME: Don Eickstedt
/*   FILE: AOF_ArrayTurn.cpp
/*   BORN: 23 July 05
/*
/*   The methods in this class are responsible for
/*   producing the objective function for the ArrayTurn
/*   behavior.
*****/

#include <iostream>
#include <math.h>
#include <assert.h>
#include "AOF_ArrayTurn.h"
#include "AngleUtils.h"
#include "IvPDomain.h"

using namespace std;

//-----
// Procedure: Constructor for the ArrayTurn class.
//-----

AOF_ArrayTurn::AOF_ArrayTurn(IvPDomain g_domain, double course_fix, double course)
{
    int crs_ix = g_domain.getIndex("course");

    assert(crs_ix != -1);

    crsDelta = g_domain.get_ddelta(crs_ix);
    crsBase  = g_domain.get_dlow(crs_ix);

    universe = IvPBox(1);
    universe.setPTS(0, 0, g_domain.get_dpoints(crs_ix)-1);

    //the course fix is the course at the time the ArrayTurn behavior
    //became active

    //find the center of the left mode
    leftabs = course_fix-90.0;
    if (leftabs < 0.0)
        leftabs += 360.0;

    //find the center of the right mode
    rightabs = course_fix + 90.0;
    if(rightabs > 360.0)

```

```

    rightabs -= 360.0;

    //half the width of the range of courses to
    //consider centered around the desired course
    //this should be an input parameter - change
    double width = 20;

    hwidth = width;
    osCourse = course;

}

//-----
// Procedure: evalBox
// Purpose: Evaluates a given course
//-----

double AOF_ArrayTurn::evalBox(const IvPBox *b) const
{
    double evalCRS = crsBase + ((double)(b->pt(0,0)) * crsDelta);
    double mval;

    mval = (lrmtric(evalCRS));

    return mval;
}

//-----
// Procedure: metric
//
// This method is responsible for applying a metric to each possible
// course we are evaluating. This produces a bi-modal function over
// course which is weighted toward the current course.
//-----

double AOF_ArrayTurn::lrmtric(double evalCRS) const
{
    double mval;

    double left_perr = fabs(evalCRS - leftabs);
    if (left_perr > 180.0)
        left_perr = 360.0 - left_perr;

    double right_perr = fabs(evalCRS - rightabs);
    if (right_perr > 180.0)
        right_perr = 360.0 - right_perr;

```

```

double left_cerr = fabs(osCourse-leftabs);
if (left_cerr > 180.0)
    left_cerr = 360.0 - left_cerr;

double right_cerr = fabs(osCourse-rightabs);
if (right_cerr > 180.0)
    right_cerr = 360.0 - right_cerr;

if (left_cerr <= right_cerr)
{
    if (left_perr < right_perr)
        mval = (((200-left_perr)/2.0) + (90.0-left_cerr)/2.0);
    else
        mval = (200-right_perr)/2.0;
}
else
{
    if (right_perr <= left_perr)
        mval = (((200-right_perr)/2.0) + (90.0 - right_cerr)/2.0);
    else
        mval = (200-left_perr)/2.0;
}

if ((left_perr > hwidth) && (right_perr > hwidth))
    mval = 0.0;

return mval;
}

```

B.2 ArrayAngle Behavior

B.2.1 AOF_ArrayAngle.h

```
/* ***** */
/* NAME: Don Eickstedt */
/* FILE: AOF_ArrayAngle.h */
/* DATE: 23 July 05 */
/* ***** */

#ifndef AOF_ARRAYANGLE_HEADER
#define AOF_ARRAYANGLE_HEADER

#include "AOF.h"

class IvPDomain;
class AOF_ArrayAngle: public AOF {

public:

    AOF_ArrayAngle(IvPDomain);
    ~AOF_ArrayAngle() {};

    double evalBox(const IvPBox*) const;    // virtual defined
    double Arraymetric(double) const;
    bool setParam(const std::string&, double);
    bool initialize();

protected:

    double osCourse;        //ownship course
    double osX;
    double osY;
    double tx,ty,raydirec;
    double leftabs,rightabs,hwidth;

    double crsBase,spdBase,tolBase;
    double crsDelta,tolDelta,spdDelta;
};
#endif
```

B.2.2 AOF_ArrayAngle.cpp

```

/*****
/*   NAME: Don Eickstedt
/*   FILE: AOF_ArrayAngle.cp
/*   BORN: 23 July 05
*****/
#include <iostream>
#include <math.h>
#include <assert.h>
#include "AOF_ArrayAngle.h"
#include "AngleUtils.h"
#include "IvPDomain.h"

using namespace std;

//-----
// Procedure: Constructor

AOF_ArrayAngle::AOF_ArrayAngle(IvPDomain g_domain)
{
    int crs_ix = g_domain.getIndex("course");
    int spd_ix = g_domain.getIndex("speed");

    assert(crs_ix != -1);
    assert(spd_ix != -1);

    crsDelta = g_domain.get_ddelta(crs_ix);
    crsBase  = g_domain.get_dlow(crs_ix);

    spdDelta = g_domain.get_ddelta(spd_ix);
    spdBase  = g_domain.get_dlow(spd_ix);

    universe = IvPBox(2);
    universe.setPTS(0, 0, g_domain.get_dpoints(crs_ix)-1);
    universe.setPTS(1, 0, g_domain.get_dpoints(spd_ix)-1);
}

//-----
// Procedure: evalBox
// Purpose: Evaluates a given course
//

double AOF_ArrayAngle::evalBox(const IvPBox *b) const
{
    double evalCRS  = crsBase + ((double)(b->pt(0,0)) * crsDelta);

```

```

double mval;

mval = (Arraymetric(evalCRS));

return mval;
}

//-----
// Procedure: ArrayMetric
//
// This method applies a metric to a course that is being evaluated.
// It produces a bi-modal objective function over course with the
// mode closest to the current course being favorably weighted.

double AOF_ArrayAngle::Arraymetric(double evalCRS) const
{
    double mval;

    double left_perr = fabs(evalCRS - leftabs);
    if (left_perr > 180.0)
        left_perr = 360.0 - left_perr;

    double right_perr = fabs(evalCRS - rightabs);
    if (right_perr > 180.0)
        right_perr = 360.0 - right_perr;

    double left_cerr = fabs(osCourse-leftabs);
    if (left_cerr > 180.0)
        left_cerr = 360.0 - left_cerr;

    double right_cerr = fabs(osCourse-rightabs);
    if (right_cerr > 180.0)
        right_cerr = 360.0 - right_cerr;

    if (left_cerr <= right_cerr)
    {
        if (left_perr < right_perr)
            mval = (((200-left_perr)/2.0) + (90.0-left_cerr)/2.0);
        else
            mval = (200-right_perr)/6.0;
    }
    else
    {
        if (right_perr <= left_perr)
            mval = (((200-right_perr)/2.0) + (90.0 - right_cerr)/2.0);
    }
}

```



```

        else
            mval =(200-left_perr)/6.0;
        }

        if ((left_perr > hwidth) && (right_perr > hwidth))
            mval = 0.0;

        return mval;
    }

//This method sets all the internal parameters
//This method is called from BHV_ArrayAngle.cpp
bool AOF_ArrayAngle::setParam(const string& param, double param_val)
{
    if(param == "width") {
        hwidth = param_val;
        return(true);
    }
    else if(param == "osCourse") {
        osCourse = param_val;
        return(true);
    }
    else if(param == "osX") {
        osX = param_val;
        return(true);
    }
    else if(param == "osY") {
        osY = param_val;
        return(true);
    }
    else if(param == "tx") {
        tx = param_val;
        return(true);
    }
    else if(param == "ty") {
        ty = param_val;
        return(true);
    }
    else
        return(false);
}

//-----
//This method initializes the class
//This method determines the center of the two
//modes of the bi-modal objective function
//given the position of the target and ownship

```

```
//-----

bool AOF_ArrayAngle::initialize()
{
    raydirec = 90.0-atan2(osY-ty,osX-tx)*180.0/M_PI;

    leftabs = raydirec-90.0;
    if (leftabs < 0.0)
        leftabs += 360.0;

    rightabs = raydirec + 90.0;
    if(rightabs > 360.0)
        rightabs -= 360.0;

    return(true);
}
```

B.2.3 BHV_ArrayAngle.h

```

/*****
/*   NAME: Don Eickstedt
/*   FILE: BHV_ArrayAngle.h
/*
*****/

#ifndef BHV_ARRAYANGLE_HEADER
#define BHV_ARRAYANGLE_HEADER

#include "IvPBehavior.h"

#define NO_TRACK 0
#define LR_TRACK 1
#define TRACKING 2

using namespace std;

class IvPDomain;
class BHV_ArrayAngle : public IvPBehavior {

public:

    BHV_ArrayAngle(IvPDomain);
    ~BHV_ArrayAngle() {};

    IvPFunction* produceOF();
    bool setParam(std::string, std::string);

    double getRelevance(double,double,double,double);

protected:

    int decode(string);

    double desired_angle;
    double txdot,tydot,ty,tx,range_max,range_min,heading,speed;

    int state,width,mnum;

};
#endif
```

B.2.4 BHV_ArrayAngle.cpp

```

/*****
/*   NAME: Don Eickstedt
/*   FILE: BHV_ArrayAngle.cpp
/*   DATE: July 03 2005
*****/

#include <string>
#include <iostream>
#include <math.h>
#include "BHV_ArrayAngle.h"
#include "MBUtils.h"
#include "AOF_ArrayAngle.h"
#include "OF_Reflector.h"

using namespace std;

//-----
// Procedure: Constructor

BHV_ArrayAngle::BHV_ArrayAngle(IvPDomain gdomain) :
    IvPBehavior(gdomain)
{

    this->setParam("descriptor", "(d)bhv_1BTrack");
    this->setParam("unifbox", "course=3");
    this->setParam("gridbox", "course=9");

    range_min = 0.0;
    range_max = 1000.0;

    info_vars.push_back("NAV_X");
    info_vars.push_back("NAV_Y");
    info_vars.push_back("NAV_HEADING");
    info_vars.push_back("TRACK_STAT");
}

//-----
// Procedure: setParam

bool BHV_ArrayAngle::setParam(string param, string val)
{

    IvPBehavior::setParamCommon(param, val);

    if(param == "width") {
        width = (int) atof(val.c_str());
    }
}
```

```

    }

    if(param == "range_min") {
        range_min = atof(val.c_str());
        return(true);
    }

    if(param == "range_max") {
        range_max = atof(val.c_str());
        return(true);
    }

    return true;
}

//-----
// Procedure: produceOF

IvPFunction *BHV_ArrayAngle::produceOF()
{
    messages.clear();

    // Need to know the name of ownship to query position
    if(us_name == "") {
        postEMessage("error,BHV_ArrayAngle: ownship name not known.");
        return(0);
    }

    bool ok1,ok2,ok3,ok4;
    //get current course
    double osCourse = info_buffer->dQuery(us_name, "NAV_HEADING", &ok1);
    //get current x
    double osX = info_buffer->dQuery(us_name, "NAV_X", &ok2);
    //get current heading
    double osY = info_buffer->dQuery(us_name, "NAV_Y", &ok3);
    //get current tracking state
    string tState = info_buffer->sQuery(us_name,"TRACK_STAT", &ok4);

    if(!ok1 || !ok2 || !ok3 ||!ok4){
        postEMessage("error,BHV_ArrayAngle: ownship data not available");
        return (0);
    }

    int new_state = decode(tState);

```

```

double relevance = getRelevance(osX, osY, tx, ty);

if(relevance <= 0)
    return(0);

AOF_ArrayAngle aof_track(domain);
aof_track.setParam("width",width);
aof_track.setParam("osCourse",osCourse);
aof_track.setParam("osX",osX);
aof_track.setParam("osY",osY);
aof_track.setParam("tx",tx);
aof_track.setParam("ty",ty);

aof_track.initialize();

OF_Reflector *ofr_track = new OF_Reflector(&aof_track,1);

ofr_track->create_uniform(unif_box,grid_box);

IvPFunction *of = ofr_track->extractOF();

//of->getPDMap()->normalize(0.0,100.0);

delete(ofr_track);

of->setDomainName(0, "course");

of->setPWT(priority_wt);

return(of);
}

int BHV_ArrayAngle::decode(string status)
{
    vector<string> svector;
    vector<string> svector2;

    // Parse the waypoint status string for "us"
    svector = parse_string(status, ',');
    for(unsigned int i=0; i<svector.size(); i++) {
        svector2 = parse_string(svector[i], '=');
        if(svector2.size() != 2) {
            postEMessage("error,BHV_CloseRange: Invalid waypoint string");
            return(0);
        }
    }
}

```

```

    }

    string left  = strip_blank_ends(svector2[0]);
    string right = strip_blank_ends(svector2[1]);
    if(left == "state")      state = atoi(right.c_str());
    if(left == "x")         tx     = atof(right.c_str());
    if(left == "y")         ty     = atof(right.c_str());
    if(left == "heading")   heading = atof(right.c_str());
    if(left == "speed")     speed  = atof(right.c_str());
    if(left == "mnum")      mnum   = (int)atof(right.c_str());
}

return(state);
}

double BHV_ArrayAngle::getRelevance(double osX, double osY,
    double cnX, double cnY)
{
    bool silent = true;

    double total_range = range_max-range_min;

    double dist = hypot((osX - cnX), (osY - cnY));

    if(!silent)
        cout << "BHV_ArrayAngle: Current Distance -----" << dist << endl;

    if(dist > range_max)
        return(0.1);
    if(dist < range_min)
        return(1.0);

    double val = 1.0-((dist - range_min) / total_range);

    if (val > 1.0)
        val = 1.0;

    if(!silent)
        cout << "relevance = " << val << endl;

    //for now always weight as 1
    return(1.0);
}

```

B.3 CloseRange Behavior

B.3.1 BHV_CloseRange.h

```
/* *****  
/*      NAME: Don Eickstedt                               */  
/*      FILE: BHV_CloseRange.h                             */  
/* *****  
  
#ifndef BHV_CLOSERANGE_HEADER  
#define BHV_CLOSERANGE_HEADER  
  
#include "IvPBehavior.h"  
  
using namespace std;  
  
class IvPDomain;  
class BHV_CloseRange : public IvPBehavior {  
  
public:  
  
    BHV_CloseRange(IvPDomain);  
    ~BHV_CloseRange() {};  
  
    IvPFunction*  produceOF();  
    bool setParam(std::string, std::string);  
  
protected:  
  
    int decode(string);  
    double getRelevance(double,double,double,double);  
  
    double heading,speed,ty,tx,mnum,range_max,range_min,init_range,curr_range;  
    int     state,meas_min;  
  
    bool range_set;  
};  
#endif
```


B.3.2 BHV_CloseRange.cpp

```

/*****
/*   NAME: Don Eickstedt
/*   FILE: BHV_CloseRange.cpp
/*   DATE: July 03 2005
*****/

#include <string>
#include <iostream>
#include <math.h>
#include "BHV_CloseRange.h"
#include "AOF_CutRangeCPA.h"
#include "MBUtils.h"
#include "OF_Reflector.h"

using namespace std;

//-----
// Procedure: Constructor

BHV_CloseRange::BHV_CloseRange(IvPDomain gdomain) :
    IvPBehavior(gdomain)
{

    this->setParam("descriptor", "(d)bhv_1BTrack");
    this->setParam("unifbox", "course=3, speed=2, tol = 2");
    this->setParam("gridbox", "course=9, speed=6, tol = 6");

    mnum = 0;
    range_set = false;

    info_vars.push_back("NAV_X");
    info_vars.push_back("NAV_Y");
    info_vars.push_back("NAV_HEADING");
    info_vars.push_back("TRACK_STAT");
}

//-----
// Procedure: setParam

bool BHV_CloseRange::setParam(string param, string val)
{

    IvPBehavior::setParamCommon(param, val);

    if(param == "meas_min") {
        meas_min = (int) atof(val.c_str());
    }
}

```

```

        return(true);
    }

    if(param == "range_min") {
        range_min = atof(val.c_str());
        return(true);
    }

    if(param == "range_max") {
        range_max = atof(val.c_str());
        return(true);
    }

    return true;
}

//-----
// Procedure: produceOF

IvPFunction *BHV_CloseRange::produceOF()
{
    messages.clear();

    // Need to know the name of ownship to query position
    if(us_name == "") {
        postEMessage("error,BHV_CloseRange: ownship name not known.");
        return(0);
    }

    bool ok1,ok2,ok3,ok4;
    //get current course
    double osCourse = info_buffer->dQuery(us_name, "NAV_HEADING", &ok1);
    //get current x
    double osX = info_buffer->dQuery(us_name, "NAV_X", &ok2);
    //get current heading
    double osY = info_buffer->dQuery(us_name, "NAV_Y", &ok3);
    //get current tracking state
    string tState = info_buffer->sQuery(us_name,"TRACK_STAT", &ok4);

    if(!ok1 || !ok2 || !ok3 ||!ok4){
        postEMessage("error,BHV_CloseRange: ownship data not available");
        return (0);
    }

    int new_state = decode(tState);

```

```

//if we don't have enough measurements, we don't want to
//close range to the target
if (mnum < meas_min)
    return(0);

//fix the initial range
if(!range_set){
    init_range = sqrt((osX-tx)*(osX-tx)+(osY-ty)*(osY-ty));
    range_set = true;
}

curr_range = sqrt((osX-tx)*(osX-tx)+(osY-ty)*(osY-ty));

double relevance = getRelevance(osX, osY, tx, ty);

if(relevance <= 0)
    return(0);

//use the CutRangeCPA AOF in the lib_behaviors-marine library
AOF_CutRangeCPA aof_range(domain, ty, tx, heading, speed, osY, osX);

OF_Reflector *ofr_range = new OF_Reflector(&aof_range,1);

ofr_range->create_uniform(unif_box,grid_box);

IvPFunction *of = ofr_range->extractOF();

//of->getPDMap()->normalize(0.0,100.0);

delete(ofr_range);

of->setDomainName(0, "course");
of->setDomainName(1, "speed");
of->setDomainName(2, "tol");

of->setPWT(relevance*priority_wt);

return(of);
}

//-----
//this method decodes the output string from the
//tracking process
//-----

int BHV_CloseRange::decode(string status)

```

```

{
    vector<string> svector;
    vector<string> svector2;

    // Parse the waypoint status string for "us"
    svector = parse_string(status, ',');
    for(unsigned int i=0; i<svector.size(); i++) {
        svector2 = parse_string(svector[i], '=');
        if(svector2.size() != 2) {
            postEMessage("error,BHV_CloseRange: Invalid waypoint string");
            return(0);
        }

        string left  = strip_blank_ends(svector2[0]);
        string right = strip_blank_ends(svector2[1]);
        if(left == "state")      state = atoi(right.c_str());
        if(left == "x")          tx    = atof(right.c_str());
        if(left == "y")          ty    = atof(right.c_str());
        if(left == "heading")    heading = atof(right.c_str());
        if(left == "speed")      speed  = atof(right.c_str());
        if(left == "mnum")       mnum   = atof(right.c_str());
    }

    return(state);
}

//-----
//this method dynamically determines the behavior weighting
//based on the input parameters and the distance to the target
//-----

double BHV_CloseRange::getRelevance(double osX, double osY,
    double cnX, double cnY)
{
    bool silent = true;

    //this gets us pointed in the right direction
    if((init_range-curr_range) < 10)
        return(1.5);

    double dist = hypot((osX - cnX), (osY - cnY));

    if(!silent)
        cout << "BHV_CloseRange: Current Distance -----" << dist << endl;

    //if we are too close, don't get any closer

```

```

    if(dist < range_min)
        return(0.0);

    double val = (dist-range_min)*.003;

    if(val < 0.0)
        val = 0.0;
    if (val > 1.0)
        val = 1.0;

    if(!silent)
        cout << "relevance = "<< val <<endl;

    return(val);
}

```

B.4 Orbit Behavior

B.4.1 AOF_DonWpt2D.h

```
/* ***** */
/*  NAME: Don Eickstedt */
/*  FILE: AOF_DonWpt2D.h */
/*  DATE: 10 April 2005 */
/* ***** */

#ifndef AOF_DONWPT2D_HEADER
#define AOF_DONWPT2D_HEADER

#include "AOF.h"
#include "IvPDomain.h"

class AOF_DonWpt2D: public AOF {

public:

    AOF_DonWpt2D(IvPDomain,double, double, double, double, double);
    ~AOF_DonWpt2D() {};

public:

    double evalBox(const IvPBox*) const;    // virtual defined

protected:

    double metric(double, double) const;

protected:

    double osLAT;    // Ownship Lat position at time Tm.
    double osLON;    // Ownship Lon position at time Tm.
    double targ_LAT; // target waypoint lat (Y)
    double targ_LON; // target waypoint lon (X)
    double dSpeed;   //desired speed;

    double crsBase;
    double crsDelta;
    double spdBase;
    double spdDelta;
    double tolBase;
    double tolDelta;
};
#endif
```

B.4.2 AOF_DonWpt2D.cpp

```

/*****
/*   NAME: Don Eickstedt
/*   FILE: AOF_DonWpt2D.cpp
/*   BORN: April 10, 200   5
*****/

#include <iostream>
#include <math.h>
#include <assert.h>
#include "AOF_DonWpt2D.h"
#include "AngleUtils.h"
#include "IvPDomain.h"

using namespace std;

//-----
// Procedure: Constructor

AOF_DonWpt2D::AOF_DonWpt2D(IvPDomain g_domain, double g_speed,
double g_osLAT, double g_osLON,
double g_targLAT, double g_targLON)
{
    int crs_ix = g_domain.getIndex("course");
    int spd_ix = g_domain.getIndex("speed");
    int tol_ix = g_domain.getIndex("tol");

    assert(crs_ix != -1);
    assert(spd_ix != -1);
    assert(tol_ix != -1);

    crsDelta = g_domain.get_ddelta(crs_ix);
    crsBase  = g_domain.get_dlow(crs_ix);
    tolBase  = g_domain.get_dlow(tol_ix);

    spdDelta = g_domain.get_ddelta(spd_ix);
    spdBase  = g_domain.get_dlow(spd_ix);
    tolBase  = g_domain.get_dlow(tol_ix);

    universe = IvPBox(3);
    universe.setPTS(0, 0, g_domain.get_dpoints(crs_ix)-1);
    universe.setPTS(1, 0, g_domain.get_dpoints(spd_ix)-1);
    universe.setPTS(2, 0, g_domain.get_dpoints(tol_ix)-1);

    osLAT = g_osLAT;
    osLON = g_osLON;
    targ_LAT = g_targLAT;
}
```

```

    targ_LON = g_targLON;
    dSpeed = g_speed;
}

//-----
// Procedure: evalBox
// Purpose: Evaluates a given <Course, Speed>
//           and returns a value after passing it thru the
//           metric() function.

double AOF_DonWpt2D::evalBox(const IvPBox *b) const
{
    double rel_angle = relAng(osLON, osLAT, targ_LON, targ_LAT);

    double evalCRS = crsBase + ((double)(b->pt(0,0)) * crsDelta);
    double evalSPD = spdBase + ((double)(b->pt(1,0)) * spdDelta);

    double crs_diff = fabs(evalCRS - rel_angle);
    if(crs_diff > 180)
        crs_diff = 360.0 - crs_diff;

    double spd_diff = fabs(dSpeed - evalSPD);

    double val = metric(crs_diff, spd_diff);

    return(val);
}

//-----
//This method applies a metric to the proposed course and speed
//-----

double AOF_DonWpt2D::metric(double crs_diff, double spd_diff) const
{
    return((180.0 - crs_diff) - (2.0*spd_diff) );
}

```


B.4.3 BHV_Orbit.h

```

/*****
/*   NAME: Don Eickstedt
/*   FILE: BHV_Orbit.h
/*
*****/

#ifndef BHV_ORBIT_HEADER
#define BHV_ORBIT_HEADER

#include <string>
#include <vector>
#include "IvPBehavior.h"

#define M_PI 3.14159265358979323846

using namespace std;

class BHV_Orbit : public IvPBehavior {

public:

    BHV_Orbit(IvPDomain);
    ~BHV_Orbit() {};

    IvPFunction* produceOF();
    bool        setParam(std::string, std::string);
    void make_orbit(double x,double y);
    int decode(string);
    bool preCheck();

protected:

    std::vector<double> x_waypt;
    std::vector<double> y_waypt;
    std::vector<std::string> tag_waypt;

    bool orbit_set;
    int  current_waypt,orbit_pieces;
    double arrival_radius;
    double orbit_radius;
    double cenx,ceny;
    double cruise_speed;
};
#endif
```

B.4.4 BHV_Orbit.cpp

```

/*****
/*   NAME: Don Eickstedt
/*   FILE: BHV_Orbit.cpp
/*   DATE:
*****/
#include <string>
#include <iostream>
#include <math.h>
#include <assert.h>
#include "BHV_Orbit.h"
#include "AOF_DonWpt2D.h"
#include "OF_Reflector.h"
#include "IvPDomain.h"
#include "MBUtils.h"
#include "AngleUtils.h"

using namespace std;

//-----
// Procedure: Constructor

BHV_Orbit::BHV_Orbit(IvPDomain gdomain) :
    IvPBehavior(gdomain)
{
    this->setParam("descriptor", "(d)bhv_Orbit");
    this->setParam("unifbox", "course=2, speed=2, tol=2");
    this->setParam("gridbox", "course=8, speed=6, tol=6");

    current_waypt = 0;
    arrival_radius = 7; // Meters
    cruise_speed = 0; // Meters/second
    orbit_set = false;
    silent = true;

    info_vars.push_back("NAV_X");
    info_vars.push_back("NAV_Y");
    info_vars.push_back("NAV_SPEED");
}

//-----

bool BHV_Orbit::preCheck()
{
    if(!checkConditions()){
        orbit_set = false;
        return(false);
    }
}
```

```

    }
    return(true);
}

//-----
// Procedure: setParam

bool BHV_Orbit::setParam(string param, string val)
{
    if(IvPBehavior::setParamCommon(param, val))
        return(true);

    if(param == "orbcen") {
        vector<string> svector = parse_string(val, ',');
        cenx = atof(svector[0].c_str());
        ceny = atof(svector[1].c_str());
        return(true);
    }
    if(param == "orbrad") {
        orbit_radius = atof(val.c_str());
        return(true);
    }
    if(param == "pieces") {
        orbit_pieces = (int) atof(val.c_str());
        return(true);
    }
    if(param == "speed") {
        cruise_speed = atof(val.c_str());
        return(true);
    }
    if(param == "radius") {
        arrival_radius = atof(val.c_str());
        return(true);
    }
    return(false);
}

//-----
// Procedure: produceOF

IvPFunction *BHV_Orbit::produceOF()
{
    // clear each time produceOF() is called
    messages.clear();
}

```

```

if(!unif_box || !grid_box) {
    postEMessage("Null UnifBox or GridBox.");
    return(0);
}

if(!silent) cout << "+++++BHV_Waypoint::produceOF() " << endl;

// Need to know the name of ownship to query position
if(us_name == "") {
    postEMessage("ownship name not known.");
    return(0);
}

bool ok;
double osX   = info_buffer->dQuery(us_name, "NAV_X",      &ok);
double osY   = info_buffer->dQuery(us_name, "NAV_Y",      &ok);
double osSPD = info_buffer->dQuery(us_name, "NAV_SPEED",  &ok);

if(!silent) cout << "  osX:" << osX << " osY:" << osY << endl;

// Must get ownship position from WorldModel
if(!ok) {
    postEMessage("No ownship info in worldmodel.");
    return(0);
}

//create the orbit waypoints if not already done
if(!orbit_set)
    make_orbit(osX,osY);

// If at the end of the waypoint list, start over
if(current_waypt >= x_waypt.size())
    current_waypt = 0;

double ptX = x_waypt[current_waypt];
double ptY = y_waypt[current_waypt];

//if we've arrived at a waypoint
if(hypot((osX-ptX),(osY-ptY)) < arrival_radius) {
    //if there are more waypoints
    if(current_waypt < (x_waypt.size()-1)) {
        current_waypt++;
        ptX = x_waypt[current_waypt];
        ptY = y_waypt[current_waypt];
    }
    else{

```

```

        //start over at beginning of wp list
        current_waypt=0;
        ptX = x_waypt[current_waypt];
        ptY = y_waypt[current_waypt];
    }
}

if(!silent) cout << "  ptX:" << ptX << " ptY:" << ptY << endl;

AOF_DonWpt2D *aof_wpt = new AOF_DonWpt2D(domain, cruise_speed, osY, osX, ptY, ptX);
OF_Reflector *ofr_wpt = new OF_Reflector(aof_wpt, 1); // 1 indicates pcwise linear

ofr_wpt->create_uniform(unif_box, grid_box);
IvPFunction *of = ofr_wpt->extractOF();

delete(ofr_wpt);

of->setDomainName(0, "course");
of->setDomainName(1, "speed");
of->setDomainName(2, "tol");
of->setPWT(priority_wt);

if(!silent) {
    IvPBox mpt = of->getPDMap()->getGrid()->getMaxPt();
    cout << "BHV_Orbit::produceOF():" << endl;
    cout << "maxpt:" << endl;
    mpt.print();
}

double dist_meters = hypot((osX-ptX), (osY-ptY));
double eta_seconds = dist_meters / osSPD;

if (!silent) cout << "dist " << dist_meters << " eta " << eta_seconds << endl;

string stat = "vname=" + us_name + ",";
stat += "index=" + int_to_string(current_waypt) + ",";
stat += "dist=" + double_to_string(dist_meters) + ",";
stat += "eta=" + double_to_string(eta_seconds);

// This one is for us to read
postMessage("VEHICLE_WPT_STAT", stat);

// This one gets bridged to other vehicles perhaps
postMessage("VEHICLE_WPT_STAT_US", stat);

return(of);
}

```

```

//-----
//this method dynamically creates a list of waypoints to
//form the orbit about the required orbit center
//-----
void BHV_Orbit::make_orbit(double x, double y)
{
    double theta_inc,reverse_angle,first_wp_rads,cen_dist;
    char tag [5];
    bool silent;

    silent = false;

    x_waypt.clear();
    y_waypt.clear();
    tag_waypt.clear();
    current_waypt = 0;

    //radians between waypoints
    theta_inc = (2*M_PI)/orbit_pieces;

    //find distance between the orbit center and current position
    cen_dist = sqrt(pow((x-cenx),2) + pow((y-ceny),2));

    //if we are inside the orbit radius
    if (cen_dist < orbit_radius)
        //choose North arbitrarily
        reverse_angle = (M_PI/2.0);
    else
        //find the reverse angle from orbcen to current position
        reverse_angle = atan2((y-ceny),(x-cenx));

    if (!silent) cout << "reverse angle = " << reverse_angle << " theta_inc = " << theta_inc <<

    //increment first wp to get smoother entry
    //clockwise
    first_wp_rads = reverse_angle - theta_inc;

    //step through the angles and create waypoints
    for (int i = 0; i < orbit_pieces; i++)
    {
        x_waypt.push_back(cenx + orbit_radius*cos(-(i*theta_inc)+first_wp_rads));
        y_waypt.push_back(ceny + orbit_radius*sin(-(i*theta_inc)+first_wp_rads));
        sprintf(tag,"%d",i);
        tag_waypt.push_back((string) tag);
        if (!silent) cout << "waypoint " << i << ": " << x_waypt[i] << " " << y_waypt[i] << endl;
    }
}

```

```
    orbit_set = true;  
}
```

B.5 Formation Behavior

B.5.1 BHV_2VAngle.h

```

/*****
/*   NAME: Don Eickstedt
/*   FILE: BHV_2VAngle.h
*****/

#ifndef BHV_2VAngle_HEADER
#define BHV_2VAngle_HEADER

#include "IvPBehavior.h"

#define NO_TRACK 0
#define LR_TRACK 1
#define TRACKING 2

using namespace std;

class IvPDomain;
class BHV_2VAngle : public IvPBehavior {
public:
    BHV_2VAngle(IvPDomain);
    ~BHV_2VAngle() {};

    IvPFunction* produceOF();
    bool setParam(std::string, std::string);

protected:

    int decode(string);
    double getRelevance(double,double,double,double);

    double heading,speed,ty,tx,mnum,range_max,range_min,init_range,curr_range;
    int state,meas_min,new_state,sign;
};
#endif
```


B.6 Formation Behavior

B.6.1 BHV_2VAngle.cpp

```

/*****
/*   NAME: Don Eickstedt
/*   FILE: BHV_2VAngle.cpp
/*   DATE: July 03 2005
*****/

#include <string>
#include <iostream>
#include <math.h>
#include "BHV_2VAngle.h"
#include "AOF_WPT3D.h"
#include "AOF_Shadow.h"
#include "MBUtils.h"
#include "OF_Reflector.h"
#include "BuildUtils.h"

using namespace std;

//-----
// Procedure: Constructor

BHV_2VAngle::BHV_2VAngle(IvPDomain gdomain) :
    IvPBehavior(gdomain)
{
    this->setParam("descriptor", "(d)bhv_2VAngle");
    this->setParam("unifbox", "course=3, speed=2, tol = 2");
    this->setParam("gridbox", "course=9, speed=6, tol = 6");

    domain = subDomain(domain, "course,speed,tol");

    range_min = 0;
    sign = 1;

    info_vars.push_back("NAV_X");
    info_vars.push_back("NAV_Y");
    info_vars.push_back("NAV_HEADING");
    info_vars.push_back("TRACK_STAT");
    info_vars.push_back("V2_X");
    info_vars.push_back("V2_Y");
}

//-----
// Procedure: setParam
```

```

bool BHV_2VAngle::setParam(string param, string val)
{
    IvPBehavior::setParamCommon(param, val);

    if(param == "meas_min") {
        meas_min = (int) atof(val.c_str());
        return(true);
    }

    if(param == "range_min") {
        range_min = atof(val.c_str());
        return(true);
    }

    if(param == "range_max") {
        range_max = atof(val.c_str());
        return(true);
    }

    if(param == "sign") {
        sign= (int) atof(val.c_str());
        return(true);
    }

    return true;
}

//-----
// Procedure: produceOF

IvPFunction *BHV_2VAngle::produceOF()
{
    // Need to know the name of ownship to query position
    if(us_name == "") {
        postEMessage("error,BHV_CloseRange: ownship name not known.");
        return(0);
    }

    bool ok1,ok2,ok3,ok4,ok5,ok6;
    //get current course
    double osCourse = info_buffer->dQuery(us_name, "NAV_HEADING", &ok1);
    //get current x
    double osX = info_buffer->dQuery(us_name, "NAV_X", &ok2);
    //get current heading
    double osY = info_buffer->dQuery(us_name, "NAV_Y", &ok3);

```

```

//get current tracking state
string tState = info_buffer->sQuery(us_name,"TRACK_STAT", &ok4);

double v2_x = info_buffer->dQuery(us_name,"V2_X",&ok5);
double v2_y = info_buffer->dQuery(us_name,"V2_Y",&ok6);

if(!ok1 || !ok2 || !ok3 ||!ok4||!ok5||!ok6){
    postEMessage("error,BHV_2VAngle: buffer data not available");
    return (0);
}

new_state = decode(tState);

double relevance = getRelevance(osX, osY, tx, ty);

if(relevance <= 0)
    return(0);

double angle1 = atan2(ty-osY,tx-osX);
double angle2 = atan2(ty-v2_y,tx-v2_x);

double sep_angle = angle1-angle2;

if (sep_angle < -180.0)
    sep_angle += 360.0;

if (sep_angle > 180.0)
    sep_angle = 360.0 - sep_angle;

sep_angle = fabs(sep_angle)*180.0/M_PI;

double ang = 90.0 - (heading + (sign)* 135.0);

double d_x = 50.0*cos(ang*M_PI/180.0) + tx;
double d_y = 50.0*sin(ang*M_PI/180.0) + ty;

double d_r = sqrt((d_x-osX)*(d_x-osX) + (d_y-osY)*(d_y-osY));

//if we are within 5 meters of our desired location just shadow
if(d_r < 5.0){

    AOF_Shadow aof(domain);
    aof.setParam("cn_crs", heading);
    aof.setParam("cn_spd", speed);
    aof.initialize();
}

```

```

    OF_Reflector reflector(&aof,1);

    reflector.create_uniform(unif_box,grid_box);

    IvPFunction *of = reflector.extractOF();

    of->setPWT(relevance*priority_wt);
    return(of);
}
//otherwise move to the desired location
else{

    AOF_WPT3D aof(domain);
    aof.setParam("oslat", osY);
    aof.setParam("oslon", osX);
    aof.setParam("ptlat", d_y);
    aof.setParam("ptlon", d_x);
    aof.setParam("desired_speed", 3.0);
    aof.initialize();

    OF_Reflector reflector(&aof,1);

    reflector.create_uniform(unif_box,grid_box);

    IvPFunction *of = reflector.extractOF();

    of->setPWT(relevance*priority_wt);
    return(of);
}

//return(of);
}

int BHV_2VAngle::decode(string status)
{
    vector<string> svector;
    vector<string> svector2;

    // Parse the waypoint status string for "us"
    svector = parse_string(status, ',');
    for(unsigned int i=0; i<svector.size(); i++) {
        svector2 = parse_string(svector[i], '=');
        if(svector2.size() != 2) {
            postEMessage("error,BHV_CloseRange: Invalid waypoint string");
            return(0);
        }
    }
}

```

```

    }

    string left  = strip_blank_ends(svector2[0]);
    string right = strip_blank_ends(svector2[1]);
    if(left == "state")      state = atoi(right.c_str());
    if(left == "x")          tx     = atof(right.c_str());
    if(left == "y")          ty     = atof(right.c_str());
    if(left == "heading")    heading = atof(right.c_str());
    if(left == "speed")      speed  = atof(right.c_str());
    if(left == "mnum")       mnum   = atof(right.c_str());
}

    return(state);
}

double BHV_2VAngle::getRelevance(double osX, double osY,
    double cnX, double cnY)
{
    bool silent = true;

    //always fully relevant
    return(1.0);
}

```


Appendix C

Sample Mission Files

C.1 MOOS Files

C.1.1 December 4, Mission 1507 - Sensor Vehicle 200

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% LOG FILE:      ../data_from_runs/Kayak_200_4_12_2005_____15_07._moos
%% FILE OPENED ON Sun Dec  4 15:07:26 2005
%% LOGSTART      1133726846.18
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
//*****
// Configuration file: M. Benjamin
//
// Note: level0 vs. level2  all references to machines names
//       in this file would be replaced by "localhost" in
//       the level2 counterpart. Also, iPWMController, iGPS
//       and iPNICController are not started up in level2.
//       And level2 would launch iMarineSim.
//
// Note: level0 vs. level1  iMarineSim is run in the level1
//       counterpart. In level1, iGPS and iPNICCompass are
//       not run.

ServerHost = 192.168.0.200
ServerPort = 9000
Simulator  = false

// Community name IS the vehicle name
Community  = nyak200

LatOrigin  = 42.3584
LongOrigin = -71.08745

//-----
```

```

// Antler configuration block

ProcessConfig = ANTLER
{
    MSBetweenLaunches = 200

    //crucial processes
    Run = MOOSDB @ NewConsole = false
    Run = iGPS @ NewConsole = false
    Run = iPNICompass @ NewConsole = false
    Run = iPWMController @ NewConsole = false
    Run = iMetaCompass @ NewConsole = false
    Run = pMOOSBridge @ NewConsole = false ~ pMOOSBridge_200
    Run = pNav @ NewConsole = false
    Run = pLogger @ NewConsole = false

    Run = pBearings @ NewConsole = false
    Run = p1BTracker @ NewConsole = false

    //Run = MOOSDump @ NewConsole = false
    //Run = pHelmIvP @ NewConsole = false
    //Run = iRemote @ NewConsole = false
}

//-----
// pMOOSBridge config block

ProcessConfig = pMOOSBridge_200
{
    AppTick = 2
    CommsTick = 2

    // SHARE = [VAR] -> to-community @ to-host:to-port [VAR]

    SHARE = [HELM_SUMMARY] -> marineviewer055 @ 192.168.0.55:9055 [HELM_SUMMARY_VIEW]
    SHARE = [TRACK_STAT,TRACKING_SIGNAL,TRACK_CONTROL] -> nyak206 @ 192.168.0.206:9000 [TRACK_S'
}

//-----
// p1BTracker config block

ProcessConfig = p1BTracker
{
    AppTick = 4
    CommsTick = 4
    range_guess = 200
    max_measurements = 150
}

```



```

}

//-----
// pBearings config block

ProcessConfig = pBearings
{
    AppTick      = 4
    CommsTick    = 4
    sigma       = 0.0 //degrees
}

//-----
// pHelmIvP config block
// Note: pHelmIvP must know the vehicle name. pHelmIvP will look
//       for the global line "Community = name" in the .moos file.

ProcessConfig = pHelmIvP
{
    AppTick      = 4
    CommsTick    = 4

    Domain       = course,0:359:360
    Domain       = speed,0:3:16
    Domain       = tol,1:45:15

    //IF BELOW IS COMMENTED OUT - BHV FILE IS GIVEN AS COMMAND LINE ARG
    //Behaviors = foobar.bhv

    // Yaw PID controller
    YAW_PID_KP   = 0.4
    YAW_PID_KD   = 0.03
    YAW_PID_KI   = 0.0
    YAW_PID_INTEGRAL_LIMIT = 0.07

    // Speed PID controller
    SPEED_PID_KP = 0.5
    SPEED_PID_KD = 0.0
    SPEED_PID_KI = 0.0
    SPEED_PID_INTEGRAL_LIMIT = 0.07

    // Setting LOGPATH will cause PID data to be logged
    LOGPATH = ../data_from_runs/

    // A non-zero SPEED_FACTOR overrides use of SPEED_PID

```

```

// Will set DESIRED_THRUST = DESIRED_SPEED * SPEED_FACTOR
SPEED_FACTOR    = 30

META_COMPASS    = true
}

//-----
// iRemote configuration block

ProcessConfig = iRemote
{
    CustomKey = 1 : HELM_VERBOSE @ "verbose"
    CustomKey = 2 : HELM_VERBOSE @ "terse"
    CustomKey = 3 : HELM_VERBOSE @ "quiet"
}

//-----
// Logger configuration block

ProcessConfig = pLogger
{
    //over loading basic params...
    AppTick    = 5.0
    CommsTick  = 5.0

    File = Kayak_200
    PATH = ../data_from_runs/
    SyncLog   = true @ 0.2
    AsyncLog   = true
    FileTimeStamp = true

    Log = DESIRED_THRUST @ 0.1
    Log = DESIRED_RUDDER @ 0.1
    Log      = NAV_X      @ 0.1
    Log      = NAV_Y      @ 0.1
    Log      = NAV_YAW    @ 0.1
    Log      = NAV_SPEED  @ 0.1
    Log      = GPS_X      @ 0.1
    Log      = GPS_Y      @ 0.1
    Log      = GPS_SPEED  @ 0.1
    Log      = GPS_HEADING @ 0.1
    Log      = GPS_TIME   @ 0.1
    Log      = COMPASS_HEADING @ 0.1
    log      = LOOP_CPU   @ 0.1
    Log      = VEHICLE_WPT_INDEX @ 0.1
    Log      = META_HEADING @ 0.1
    Log      = META_OFFSET @ 0.1

```

```

Log          = META_SOURCE @ 0.1
Log          = MICROMODEM_RAW @ 0.1
Log          = MICROMODEM_SERVICE @ 0.1
Log          = MICROMODEM_COMMAND @ 0.1
Log          = MICROMODEM_STATUS @ 0.1
Log          = MICROMODEM_DATA @ 0.1
Log          = MOW_TARGET @ 0.1
Log          = MOW_HISTORY @ 0.1
Log          = MOW_QUALITY @ 0.1
Log          = BHV_RESOLVE_ACTIVE @ 0.1
Log          = TIME_SINCE_UPGRADE @ 0.1
Log          = BHV_RESOLVE_WT @ 0.1
Log          = BHV_SENTRY_WT @ 0.1
Log          = TRACK_STAT @ 0.1
Log          = BEARING_STAT @ 0.1
Log          = TARGET_X @ 0.1
Log          = TARGET_Y @ 0.1
Log          = TARGET_XPOS @ 0.1
Log          = TARGET_YPOS @ 0.1
Log          = TARGET_HEADING @ 0.1
Log          = TARGET_SPEED @ 0.1
Log          = TRACKING_SIGNAL @ 0.1
Log          = DESIRED_SPEED @ 0.1
Log          = DESIRED_HEADING @ 0.1
}

```

```

//-----
// iPWMController configuration block

```

```

ProcessConfig = iPWMController
{
    AppTick      = 4
    CommsTick    = 4
    Port         = /dev/ttyS0
    ThrustPWM     = 6
    RudderPWM     = 7
    Timeout      = 10
    RudderOffset = 0
}

```

```

//-----
// pNav configuration block

```

```

ProcessConfig = pNav
{
    AppTick      = 5
    CommsTick    = 5
}

```

```

X = GPS @ 5.0
Y = GPS @ 5.0
Yaw = Compass @ 5.0, GPS @ 5.0
Speed = GPS @ 5.0
Z =

////////////////////////////////////
//      FILTER CONTROL:
////////////////////////////////////

UseLSQ = false
UseEKF = false
UseTOPDOWN = false

FixedDepth = 0 @ 0.1
}

//-----
// iGPS configuration block

ProcessConfig = iGPS
{
    AppTick    = 10
    CommsTick  = 10
    Port       = /dev/ttyS1
    BaudRate   = 4800
    Streaming  = true
    Verbose    = true
    Type       = GARMIN
}

//-----
// iPNICompass configuration block

ProcessConfig = iPNICompass
{
    AppTick      = 4
    CommsTick    = 4
    Port         = /dev/ttyS3
    Type         = V2Xe
    Speed        = 9600
    PreRotation  = -90
}

//-----
// iMetaCompass configuration block

```

```

ProcessConfig = iMetaCompass
{
    AppTick      = 5
    CommsTick    = 5
    SpeedThresh  = 0.2
}

```

C.1.2 December 4, Mission 1507 - Target Vehicle 201

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% LOG FILE:      ../data_from_runs/Kayak_201_4_12_2005_15_15._moos
%% FILE OPENED ON Sun Dec  4 15:15:11 2005
%% LOGSTART      1133727310.42
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
//*****
// Configuration file: M. Benjamin
//
// Note: level0 vs. level2  all references to machines names
//       in this file would be replaced by "localhost" in
//       the level2 counterpart. Also, iPWMController, iGPS
//       and iPNICController are not started up in level2.
//       And level2 would launch iMarineSim.
//
// Note: level0 vs. level1  iMarineSim is run in the level1
//       counterpart. In level1, iGPS and iPNICCompass are
//       not run.

ServerHost = 192.168.0.201
ServerPort = 9000
Simulator  = false

// Community name IS the vehicle name
Community  = nyak201

LatOrigin  = 42.3584
LongOrigin = -71.08745

//-----
// Antler configuration block

ProcessConfig = ANTLER
{
    MSBetweenLaunches = 200

    //crucial processes
    Run = MOOSDB @ NewConsole = false

```

```

Run = iGPS @ NewConsole = false
Run = iPNCompass @ NewConsole = false
Run = iPWMController @ NewConsole = false
Run = iMetaCompass @ NewConsole = false
Run = pMOOSBridge @ NewConsole = false ~ pMOOSBridge_201
Run = pNav @ NewConsole = false
Run = pLogger @ NewConsole = false

Run = pTarget @ NewConsole = false

//Run = MOOSDump @ NewConsole = false
//Run = pHelmIvP @ NewConsole = false
//Run = iRemote @ NewConsole = false
}

//-----
// pMOOSBridge config block

ProcessConfig = pMOOSBridge_201
{
    AppTick = 1
    CommsTick = 1

    // SHARE = [VAR] -> to-community @ to-host:to-port [VAR]

    SHARE = [HELM_SUMMARY] -> marineviewer055 @ 192.168.0.55:9055 [HELM_SUMMARY_VIEW]

    SHARE = [TARGET_XPOS,TARGET_YPOS] ->nyak200 @ 192.168.0.200:9000 [TARGET_XPOS,TARGET_YPOS]
    //SHARE = [TARGET_XPOS,TARGET_YPOS] ->nyak206 @ 192.168.0.206:9000 [TARGET_XPOS,TARGET_YPOS]
}

//-----
// pTarget config block

ProcessConfig = pTarget
{
    AppTick = 4
    CommsTick = 4
    //polygon = -200,-25:200,-25:200,-400:-200,-400
    polygon = 80,-10:269,37:360,-400:120,-240
}

//-----
// pHelmIvP config block
// Note: pHelmIvP must know the vehicle name. pHelmIvP will look
// for the global line "Community = name" in the .moos file.

```

```

ProcessConfig = pHelmIvP
{
    AppTick      = 4
    CommsTick    = 4

    Domain       = course,0:359:360
    Domain       = speed,0:3:16
    Domain       = tol,1:45:15

    //IF BELOW IS COMMENTED OUT - BHV FILE IS GIVEN AS COMMAND LINE ARG
    //Behaviors = foobar.bhv

    // Yaw PID controller
    YAW_PID_KP   = 0.4
    YAW_PID_KD   = 0.03
    YAW_PID_KI   = 0.0
    YAW_PID_INTEGRAL_LIMIT = 0.07

    // Speed PID controller
    SPEED_PID_KP = 0.5
    SPEED_PID_KD = 0.0
    SPEED_PID_KI = 0.0
    SPEED_PID_INTEGRAL_LIMIT = 0.07

    // Setting LOGPATH will cause PID data to be logged
    LOGPATH = ../data_from_runs/

    // A non-zero SPEED_FACTOR overrides use of SPEED_PID
    // Will set DESIRED_THRUST = DESIRED_SPEED * SPEED_FACTOR
    SPEED_FACTOR = 30

    META_COMPASS = true
}

//-----
// iRemote configuration block

ProcessConfig = iRemote
{
    CustomKey = 1 : HELM_VERBOSE @ "verbose"
    CustomKey = 2 : HELM_VERBOSE @ "terse"
    CustomKey = 3 : HELM_VERBOSE @ "quiet"
}

//-----
// Logger configuration block

```

```

ProcessConfig = pLogger
{
    //over loading basic params...
    AppTick    = 5.0
    CommsTick  = 5.0

    File = Kayak_201
    PATH = ../data_from_runs/
    SyncLog   = true @ 0.2
    AsyncLog  = true
    FileTimeStamp = true

    Log = DESIRED_THRUST @ 0.1
    Log = DESIRED_RUDDER @ 0.1
    Log      = NAV_X    @ 0.1
    Log      = NAV_Y    @ 0.1
    Log      = NAV_YAW @ 0.1
    Log      = NAV_SPEED @ 0.1
    Log      = GPS_X    @ 0.1
    Log      = GPS_Y    @ 0.1
    Log      = GPS_SPEED @ 0.1
    Log      = GPS_HEADING @ 0.1
    Log      = GPS_TIME @ 0.1
    Log      = COMPASS_HEADING @ 0.1
    log      = LOOP_CPU @ 0.1
    Log      = VEHICLE_WPT_INDEX @ 0.1
    Log      = META_HEADING @ 0.1
    Log      = META_OFFSET  @ 0.1
    Log      = META_SOURCE  @ 0.1
    Log      = MICROMODEM_RAW @ 0.1
    Log      = MICROMODEM_SERVICE @ 0.1
    Log      = MICROMODEM_COMMAND @ 0.1
    Log      = MICROMODEM_STATUS @ 0.1
    Log      = MICROMODEM_DATA @ 0.1
    Log      = MOW_TARGET @ 0.1
    Log      = MOW_HISTORY @ 0.1
    Log      = MOW_QUALITY @ 0.1
    Log      = BHV_RESOLVE_ACTIVE @ 0.1
    Log      = TIME_SINCE_UPGRADE @ 0.1
    Log      = BHV_RESOLVE_WT @ 0.1
    Log      = BHV_SENTRY_WT @ 0.1
    Log      = TRACK_STAT @ 0.1
    Log      = BEARING_STAT @ 0.1
    Log      = TARGET_X @ 0.1
    Log      = TARGET_Y @ 0.1
    Log      = TARGET_XPOS @ 0.1

```



```

    Log          = TARGET_YPOS @ 0.1
    Log          = TARGET_HEADING @ 0.1
    Log          = TARGET_SPEED @ 0.1
    Log          = TRACKING_SIGNAL @ 0.1
    Log          = DESIRED_SPEED @ 0.1
    Log          = DESIRED_HEADING @ 0.1
}

//-----
// iPWMController configuration block

ProcessConfig = iPWMController
{
    AppTick      = 4
    CommsTick    = 4
    Port         = /dev/ttyS0
    ThrustPWM     = 6
    RudderPWM    = 7
    Timeout      = 10
    RudderOffset = 0
}

//-----
// pNav configuration block

ProcessConfig = pNav
{
    AppTick      = 5
    CommsTick    = 5

    X = GPS @ 5.0
    Y = GPS @ 5.0
    Yaw = Compass @ 5.0, GPS @ 5.0
    Speed = GPS @ 5.0
    Z =

    //////////////////////////////////////
    //      FILTER CONTROL:
    //////////////////////////////////////

    UseLSQ = false
    UseEKF = false
    UseTOPDOWN = false

    FixedDepth = 0 @ 0.1
}

```

```

//-----
// iGPS configuration block

ProcessConfig = iGPS
{
    AppTick    = 10
    CommsTick  = 10
    Port       = /dev/ttyS1
    BaudRate   = 4800
    Streaming  = true
    Verbose    = true
    Type       = GARMIN
}

//-----
// iPNICompass configuration block

ProcessConfig = iPNICompass
{
    AppTick      = 4
    CommsTick    = 4
    Port         = /dev/ttyS3
    Type         = V2Xe
    Speed        = 9600
    PreRotation  = -90
}

//-----
// iMetaCompass configuration block

ProcessConfig = iMetaCompass
{
    AppTick      = 5
    CommsTick    = 5
    SpeedThresh  = 0.2
}

```

C.1.3 December 4, Mission 1507 - Classification Vehicle 206

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% LOG FILE:                ../data_from_runs/Kayak_206_4_12_2005_15_08._moos
%% FILE OPENED ON   Sun Dec  4 15:08:09 2005
%% LOGSTART                1133726887.59
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
//*****
// Configuration file: M. Benjamin
//
// Note: level0 vs. level2  all references to machines names

```

```
//          in this file would be replaced by "localhost" in
//          the level2 counterpart. Also, iPWMController, iGPS
//          and iPNICController are not started up in level2.
//          And level2 would launch iMarineSim.
//
// Note: level0 vs. level1  iMarineSim is run in the level1
//          counterpart. In level1, iGPS and iPNICCompass are
//          not run.
```

```
ServerHost = 192.168.0.206
ServerPort = 9000
Simulator  = false
```

```
// Community name IS the vehicle name
Community  = nyak206
```

```
LatOrigin  = 42.3584
LongOrigin = -71.08745
```

```
//-----
// Antler configuration block
```

```
ProcessConfig = ANTLER
{
    MSBetweenLaunches = 200

    //crucial processes
    Run = MOOSDB @ NewConsole = false
    Run = iGPS @ NewConsole = false
    Run = iPNICCompass @ NewConsole = false
    Run = iPWMController @ NewConsole = false
    Run = iMetaCompass   @ NewConsole = false
    Run = pMOOSBridge    @ NewConsole = false ~ pMOOSBridge_206
    Run = pNav @ NewConsole = false
    Run = pLogger @ NewConsole = false

    Run = pExtTracker @ NewConsole = false

    //Run = MOOSDump @ NewConsole = false
    //Run = pHelmIvP @ NewConsole = false
    //Run = iRemote @ NewConsole = false
}
```

```
//-----
// pMOOSBridge config block
```

```

ProcessConfig = pMOOSBridge_206
{
    AppTick    = 2
    CommsTick  = 2
    // SHARE = [VAR] -> to-community @ to-host:to-port [VAR]

    SHARE = [HELM_SUMMARY] -> marineviewer055 @ 192.168.0.55:9055 [HELM_SUMMARY_VIEW]
}

//-----
// pExtTracker config block

ProcessConfig = pExtTracker
{
    AppTick    = 4
    CommsTick  = 4
    initial_state = 2
}

//-----
// pHelmIvP config block
// Note: pHelmIvP must know the vehicle name. pHelmIvP will look
//       for the global line "Community = name" in the .moos file.

ProcessConfig = pHelmIvP
{
    AppTick    = 4
    CommsTick  = 4

    Domain     = course,0:359:360
    Domain     = speed,0:3:16
    Domain     = tol,1:45:15

    //IF BELOW IS COMMENTED OUT - BHV FILE IS GIVEN AS COMMAND LINE ARG
    //Behaviors = foobar.bhv

    // Yaw PID controller
    YAW_PID_KP  = 0.4
    YAW_PID_KD  = 0.03
    YAW_PID_KI  = 0.0
    YAW_PID_INTEGRAL_LIMIT = 0.07

    // Speed PID controller
    SPEED_PID_KP  = 0.5
    SPEED_PID_KD  = 0.0
    SPEED_PID_KI  = 0.0

```

```

SPEED_PID_INTEGRAL_LIMIT = 0.07

// Setting LOGPATH will cause PID data to be logged
LOGPATH = ../data_from_runs/

// A non-zero SPEED_FACTOR overrides use of SPEED_PID
// Will set DESIRED_THRUST = DESIRED_SPEED * SPEED_FACTOR
SPEED_FACTOR = 30

META_COMPASS = true
}

//-----
// iRemote configuration block

ProcessConfig = iRemote
{
    CustomKey = 1 : HELM_VERBOSE @ "verbose"
    CustomKey = 2 : HELM_VERBOSE @ "terse"
    CustomKey = 3 : HELM_VERBOSE @ "quiet"
}

//-----
// Logger configuration block

ProcessConfig = pLogger
{
    //over loading basic params...
    AppTick = 5.0
    CommsTick = 5.0

    File = Kayak_206
    PATH = ../data_from_runs/
    SyncLog = true @ 0.2
    AsyncLog = true
    FileTimeStamp = true

    Log = DESIRED_THRUST @ 0.1
    Log = DESIRED_RUDDER @ 0.1
    Log = NAV_X @ 0.1
    Log = NAV_Y @ 0.1
    Log = NAV_YAW @ 0.1
    Log = NAV_SPEED @ 0.1
    Log = GPS_X @ 0.1
    Log = GPS_Y @ 0.1
    Log = GPS_SPEED @ 0.1
    Log = GPS_HEADING @ 0.1

```

```

Log          = GPS_TIME @ 0.1
Log          = COMPASS_HEADING @ 0.1
log          = LOOP_CPU @ 0.1
Log          = VEHICLE_WPT_INDEX @ 0.1
Log          = META_HEADING @ 0.1
Log          = META_OFFSET @ 0.1
Log          = META_SOURCE @ 0.1
Log          = MICROMODEM_RAW @ 0.1
Log          = MICROMODEM_SERVICE @ 0.1
Log          = MICROMODEM_COMMAND @ 0.1
Log          = MICROMODEM_STATUS @ 0.1
Log          = MICROMODEM_DATA @ 0.1
Log          = MOW_TARGET @ 0.1
Log          = MOW_HISTORY @ 0.1
Log          = MOW_QUALITY @ 0.1
Log          = BHV_RESOLVE_ACTIVE @ 0.1
Log          = TIME_SINCE_UPGRADE @ 0.1
Log          = BHV_RESOLVE_WT @ 0.1
Log          = BHV_SENTRY_WT @ 0.1
Log          = TRACK_STAT @ 0.1
Log          = BEARING_STAT @ 0.1
Log          = TARGET_X @ 0.1
Log          = TARGET_Y @ 0.1
Log          = TARGET_XPOS @ 0.1
Log          = TARGET_YPOS @ 0.1
Log          = TARGET_HEADING @ 0.1
Log          = TARGET_SPEED @ 0.1
Log          = TRACKING_SIGNAL @ 0.1
Log          = DESIRED_SPEED @ 0.1
Log          = DESIRED_HEADING @ 0.1
}

//-----
// iPWMController configuration block

ProcessConfig = iPWMController
{
    AppTick      = 4
    CommsTick    = 4
    Port         = /dev/ttyS0
    ThrustPWM     = 6
    RudderPWM    = 7
    Timeout      = 10
    RudderOffset = 0
}

//-----

```

```

// pNav configuration block

ProcessConfig = pNav
{
    AppTick    = 5
    CommsTick  = 5

    X = GPS @ 5.0
    Y = GPS @ 5.0
    Yaw = Compass @ 5.0, GPS @ 5.0
    Speed = GPS @ 5.0
    Z =

    //////////////////////////////////////
    //      FILTER CONTROL:
    //////////////////////////////////////

    UseLSQ = false
    UseEKF = false
    UseTOPDOWN = false

    FixedDepth = 0 @ 0.1
}

//-----
// iGPS configuration block

ProcessConfig = iGPS
{
    AppTick    = 10
    CommsTick  = 10
    Port       = /dev/ttyS1
    BaudRate   = 19200
    Streaming  = true
    Verbose    = true
    Type       = GARMIN
}

//-----
// iPNICompass configuration block

ProcessConfig = iPNICompass
{
    AppTick      = 4
    CommsTick    = 4
    Port         = /dev/ttyS3
    Type         = V2Xe

```

```

    Speed = 9600
    PreRotation = -90
}

//-----
// iMetaCompass configuration block

ProcessConfig = iMetaCompass
{
    AppTick      = 5
    CommsTick    = 5
    SpeedThresh = 0.2
}

```

C.2 Behavior Files

C.2.1 December 4, Mission 1507 - Sensor Vehicle 200

```

// This is a behavior configuration file.
// for the one-bearing tracking sensor vehicle 200.
//
// Legal comments are anything to the right of "/"
// or anything to the right of "#"

Initial_Var = RETURN_HOME,false

//-----
Behavior = BHV_ArrayTurn
{
    name = bhv_arrayturn
    pwt = 100
    condition = TRACKING,AMBIGUOUS
    condition = :RETURN_HOME,false
}

//-----
Behavior = BHV_ArrayAngle
{
    name = bhv_arrayangle
    pwt = 100
    width = 30
    range_max = 250
    range_min = 30
    condition = TRACKING,TRACKING
    condition = :RETURN_HOME,false
}

```



```

//-----
Behavior = BHV_CloseRange
{
    name = bhv_closerange
    pwt = 110
    meas_min = 40
    range_max = 250
    range_min = 30
    condition = TRACKING,TRACKING
    condition = :RETURN_HOME,false
}

//-----
Behavior = BHV_Orbit
{
    name = bhv_orbit
    pwt = 100
    orbcen = -100,-150
    orbrad = 50
    pieces = 8
    radius = 9
    speed = 2
    condition = TRACKING,NO_TRACK
    condition = :RETURN_HOME,false
}

//-----
Behavior = BHV_Waypoint
{
    name = bhv_return
    speed = 1.4
    radius = 8.0
    points = 0,-10:
    condition = :RETURN_HOME,true
}

//-----
Behavior = BHV_OpRegion
{
    polygon = label,SafeOpBox: -280,-110:-130,-490:400,-280:250,10:80,40
    name = bhv_op_region
}

```

C.2.2 December 4, Mission 1507 - Target Vehicle 201

```
Initial_Var = RETURN_HOME,false
```

```
//-----
```

```

Behavior = BHV_Waypoint
{
    name      = bhv_waypt
    pwt       = 100
    speed     = 0.9
    radius    = 8.0
    points    = 100,0:250,-190:20,-10
    condition = :RETURN_HOME,false
}

//-----
Behavior = BHV_Waypoint
{
    name      = bhv_return
    speed     = 1.4
    radius    = 8.0
    points    = 0,-10:
    condition = :RETURN_HOME,true
}

//-----
Behavior = BHV_OpRegion
{
    polygon   = label,SafeOpBox: -280,-110:-130,-490:400,-280:250,10:80,40
    name      = bhv_op_region
}

```

C.2.3 December 4, Mission 1507 - Classification Vehicle 206

```

// This is a behavior configuration file.
//
// Legal comments are anything to the right of "/"
// or anything to the right of "#"

Initial_Var = RETURN_HOME,false

//-----

Behavior = BHV_Classify
{
    name = bhv_closerange
    pwt  = 100
    us   = nyak204
    meas_min = 10
    range_max = 350
    range_min = 35
    condition = TRACKING,TRACKING
    condition = :RETURN_HOME,false
}

```

```

}

//-----
Behavior = BHV_Orbit
{
    name = bhv_orbit
    pwt = 100
    us = nyak204
    orbcen = 0,-100
    orbrad = 50
    pieces = 8
    radius = 9
    speed = 2
    condition = TRACKING,NO_TRACK
    condition = :RETURN_HOME,false
}

//-----
Behavior = BHV_ConstantSpeed
{
    name = bhv_speed
    pwt = 1
    us = nyak204
    speed = 2.0
    condition = :RETURN_HOME,false
}

//-----
Behavior = BHV_Waypoint
{
    name = bhv_return
    speed = 1.4
    radius = 8.0
    points = 0,-10:
    condition = :RETURN_HOME,true
}

//-----
Behavior = BHV_OpRegion
{
    polygon = label,SafeOpBox: -280,-110:-130,-490:400,-280:250,10:80,40
    name = bhv_op_region
}

```


Bibliography

- [1] T. Curtin, J. Bellingham, J. Catipovic, and D. Webb, "Autonomous Oceanographic Sampling Networks," *Oceanography*, vol. 6, no. 3, pp. 86–94, 1993.
- [2] R. Brooks, "A Robust Layered Control System for a Mobile Robot," *IEEE Journal of Robotics and Automation*, vol. 2, 1986.
- [3] M. Benjamin, "Interval Programming: A Multi-Objective Optimization Model for Autonomous Vehicle Control," Ph.D. dissertation, Brown University, 2002.
- [4] M. Abidi and R. Gonzales, *Data Fusion in Robotics and Machine Intelligence*, 1992.
- [5] J. Farrell and W. Li and S. Pang and R. Arrieta, "Chemical Plume Tracing Experimental Results with a REMUS AUV," in *OCEANS 2003, MTS/IEEE Oceans Conference*, San Diego, CA, 2003.
- [6] R. Camilli and B. Bingham and M. Jakuba and H. Singh and J. Whelan, "Integrating In-Situ Chemical Sampling with AUV Control Systems," in *OCEANS 2004, MTS/IEEE Oceans Conference*, 2004.
- [7] M. Jakuba and D. Yoerger and A. Bradley and C. German and C. Langmuir and T. Shank, "Multiscale, Multimodal AUV Surveys for Hydrothermal Vent Localization," in *Fourteenth International Symposium on Unmanned Untethered Submersible Technology (UUST05)*, Durham, NH, 2005.
- [8] C. R. German, D. P. Connelly, R. D. Prien, D. R. Yoerger, M. Jakuba, A. Bradley, T. Shank, K. Nakamura, C. Langmuir, and L. M. Parsons, "New Techniques for Hydrothermal Plume Investigation by AUVs," in *Geophysical Research Abstracts, volume 7*, Vienna, Austria, 2005.
- [9] D. Yoerger, M. Jakuba, A. Bradley, and B. Bingham, "Techniques for Deep Sea Near Bottom Survey Using an Autonomous Underwater Vehicle," in *12th International Symposium of Robotics Research (ISRR 2005)*, San Francisco, CA, USA, Oct. 12th–15th 2005.
- [10] D. Popa, A.C. Sanderson, and R.J. Komerska, "Adaptive Sampling Algorithms for Multiple Autonomous Underwater Vehicles," in *Multiple Cooperating AUV (MCAUV) Digest*, 2005, pp. 73–83.
- [11] V. Hombal, A.C. Sanderson, D. Popa, S. Mupparapu, and R. Blidberg, "A Non-Parametric Iterative Algorithm for Adaptive Sampling," in *Multiple Cooperating AUV (MCAUV) Digest*, 2005, pp. 63–72.

- [12] E. Fiorelli, P. Bhatta, N.E. Leonard and I. Shulman, "Adaptive Sampling Using Feedback Control of an Autonomous Underwater Glider Fleet," in *Proceedings of the Thirteenth International Symposium on Unmanned, Untethered Submersible Technology (UUST03)*, 2003, pp. 1–16.
- [13] P. Ogren and E. Fiorelli and N.E. Leonard, "Cooperative Control of Mobile Sensor Networks: Adaptive Gradient Climbing in a Distributed Environment," *IEEE Transactions on Automatic Control*, vol. 49, no. 8, pp. 1292–1302, 2004.
- [14] E. Fiorelli, N. Leonard, P. Bhatta, and D. Paley, "Multi-AUV Control and Adaptive Sampling in Monterey Bay," in *AUV 2004: Workshop on Multiple AUV Operations*, 2004.
- [15] M. Triantafyllou and K. Streitlien, "Development of Autonomous Vehicles for Long Term Ocean Eddy Observation," MIT Sea Grant, Tech. Rep. MITSG91-16, 1991.
- [16] E. Burian, D. Yoerger, A. Bradley, and H. Singh, "Gradient Search with Autonomous Underwater Vehicles Using Scalar Measurements," in *Proc. AUV' 96*, 1996, pp. 86–98.
- [17] A.R. Robinson and S.M. Glenn, "Adaptive Sampling for Ocean Forecasting," *Naval Research Reviews*, vol. 51, no. 2, pp. 28–38, 1999.
- [18] J. Bellingham, "New Oceanographic Uses of Autonomous Underwater Vehicles," *Marine Technology Society Journal*, vol. 31, no. 3, pp. 34–47, 1997.
- [19] J. Bellingham and J. S. Willcox, "Optimizing AUV Oceanographic Surveys," in *AUV '96*, Monterey, CA, 1996, pp. 391–398.
- [20] J.S. Willcox and J.G. Bellingham and Y. Zhang and A.B. Baggeroer, "Performance Metrics for Oceanographic Surveys with Autonomous Underwater Vehicles," *IEEE Journal of Oceanographic Engineering*, vol. 26, no. 4, pp. 711–725, 2001.
- [21] D.R. Yoerger and A.M. Bradley and B.B. Walden, "The Autonomous Benthic Explorer (ABE): An AUV optimized for Deep Seafloor Studies," in *Proceedings of the Seventh International Symposium on Unmanned, Untethered Submersible Technology (UUST91)*, Durham, NH, 1991, pp. 60–70.
- [22] L. Freitag, M. Stojanovic, M. Grund, and S. Singh, "Acoustic Communications for Regional Undersea Observatories," Woods Hole Oceanographic Institution, Tech. Rep., 2002.
- [23] L. Freitag, M. Stojanovic, S. Singh, and M. Johnson, "Analysis of Channel Effect on Direct-Sequence and Frequency-Hopped Spread-Spectrum Acoustic Communication," *IEEE Journal of Oceanic Engineering*, vol. 26, no. 4, pp. 586–593, 2001.
- [24] H. J. Feder and J. Leonard, "Adaptive Mobile Robot Navigation and Mapping," *International Journal of Robotics Research*, vol. 18, no. 7, pp. 650–668, 1999.
- [25] S.T. Tuohy, N.M. Patrikalakis, J.J. Leonard, J.G. Bellingham, and C. Chrysosostomidis, "AUV Navigation Using Geophysical Maps with Uncertainty," in *Proc. 8th Int. Symposium on Unmanned, Untethered Submersible Technology (UUST93)*, 1993, pp. 265–276.
- [26] H. Singh, J. G. Bellingham, F. Hover, S. Lerner, B. A. Moran, K. von Der Heydt, and D. Yoerger, "Docking for an Autonomous Ocean-Sampling Network," *IEEE Journal of Oceanic Engineering*, vol. 26, no. 4, pp. 498–514, 2001.

- [27] H. Schmidt, J. Bellingham, M. Johnson, D. Herold, D. Farmer, and R. Pawlowicz, "Real-Time Frontal Mapping with AUVs in a Coastal Environment," in *Oceans 96 MTS/IEEE*, Washington, D.C., 1996, pp. 1094–1098.
- [28] D. Webb, P. J. Simonetti, and C. P. Jones, "SLOCUM: An Underwater Glider Propelled by Environmental Energy," *IEEE Journal of Oceanic Engineering*, vol. 26, no. 4, pp. 447–452, 2001.
- [29] B. Howe, B. Kirkham, A. Chave, A. Maffei, and S. Gaudet, "Wiring the Juan de Fuca Plate for Science: The NEPTUNE System," in *Oceanology International*, Miami, FL, 2001.
- [30] H. Schmidt, J. Edwards, and K. LePage, "Bistatic Synthetic Aperture Sonar Concept for MCM AUV Networks," in *International Workshop on Sensors and Sensing Technology for Autonomous Ocean Systems*, Kona, HI, 2000.
- [31] J. Zufferey and D. Floreano, "Fly-Inspired Visual Steering of an Ultralight Indoor Aircraft," *IEEE Transactions on Robotics and Automation*, vol. 22, no. 1, pp. 137–146, 2006.
- [32] J. Bellingham and T. Consi, "State Configured Layered Control," in *IARP 1st Workshop on Mobile Robotics for Subsea Environments*, Monterey, CA, October 1990.
- [33] R. C. Arkin, "Motor Schema Based Mobile Robot Navigation," *International Journal of Robotics Research*, vol. 8, no. 4, pp. 92–112, 1989.
- [34] N.E. Leonard and E. Fiorelli, "Virtual Leaders, artificial Potentials and Coordinated Control of Groups," in *Proc. 40th IEEE Conf. on Decision and Control*, 2001, pp. 2968–2973.
- [35] C.R. McInnes, "Potential Function Methods for Autonomous Spacecraft Guidance and Control," *Adv. Astronaut. Sci.*, vol. 90, pp. 2093–2109, 1996.
- [36] R. Olfati-Saber and R.M. Murray, "Distributed Cooperative Control of Multiple Vehicle Formations Using Structural Potential Functions," in *Proc. 15th IFAC World Cong.*, 2002, pp. 1–7.
- [37] E. Rimon and D.E. Koditschek, "Exact Robot Navigation Using Potential Functions," *IEEE Trans. Robot. Automat.*, vol. 8, pp. 501–518, 1992.
- [38] P. Pirjanian, "Multiple Objective Behavior-Based Control," *Robotics and Autonomous Systems, Special Issue*, 1999.
- [39] J. Riekkki, "Reactive Task Execution of a Mobile Robot," Ph.D. dissertation, Oulu University, 1999.
- [40] J. K. Rosenblatt, "DAMN: A Distributed Architecture for Mobile Navigation," Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, PA, 1997.
- [41] R. Brooks and C. Breazeal, "Embodied Intelligence," forthcoming from MIT Press.
- [42] L. Parker, "Heterogeneous Multi-Robot Cooperation," Ph.D. dissertation, Massachusetts Institute of Technology, 1994.
- [43] R. Brooks, P. Maes, M. Mataric, and G. Moore, "Lunar Base Construction Robots," in *IEEE International Workshop on Intelligent Robots and Systems*, Tsuchiura, Japan, 1990, pp. 389–392.

- [44] T. Nguyen, C. O'Donnell, and T. Nguyen, "Multiple Autonomous Robots for UXO Clearance, the Basic UXO Gathering System (BUGS) Project," in *2002 NRL Workshop on Multi-Robot Systems*, Washington, D.C., 2002, pp. 53–61.
- [45] K. Konolige and D. G. amnd K. Nicewarner, "A Multi-Agent System for Multi-Robot Mapping and Exploration," in *2002 NRL Workshop on Multi-Robot Systems*, Washington, D.C., 2002, pp. 11–19.
- [46] T. Balch and R. Arkin, "Behavior-Based Formation Control for Multi-Robot Teams," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 6, pp. 926–939, 1998.
- [47] A. Okubo, "Dynamical Aspects of Animal Grouping," *Adv. Biophys.*, vol. 22, pp. 1–94, 1986.
- [48] C. Reynolds, "Flocks, Herds, and Schools: A distributed Behavioral Model," in *Proc. ACM SIGGRAPH*, Anaheim, CA, 1987.
- [49] J. Lawton, B. Young, and R. Beard , "A Decentralized Approach to Elementary Formation Maneuvers," *IEEE Trans. Robot. Automat.*, vol. 19, pp. 933–941, 2003.
- [50] H. Tanner, A. Jadbabaie, and G.J Pappas, "Stable Flocking of Mobile Agents, Part I: Fixed Topology," in *Proc. 42nd IEEE Conf. Decision Control*, 2003, pp. 2010–2015.
- [51] M. R. Benjamin, "Interval Programming: A Multi-Objective Optimization Model for Autonomous Vehicle Control," Ph.D. dissertation, Brown University, Providence, RI, May 2002.
- [52] P. Pirjanian, "Multiple Objective Action Selection and Behavior Fusion," Ph.D. dissertation, Aalborg University, 1998.
- [53] R. A. Brooks, "A Robust Layered Control System for a Mobile Robot," *IEEE Journal of Robotics and Automation*, vol. RA-2, no. 1, pp. 14–23, April 1986.
- [54] M. R. Benjamin and J. Curcio, "COLREGS-Based Navigation in Unmanned Marine Vehicles," in *AUV-2004*, Sebasco Harbor, Maine, June 2004.
- [55] M. Benjamin, J. Curcio, J. Leonard, and P. Newman, "Navigation of Unmanned Marine Vehicles in Accordance with the Rules of the Road," in *International Conference on Robotics and Automation (ICRA)*, Orlando, Florida, May 2006.
- [56] M. Benjamin, M. Grund, and P. Newman, "Multi-objective Optimization of Sensor Quality with Efficient Marine Vehicle Task Execution," in *International Conference on Robotics and Automation (ICRA)*, Orlando, Florida, May 2006.
- [57] T. Henderson and E. Shilcrat, "Logical Sensor Systems," *Journal of Robotic Systems*, vol. 1, no. 2, pp. 169–193, 1984.
- [58] T.C. Henderson, C. Hansen, and B. Bhanu, "A Framework for Distributed Sensing and Control," in *Proc. 9th Int. Joint Conf. Artificial Intell.*, 1985, pp. 1106–1109.
- [59] —, "The Specification of Distributed Sensing and Control," *J. Robotic Syst.*, vol. 2, no. 4, pp. 387–396, 1985.
- [60] M. Montanari, J. Edwards, and H. Schmidt, "Autonomous Underwater Vehicle Based Concurrent Detection and Classification Concept Using Higher Order Spectral Analysis," forthcoming in the *IEEE Journal of Ocean Engineering*.

- [61] Y. Bar-Shalom and X. Li, *Estimation and Tracking: Principles, Techniques, and Software*, 1998.
- [62] J. Curcio, J. Leonard, and A. Patrikalakis, "SCOUT - A Low Cost Autonomous Surface Platform for Research in Cooperative Autonomy," in *OCEANS 2005*, Washington DC, September 2005.
- [63] J. N. Sanders-Reed, "Error Propagation in Two-sensor 3D Position Estimation," *Optical Engineering*, vol. 40, no. 4, 2001.
- [64] D. Eickstedt and M. Benjamin, "Adaptive Control of Heterogeneous Marine Sensor Platforms in an Autonomous Sensor Network," MIT Computer Science and AI Laboratory, Tech. Rep. 0000, 2006.
- [65] D. Eickstedt and H. Schmidt, "A Low-Frequency Sonar for Sensor-Adaptive, Multi-Static, Detection and Classification of Underwater Targets with AUVs," in *Oceans 2003*, San Diego, CA, 2003.
- [66] M. Jenkin and G. Dudek, "The Paparazzi Problem," in *IEEE/RSJ Conference on Intelligent Robotics and Systems*, 2000, pp. 2042–2047.
- [67] Y. Bar-Shalom, *Multitarget-Multisensor Tracking: Advanced Applications*. Artech House, 1990.