Robust State Fusers over Long-Haul Sensor Networks with Applications to Target Tracking

Submitted in partial fulfillment of the requirements for
the degree of
Doctor of Philosophy
in
Department of Electrical and Computer Engineering

Katharine G. Brigham

B.E., Electrical Engineering, Vanderbilt University
M.E., Systems Engineering, Cornell University

Carnegie Mellon University
Pittsburgh, PA

March 2015

Thesis Committee Members

Vijayakumar Bhagavatula (Advisor)
Department of Electrical and Computer Engineering,
Carnegie Mellon University

Xin Li
Department of Electrical and Computer Engineering,
Carnegie Mellon University

Nageswara Rao
Computer Science and Mathematics Division,
Oak Ridge National Laboratory

Xin Wang
Department of Electrical and Computer Engineering,
Stony Brook University

Abstract

In general, sensor networks consist of sensing, data processing, and communication components, and these sensors may communicate with each other or with a central processing center, which then performs some form of data aggregation or data fusion. The terms aggregation and fusion are often used for the same general purpose: how to simultaneously use pieces of information provided by several sources in order to come to a conclusion or a decision. A number of data fusion methods have been developed for sensor networks for a variety of applications, with a primary function of taking in the data from multiple sensors and combining this data to produce a condensed set of meaningful information with the highest possible degree of accuracy and certainty.

In this work we primarily explore the use of state fusers for target tracking applications that utilize long-haul communication networks where the underlying target dynamics are nonlinear (as is the case, for example, for a maneuvering target or a ballistic target). However, it is noted that the most popular approaches linearly combine the dataTherefore, the goal of the work is two-fold: 1) investigate/improve nonlinear fusion algorithms for target tracking and 2) develop methods to ensure that these nonlinear fusion algorithms are also robust against packet losses and delays that result from long-haul communications. In particular, we investigate the use of artificial neural networks (ANNs) for multisensor fusion. ANNs possess the capability of modeling arbitrary mappings, as long as a sufficient number of training samples are available from the same distribution. This also provides us with the ability to use nonlinear functions for fusing the data, which may yield better results than

with linear fusion given proper training.

More specifically, this thesis investigates several aspects of using ANN fusers for multi-sensor fusion in target tracking. Simulation experiments show that a significant amount of training data is required in close proximity to the test target in order to obtain good performance. Alternate methods in ANN training are then introduced which reduce the amount of training data required to obtain good performance, and widens the allowable training space as well. Then, the use of multiple fusers, different input features, and varied ANN architectures are investigated with the intent to further improve fuser performance. The effects of imperfect communications are then explored for the ANN fuser, and another training enhancement is suggested to generate ANN fusers that are more robust against packet losses and delays. Overall, this thesis intends to provide suggestions as to what parameters or aspects of the ANN may be explored to help improve fuser performance for use in target tracking.

Acknowledgments

I want to first and foremost thank Professor Vijayakumar Bhagavatula for being the best advisor that I could have ever hoped for. Thank you so much for all of your guidance, teachings, and patience with me throughout all of these years.

My thanks to the members of my committee for being so supportive and helpful with everything. I am very thankful to Dr. Nageswara Rao, Professor Xin Wang, and Professor Xin Li for all of their help with my project and for sharing their wealth of knowledge with me throughout all of my efforts.

I would also like to thank each and every one of my brilliant fellow graduate student colleagues that I had the pleasure of working with over these years. I am especially grateful to Jonathon Smereka for always "overdescending" himself (if I may use a "Smereka-ism"), who always went to great lengths to keep everyone on the ball. I must similarly thank Stephen "Patch" Siena, for always getting the food orders, and who sometimes managed to be quite the supportive fellow, even against his true nature. Thanks also to Ramu Bhagavatula, who was always there to bounce ideas off of, and has been very helpful throughout our friendship. And thanks to John Kelly, my classmate and good friend, for making my time at CMU that much more enjoyable.

I will be forever grateful to my family (the Gaus and the Brighams) for their neverending love and support. Their faith in my capabilities has never ceased to surprise me and was a huge factor in my ability to accomplish my goals. I am, of course and always, especially thankful to my husband John Brigham. Without him, I would not be anywhere near where

I am today.

I was also extremely lucky to have several organizations who contributed to my support throughout my education. Many thanks to Carnegie Mellon University and to the amazing ECE department, to Oak Ridge National Laboratory for supporting this project, and to Voci Technologies, who believed in me and supported me towards the end of my graduate career.

Contents

List of Figures

List of Tables

Notation and Abbreviations

$\mathbf{a}$	Vector
A	Matrix
a	Scalar
A^{ij}	Element of a Matrix at the i^{th} row and the j^{th} column
ANN	Artificial Neural Network
BLUE	Best Linear Unbiased Estimator
CC	Cross-Covariance
CI	Covariance Intersection
CMKF-D	Converted Measurements Kalman Filter with Debiasing
CWNA	Continuous White Noise Acceleration
EKF	Extended Kalman Filter
ER	Error Regularization
IMM	Interacting Multiple Model
KF	Kalman Filter
LMV	Linear Minimum-Variance
MMSE	Minimum Mean-Squared Error
NCT	Nearly Coordinated Turn
NN	Nearest Neighbor
NW	Nadaraya-Watson
SVR	Support Vector Regression

Chapter 1

Introduction

1.1 Motivation

In general, sensor networks contain sensor nodes that consist of sensing, data processing, and communication components, and these sensors may communicate with each other or with a central processing center, which often performs some form of data aggregation or data fusion. The terms aggregation and fusion are typically used for the same general purpose: how to simultaneously use multiple pieces of information provided by several sources in order to come to a conclusion or a decision [5]. A number of data fusion methods have been developed for sensor networks for a variety of applications, with a primary function of taking in the data from multiple sensors and combining this data to produce a condensed set of meaningful information with the highest possible degree of accuracy and certainty [6,7]. Sensor networks have been used in a wide variety of applications, including healthcare, military, security, and environmental monitoring, among others [8]. A particular class of sensor networks, which we call long-haul sensor networks, have communication connections that may span tens of thousands of miles. Such long-haul sensor networks have been deployed in a number of applications such as "the monitoring of greenhouse gas emissions using airborne and ground sensors [9], processing global cyber events using cyber sensors distributed over the

Internet [10], space exploration using a network of telescopes [11], and target detection and tracking for air and missile defense" [1].

In conventional sensor networks, the underlying communication network is regarded to be relatively "small" so that the effects of most network latencies and/or losses can be mitigated through an appropriate choice of communication protocols and data processing algorithms [12, 13]. In contrast, for long-haul networks, the effects of the network latencies and losses can become significant as the communication time may be on the same order as the time required and/or is available for algorithms to run for reporting. For example, in target tracking, consider the scenario where a report of several targets' positions must be made every 5 seconds at the central processing center. The central processing center receives data about target positions from various sensors that may be distributed across the globe or even in space, and the central processing center runs data aggregation or data fusion algorithms to agglomerate the received data, which, if performed appropriately, often results in more accurate results than with any single sensor. These data fusion algorithms may require several seconds to run to completion in order to produce a report (where suboptimal/potentially inaccurate results are obtained if the data fusion algorithms are stopped prior to completion). If the communication time between the sensors and the central processing center is also on the order of seconds (which may be the case for long-haul networks), then any additional latencies in receiving the data from the sensors may significantly cut into the fixed time that is allotted for the data aggregation algorithms to run before a report must be made, thus impacting the accuracy of the algorithms' result. A decision must therefore be made of whether to wait for the sensor data to be received or to continue and run the algorithms without it, which is a problem that is generally not considered in smaller networks. Therefore, mitigation techniques that are used for networks with short communication links may not generally work well for larger networks. The long-haul connections considered here present challenges that have not been adequately addressed by existing fusion methods, particularly methods to accommodate or account for missing or delayed data.

1.2 Problem Statement and Goal

One area that has seen considerable work in data fusion is in target tracking, whereby multiple sensors track a target by estimating its current state over time. Examples of such states that may be of interest are the target's kinematic states (e.g., its position, velocity), its physical state (e.g., radar cross-section), or its target classification (e.g., friend or foe). The main objective of target tracking is to estimate the state trajectories of moving targets, and a typical target tracking system may consist of multiple sensors, which communicate with a central processing center. We primarily explore the use of state fusers for target tracking applications that utilize long-haul communication networks, where the underlying target dynamics are nonlinear. We consider common communication problems such as power or limited bandwidth to be not major issues in this application. The main focus of this work is to investigate and develop improved fusion algorithms that have reduced error and are robust against packet losses and delays that result from long-haul communications.

There are a number of sensor fusion methods that have been developed that are well-suited for shorter and/or reliable communication links [7], and several such algorithms will be briefly described in Chapter 3. However, it is noted that the most popular approaches linearly combine the data. Therefore, the goal of the work is two-fold: 1) evaluate nonlinear fusion algorithms for target tracking that have reduced error and 2) develop methods to ensure that these nonlinear fusion algorithms are also robust against packet losses and delays that result from long-haul communications.

It is noted that in most applications, field tests may be performed using the sensor networks to collect measurements. We propose to use these measurements collected *a priori* to learn ways to fuse the data; in particular, we investigate the use of learning-based fusers for multisensor fusion. The methods explored here provide us with the ability to use nonlinear functions for fusing the data, which may potentially yield better results than with linear fusion. However, most learning-based fusers generally require that all of the inputs be present when computing the final fused output. Therefore, we further develop methods for

utilizing these nonlinear fusers in the case when there is missing input test data.

1.3 Contributions of the Thesis

In summary, this work primarily focuses on the investigation and development of improved fusion algorithms that are robust with specific application to target tracking, and can also deal with packet losses and delays that result from long-haul communications. The primary contributions of this thesis are as follows:

- **Machine Learning in Multisensor Fusion.** The investigation and evaluation of various nonlinear machine learning techniques for use in multisensor fusion for target tracking.
- **Robust Fusers.** The development of improved nonlinear fusion algorithms that have reduced error and are also robust against packet losses and delays that result from long-haul communications.
- **Training Considerations.** An analysis of the impact of various heuristics that are used in training learning-based fusers.
- **Simulation Results.** Demonstration of the performance of existing and new machine learning algorithms for fusing data collected from simulated targets.

1.4 Outline of the Document

The rest of this dissertation is organized as follows. Chapter 2 provides a description of the target tracking system considered herein, including a general architecture for the system and several common target motion and sensor measurement models and state estimate generation approaches, which will be used in subsequent chapters for simulating/demonstrating fuser performance. In Chapter 3, a brief summary of prior work in the area of multisensor fusion is

given. Chapter 4 discusses in more detail one of the best-performing nonlinear fusers found in Chapter 3 and describes modifications that can be made to make the fuser more robust for the purposes of target tracking. Chapter 4 also provides a more in-depth exploration of various heuristics that can affect our fuser performance. In Chapter 5, we address the long-haul aspect of our problem by investigating the application of previous methods for dealing with missing data and introduce new methods. Chapter 6 concludes this dissertation by revisiting our main contributions and includes some ideas for future work.

Chapter 2

Target Tracking

The main objective of target tracking is to estimate the state trajectories of a moving target. The general architecture of the target tracking system that we consider here is shown in Figure 2.1.

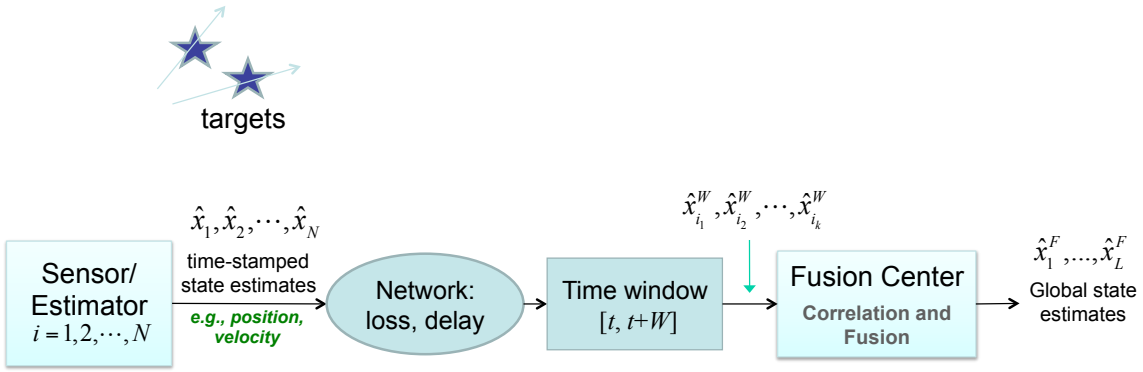


Figure 2.1: Target Tracking System Architecture [1].

There are objects, or "targets", that are within the field of view of our N sensors, and we wish to know the state of the target(s) (e.g., its position and velocity) while it is within the field of view. The sensors collect information about the target (e.g., its range and/or bearing), and each sensor can generate its own estimates of the true target state. These state estimates, which will be denoted as $\hat{\mathbf{x}}_1, ..., \hat{\mathbf{x}}_N$ (where the state estimate $\hat{\mathbf{x}}_i$ is generated by sensor i), are time-stamped and transmitted to a fusion center over a long-haul network

(e.g., over satellite links) over which random delays and packet losses may occur. Of these N state estimates, only $k \leq N$ state estimates, namely $\hat{\mathbf{x}}_{i_1}^W, \ldots \hat{\mathbf{x}}_{i_k}^W$, arrive within a time-window $[t, t+W]$ at the fusion center, and are used as inputs into the correlation and fusion algorithms. At the fusion center, the objective of the correlation algorithm is to group the state estimates such that each group corresponds to a single target. The fusion algorithm then combines the state estimates of each group into a single global estimate for that target (e.g., $\hat{\mathbf{x}}_1^F, \ldots \hat{\mathbf{x}}_L^F$ for L targets). In this work, as the focus is on multisensor fusion, the correlation will be assumed to be perfect and will therefore not be discussed further herein. The simulation results shown throughout are for a single target.

In the following sections, we will provide the necessary background information required for simulating of the target tracking system shown in Figure 2.1. We will step through each module in the diagram in Figure 2.1 and describe the mathematical models or algorithms used to simulate each module:

- (Section 2.1) Target dynamic model
- (Section 2.2) Sensor measurement model
- (Section 2.3) State estimate generation
- (Section 2.4) Network loss/delay
- (Chapter 3) Sensor Fusion

The correlation aspect of the 'Fusion Center' will not be discussed in this thesis since it is assumed that there is a single target in all simulations, and that there are no false alarms. In addition, since sensor fusion is the primary topic of this thesis, previous work in sensor fusion will be discussed in the next chapter.

2.1 Target Dynamic/Motion Models

Most tracking algorithms utilize a motion model because not only is knowledge of the target motion typically available, but tracking algorithms that use a model typically outperform model-free tracking algorithms [14]. Model-based tracking algorithms have become the most common type of tracking algorithm due to the improved performance that is gained by incorporating physical knowledge of the target motion in comparison to model-free algorithms, provided that the model is an accurate representation of the system [14]. A target dynamic (motion) model describes the evolution of a target's state over time. In this work, we generally consider targets that move according to a discrete-time dynamic system that takes on the following form:

$$\mathbf{x}(k+1) = f(\mathbf{x}(k)) + \mathbf{v}(k) \tag{2.1.1}$$

where $\mathbf{x}(k+1)$ is the n-dimensional state of the target at time $k+1$ and is a function of its previous state plus process noise $\mathbf{v}(k)$ that models "unpredictable disturbances". Typically, $\mathbf{v}(k)$ is modeled as zero-mean white Gaussian noise with covariance

$$E[\mathbf{v}(k)\mathbf{v}(k)^T] = Q(k). \tag{2.1.2}$$

In this work, we examine state fusion performance through simulation where targets move according to a linear or nonlinear dynamic system. We look at three different types of motion models: 1) the target is traveling at a (nearly) constant speed, 2) the target performs a maneuver; that is, a (nearly) coordinated turn (i.e., its turn rate and speed are nearly constant), and 3) the target is a ballistic object (e.g., a missile) traveling in the "coast" phase, where the dominant force acting upon the target is gravity. The motion models used herein to represent these cases take on the general form of Eq. (2.1.1). In the motion model for the first case, where the target is traveling at a nearly constant speed, the states evolve linearly (i.e., the function $f(\cdot)$ in Eq. (2.1.1) is a linear function), yielding a

relatively simple motion model. In the second and third cases, the states evolve nonlinearly, thus necessitating the use of nonlinear target tracking filters such as the Interacting Multiple Model (IMM) estimator and the Extended Kalman Filter (EKF). The following subsections provide further details on the motion models used.

2.1.1 Discretized Continuous White Noise Acceleration (CWNA) Model

The discretized Continuous White Noise Acceleration (CWNA) model [2] is a commonly used motion model in which an object moving in a generic coordinate ξ is assumed to be traveling at a near constant speed. The discrete-time state equation is as follows:

$$\mathbf{x}(k+1) = F\mathbf{x}(k) + \mathbf{v}(k) \tag{2.1.3}$$

where, (dropping the time index k), $\mathbf{x} = [\xi \quad \dot{\xi}]^T$ here is a two-dimensional vector representing the position and velocity, and F is known as the transition matrix.

Its acceleration is modeled as continuous time zero-mean white noise $\tilde{v}$:

$$\ddot{\xi}(t) = \tilde{v}(t) \tag{2.1.4}$$

where

$$E[\tilde{v}(t)] = 0 \tag{2.1.5}$$

$$E[\tilde{v}(t)\tilde{v}(\tau)] = \tilde{q}(t)\delta(t-\tau) \tag{2.1.6}$$

where $\tilde{q}$ is referred to as the process noise intensity.

Derivation of the State Transition Matrix and Process Noise Covariance [2]

The transition matrix F in Eq. (2.1.3) is derived from the continuous-time state equation, which is given in continuous time by:

$$\dot{\mathbf{x}}(t) = A(t)\mathbf{x}(t) + D(t)\tilde{v}(t) \tag{2.1.7}$$

where A and D are known matrices, $\mathbf{x}$ is the state vector, and $\tilde{v}$ is the process noise. The state equation in Eq. (2.1.7) has the following solution (see section 4.2.2 of [2]):

$$\mathbf{x}(t) = F(t, t_0)\mathbf{x}(t_0) + \int_{t_0}^{t} F(t, \tau)D(\tau)\tilde{v}(\tau)d\tau \tag{2.1.8}$$

where $\mathbf{x}(t_0)$ is the initial state and $F(t, t_0)$ is the state transition matrix from time t_0 to t.

State Transition Matrix:

The transition matrix has certain properties, such as:

$$F(t, t_0) = F(t_0, t)^{-1}, \tag{2.1.9}$$

and in general, it has no explicit form, unless the following commutativity property is satisfied:

$$A(t)\int_{t_0}^{t} A(\tau)d\tau = \int_{t_0}^{t} A(\tau)d\tau A(t). \tag{2.1.10}$$

Then, and only then, does the transition matrix have the form:

$$F(t, t_0) = e^{\int_{t_0}^{t} A(\tau)d\tau} \tag{2.1.11}$$

The condition in Eq. (2.1.10) is satisfied for time-invariant systems. For a time-invariant

system, assuming $t_0 = 0$, one has:

$$F(t) \triangleq F(t, 0) = e^{At}. \tag{2.1.12}$$

For the discretized CWNA model, our A and D matrices are

$$A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \tag{2.1.13}$$

$$D = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{2.1.14}$$

We can verify from plugging in the A and D from Eqs. (2.1.13) and (2.1.14) into Eq. (2.1.7) that we get:

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \xi(t) \\ \dot{\xi}(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \ddot{\xi}(t) \\ &= \begin{bmatrix} \dot{\xi}(t) \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \ddot{\xi}(t) \end{bmatrix} \\ &= \begin{bmatrix} \dot{\xi}(t) \\ \ddot{\xi}(t) \end{bmatrix} \end{aligned} \tag{2.1.15}$$

which is our desired result for the discretized CWNA model.

Therefore, for a sampling period of T, we can evaluate the state transition matrix $F(t, t+T)$ with the given A as follows:

$$F(t, t+T) = e^{\int_t^{t+T} A(\tau) d\tau} = e^{AT}. \tag{2.1.16}$$

We can make use of the following series expansion:

$$e^{AT} = \sum_{k=0}^{\infty} \frac{(AT)^k}{k!} = I + AT + \frac{A^2T^2}{2} + \dots \tag{2.1.17}$$

where I is an identity matrix of the same dimension as A. We can see that with the given A, we have

$$A^k = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \quad k \geq 2 \tag{2.1.18}$$

so:

$$e^{AT} = I + AT = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix} \tag{2.1.19}$$

Therefore, for a sampling period of T, the F in our discrete-time state equation in Eq. (2.1.3) is:

$$\boxed{F = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix}} \tag{2.1.20}$$

Process Noise Covariance:

To find the process noise covariance, we can relate the discrete-time process noise $\mathbf{v}(k)$ to the continuous-time process noise by:

$$\mathbf{v}(k) = \int_0^T e^{A(T-\tau)} D\tilde{v}(kT + \tau) d\tau. \tag{2.1.21}$$

We have $e^{A(T-\tau)} = \begin{bmatrix} 1 & T-\tau \\ 0 & 1 \end{bmatrix}$, so with the D from Eq. (2.1.14), Eq. (2.1.21) becomes:

$$\begin{aligned} \mathbf{v}(k) &= \int_0^T \begin{bmatrix} 1 & T-\tau \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tilde{v}(kT+\tau)d\tau \\ &= \int_0^T \begin{bmatrix} T-\tau \\ 1 \end{bmatrix} \tilde{v}(kT+\tau)d\tau \end{aligned} \tag{2.1.22}$$

Thus, the covariance of the process noise $\mathbf{v}(k)$ is (given that $E[\tilde{v}(t)] = 0$ from Eq. (2.1.5) and $E[\tilde{v}(t)\tilde{v}(\tau)] = \tilde{q}(t)\delta(t-\tau)$ from Eq. (2.1.6), and assuming $\tilde{q}$ to be constant):

$$\begin{aligned} E[\mathbf{v}(k)\mathbf{v}(k)^T] &= E\left[\left(\int_0^T \begin{bmatrix} T-\eta \\ 1 \end{bmatrix} \tilde{v}(kT+\eta)d\eta\right)\left(\int_0^T \begin{bmatrix} T-\tau \\ 1 \end{bmatrix} \tilde{v}(kT+\tau)d\tau\right)^T\right] \\ &= \int_0^T \int_0^T \begin{bmatrix} T-\eta \\ 1 \end{bmatrix} E[\tilde{v}(kT+\eta)\tilde{v}(kT+\tau)^T] \begin{bmatrix} T-\tau & 1 \end{bmatrix} d\eta d\tau \\ &= \tilde{q}\int_0^T \left(\int_0^T \begin{bmatrix} T-\eta \\ 1 \end{bmatrix} \delta(\eta-\tau)d\eta\right) \begin{bmatrix} T-\tau & 1 \end{bmatrix} d\tau \\ &= \tilde{q}\int_0^T \begin{bmatrix} T-\tau \\ 1 \end{bmatrix} \begin{bmatrix} T-\tau & 1 \end{bmatrix} d\tau \\ &= \tilde{q}\begin{bmatrix} T^3/3 & T^2/2 \\ T^2/2 & T \end{bmatrix} \end{aligned} \tag{2.1.23}$$

Thus our process noise covariance is:

$$\boxed{Q \triangleq E[\mathbf{v}(k)\mathbf{v}(k)^T] = \begin{bmatrix} T^3/3 & T^2/2 \\ T^2/2 & T \end{bmatrix} \tilde{q}} \tag{2.1.24}$$

We can also extend the discretized CWNA model to two coordinates, ξ, and η, so that the state of the target is $\mathbf{x} = [\xi \quad \dot{\xi} \quad \eta \quad \dot{\eta}]^T$:

$$\mathbf{x}(k+1) = \begin{bmatrix} 1 & T & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & T \\ 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{x}(k) + \mathbf{v}(k), \tag{2.1.25}$$

and the process noise covariance is given by

$$Q(k) \triangleq E[\mathbf{v}(k)\mathbf{v}(k)^T] = \begin{bmatrix} \tilde{q}_\xi \begin{bmatrix} T^3/3 & T^2/2 \\ T^2/2 & T \end{bmatrix} & \begin{matrix} 0 & 0 \\ 0 & 0 \end{matrix} \\ \begin{matrix} 0 & 0 \\ 0 & 0 \end{matrix} & \tilde{q}_\eta \begin{bmatrix} T^3/3 & T^2/2 \\ T^2/2 & T \end{bmatrix} \end{bmatrix}, \tag{2.1.26}$$

where $\tilde{q}_\xi$ and $\tilde{q}_\eta$ are the process noise intensities of the respective coordinates.

2.1.2 Target Maneuver

Targets may also move in a nonlinear fashion; for example, an aircraft may be traveling in a straight line at some constant speed, but then perform a maneuver such as a turn. A turn usually follows a pattern known as a *coordinated turn*, which is characterized by a constant turn rate and a constant speed [2]. The turn rate Ω is incorporated into the motion model

by augmenting the state vector for a horizontal motion model as follows:

$$\mathbf{x} = \begin{bmatrix} \xi & \dot{\xi} & \eta & \dot{\eta} & \Omega \end{bmatrix}^T, \tag{2.1.27}$$

which gives rise to the Nearly Coordinated Turn (NCT) model [2], given by:

$$\mathbf{x}(k+1) = \begin{bmatrix} 1 & \frac{\sin\Omega(k)T}{\Omega(k)} & 0 & -\frac{1-\cos\Omega(k)T}{\Omega(k)} & 0 \\ 0 & \cos\Omega(k)T & 0 & -\sin\Omega(k)T & 0 \\ 0 & \frac{1-\cos\Omega(k)T}{\Omega(k)} & 1 & \frac{\sin\Omega(k)T}{\Omega(k)} & 0 \\ 0 & \sin\Omega(k)T & 0 & \cos\Omega(k)T & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{x}(k) + \mathbf{v}(k), \tag{2.1.28}$$

and the covariance matrix of the process noise is

$$Q(k) \triangleq E[\mathbf{v}(k)\mathbf{v}(k)^T] = \begin{bmatrix} \tilde{q}_\xi \begin{bmatrix} T^3/3 & T^2/2 \\ T^2/2 & T \end{bmatrix} & \mathbf{0}_{2\times 2} & 0 \\ \mathbf{0}_{2\times 2} & \tilde{q}_\eta \begin{bmatrix} T^3/3 & T^2/2 \\ T^2/2 & T \end{bmatrix} & 0 \\ 0 \quad 0 & 0 \quad 0 & \tilde{q}_\Omega T \end{bmatrix}, \tag{2.1.29}$$

where $\tilde{q}_\Omega$ is the process noise intensity of the turn rate and $\mathbf{0}_{2\times 2}$ is a 2 by 2 matrix of zeros. It can typically be assumed that the horizontal and vertical motion models are decoupled [15], so the vertical component of the motion is not incorporated into this model.

2.1.3 Ballistic Coast Target

The final motion model we will consider for our simulations is the model for a ballistic coast target (i.e., a ballistic target in "coast" phase). Li and Jilkov provide a comprehensive survey of motion models for ballistic and space targets in [16], and some of the material from their

survey that is utilized in our simulations will be summarized in this subsection.

In general, the state-space model of a ballistic target has the form

$$\dot{\mathbf{x}} = \begin{bmatrix} \mathbf{v} \\ \mathbf{a} \end{bmatrix}, \tag{2.1.30}$$

where $\mathbf{x} = [\mathbf{p}^T \ \ \mathbf{v}^T]^T$ is the state vector consisting of the target's position $\mathbf{p} = [x \ \ y \ \ z]^T$ and velocity $\mathbf{v} = [\dot{x} \ \ \dot{y} \ \ \dot{z}]^T$, and x, y, and z characterize the coordinate system of interest. In this work, we will use the Earth-centered Inertial (ECI) coordinate system, as it is typically used in ballistic target tracking. The ECI coordinate system has its origin at the center of the Earth and does not rotate with the Earth (i.e., it is fixed relative to the "fixed stars") [16].

When a ballistic target is in the coast phase, gravity is considered to be the dominant force acting on the target, so the total acceleration is $\mathbf{a} = \mathbf{a}_G$, where $\mathbf{a}_G$ is the gravitational acceleration. There are three gravity models that are typically used for ballistic target tracking: 1) a Flat Earth model, which is a model that assumes a flat, non-rotating Earth, 2) a Spherical Earth model, which assumes that the Earth and the target can be represented as point masses at their centers, and the gravitational forces of the moon (and stars) can be neglected, and 3) an Ellipsoidal Earth model, which, as the name suggests, replaces the spherical Earth model with a (more accurate) ellipsoidal Earth model [16]. Li and Jilkov note that for a long-range coast ballistic target (our scenario of interest), accounting for the Earth sphericity and rotation may be essential, so we use the spherical Earth model for our gravity model in our simulations.

The following expression for $\mathbf{a}_G$, the gravitational acceleration, assumes a spherical Earth model [16]:

$$\mathbf{a}_G = -\frac{\mu}{\|\mathbf{p}\|^3}\mathbf{p} \tag{2.1.31}$$

where $\mathbf{p}$ is the target position vector from the Earth's center to the target, $\|\mathbf{p}\| = \sqrt{x^2 + y^2 + z^2}$ is its length, and $\mu = 3.986012 \times 10^5$ km^3/s^2 is the Earth's gravitational constant. The

continuous-time model of the system in Eq. (2.1.30) can therefore be rewritten using Eq. (2.1.31) as follows:

$$\frac{d}{dt}\begin{bmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ -\mu x/\|\mathbf{p}\|^3 \\ -\mu y/\|\mathbf{p}\|^3 \\ -\mu z/\|\mathbf{p}\|^3 \end{bmatrix} \tag{2.1.32}$$

An efficient algorithm for computing the state propagation can be found in [17].

2.2 Sensor Measurement Models

Each of the sensors collect measurements (e.g., the target range) according to the following measurement model:

$$\mathbf{z}(k) = h(\mathbf{x}(k)) + \mathbf{w}(k) \tag{2.2.1}$$

where $\mathbf{z}(k)$ is the measurement at time k, and is a function of the true target state, $\mathbf{x}(k)$, plus measurement noise, $\mathbf{w}(k)$. $\mathbf{w}(k)$ is typically modeled as zero-mean white Gaussian noise with covariance $E[\mathbf{w}(k)\mathbf{w}(k)^T] = R(k)$ and is independent of the process noise.

In most tracking applications, the target dynamics are best modeled in Cartesian coordinates, while the measurements are typically available in sensor coordinates (most often spherical coordinates) [18]. In a two-dimensional scenario (which we will employ for the discretized CWNA and NCT models since it is assumed that the x and y coordinates can be decoupled from the z coordinate), suppose we have an active sensor located at the Cartesian coordinates (x_a, y_a). The sensor will collect measurements of the target range (r) and azimuth angle (A) according to the following measurement model [19] (dropping the time

index k):

$$\mathbf{z} = \begin{bmatrix} r \\ A \end{bmatrix} = \begin{bmatrix} \sqrt{(x - x_a)^2 + (y - y_a)^2} \\ \tan^{-1}\left(\dfrac{y - y_a}{x - x_a}\right) \end{bmatrix} + \mathbf{w}, \tag{2.2.2}$$

where $\mathbf{w}$ is white Gaussian noise with covariance

$$R = \begin{bmatrix} \sigma_r^2 & 0 \\ 0 & \sigma_A^2 \end{bmatrix}, \tag{2.2.3}$$

and x and y are the true coordinates of the target.

We simulate the measurements for a ballistic coast target in 3D following the simulations in [20] by adding another parameter, the elevation. The measurements of the range (r), elevation (E), and azimuth (A) of the target are computed as follows:

$$\mathbf{z} = \begin{bmatrix} r \\ E \\ A \end{bmatrix} = \begin{bmatrix} \sqrt{x^2 + y^2 + z^2} \\ \tan^{-1}\left(z/\sqrt{x^2 + y^2}\right) \\ \tan^{-1}(x/y) \end{bmatrix} + \mathbf{w}, \tag{2.2.4}$$

where $\mathbf{w}$ is white Gaussian noise with covariance

$$R = \begin{bmatrix} \sigma_r^2 & 0 & 0 \\ 0 & \sigma_E^2 & 0 \\ 0 & 0 & \sigma_A^2 \end{bmatrix}. \tag{2.2.5}$$

2.2.1 State-Dependent Noise

We may have to assess state fuser performance under the presence of state-dependent noise since this type of noise is likely present in real-world situations although most fusers are not typically designed to account for state-dependent noise. For example, radar measurements are typically noisier when a target is off of the antenna boresight than if it were directly on the

axis of maximum antenna gain, so the measurement noise here depends on the target's state relative to the sensor location. Therefore, for our simulations, we also incorporate target state-dependent noise into the measurement model by using a simplified radar model to generate state values for σ_r, σ_E, and σ_A so that the errors are state-dependent and correlated across sensors. We will only consider the error that is dependent on the signal-to-noise ratio (SNR) for both the range and angle measurement accuracy since they usually dominate their overall radar error [21]. From [21], we have the following relationship between the standard deviation of the range and angle measurement errors and the SNR:

$$\sigma_r, \sigma_E, \sigma_A \propto \frac{1}{\sqrt{SNR}} \tag{2.2.6}$$

The SNR (from the well-known radar range equation [21]) is inversely proportional to r^4, where r is the range from the sensor to the target. To simplify, we assume a number of the radar parameters from the radar range equation are constant (e.g., the radar pulse duration, antenna gain, etc.) so that

$$\sigma_r, \sigma_E, \sigma_A \propto r^2 \tag{2.2.7}$$

The range and angle error of a ballistic target/satellite tracking phased array radar, the Cobra Dane, are published in Table 1 of [22] as 15ft and 0.05°, respectively, at a distance of 1000 nautical miles. These parameters are used to find reasonable values for scaling the standard deviations σ_r, σ_E, and σ_A used in the simulations to generate the state-dependent measurement noise.

2.3 Generating State Estimates

Each sensor then uses these measurements to estimate the true state, $\mathbf{x}(k)$. The method used for generating the state estimates depends on the assumed target motion model. For example, the Kalman Filter is typically employed to estimate the state of a target moving according to

a linear dynamic system (e.g., following the Discretized CWNA model from Section 2.1.1). When the target is moving in a nonlinear fashion, such as when it is performing a turn, a filter designed to account for the nonlinearities (e.g., the Extended Kalman Filter) should be used to estimate the target state. As stated previously, for our simulations, two types of targets will be considered for assessing fuser performance: 1) a civilian aircraft, which effectively operates in two different modes: uniform motion (at a constant speed and course), and a maneuver (e.g., a turn) [2], and 2) a ballistic target in the coast phase. As it is not the focus of this thesis, existing methods will be used to generate the state estimates for each target. For the aircraft, to generate the state estimates during uniform motion and during a maneuver, a Kalman filter and an Extended Kalman Filter are utilized with an Interacting Multiple Model (IMM) estimator, and for the ballistic target, a recursive Best Linear Unbiased Estimator (BLUE) filter developed by Zhao et. al. [3] is employed.

2.3.1 Kalman Filter (KF)

The Kalman Filter is the optimal Minimum Mean Square Error (MMSE) estimator when all noises entering the system are independent and normally distributed. If the noises are not Gaussian, and one only has the first two moments of the noise, then the Kalman Filter algorithm is the best *linear* MMSE state estimator [2]. Bar-Shalom et al. provide an excellent overview of the Kalman Filter (KF) in [2], which will be summarized here. Consider the scenario where we wish to know the true quantity x, but we are only given k measurements (which are each a function of x), made in the presence of disturbances (noise), $w(j)$:

$$z(j) = h\left(j, x, w(j)\right), \quad j = 1, ..., k. \tag{2.3.1}$$

We would like to find a function of the k observations, $\hat{x}(k, Z^k)$, which is called the estimator of x, and is a function of the given measurements $Z^k = \{z(j)\}_{j=1}^{k}$. Dropping the time index

k (assuming it is fixed), the minimum mean-square error (MMSE) estimator is given by

$$\begin{aligned} \hat{x}^{MMSE}(Z) &= \arg\min_{\hat{x}} E[(\hat{x}-x)^2|Z] \\ &= E[x|Z]. \end{aligned} \tag{2.3.2}$$

For two random vectors $\mathbf{x}$ and $\mathbf{z}$ that are jointly Gaussian, the MMSE estimator $\hat{\mathbf{x}}$, i.e., the conditional mean, is given by

$$\hat{\mathbf{x}} \triangleq E[\mathbf{x}|\mathbf{z}] = \bar{\mathbf{x}} + P_{xz}P_{zz}^{-1}(\mathbf{z}-\bar{\mathbf{z}}), \tag{2.3.3}$$

and the corresponding conditional covariance matrix is

$$P_{xx|z} \triangleq E[(\mathbf{x}-\hat{\mathbf{x}})(\mathbf{x}-\hat{\mathbf{x}})^T|\mathbf{z}] = P_{xx} - P_{xz}P_{zz}^{-1}P_{zx}, \tag{2.3.4}$$

where the overbar denotes the unconditional expected values (e.g., $\bar{\mathbf{x}} \triangleq E[\mathbf{x}]$), and

$$P_{xx} \triangleq \mathrm{cov}(\mathbf{x}) = E[(\mathbf{x}-\bar{\mathbf{x}})(\mathbf{x}-\bar{\mathbf{x}})^T]; \tag{2.3.5}$$

$$P_{xz} \triangleq \mathrm{cov}(\mathbf{x},\mathbf{z}) = E[(\mathbf{x}-\bar{\mathbf{x}})(\mathbf{z}-\bar{\mathbf{z}})^T] = P_{zx}^T; \tag{2.3.6}$$

$$P_{zz} \triangleq \mathrm{cov}(\mathbf{z}) = E[(\mathbf{z}-\bar{\mathbf{z}})(\mathbf{z}-\bar{\mathbf{z}})^T]. \tag{2.3.7}$$

Now consider a linear version of the general discrete-time dynamic system described earlier in Eq. (2.1.1):

$$\mathbf{x}(k+1) = F(k)\mathbf{x}(k) + \mathbf{v}(k), \tag{2.3.8}$$

and a linear version of the measurement equation given in Eq. (2.2.1):

$$\mathbf{z}(k) = H(k)\mathbf{x}(k) + \mathbf{w}(k) \tag{2.3.9}$$

where $F(k)$ and $H(k)$ are matrices, and $\mathbf{v}(k)$ and $\mathbf{w}(k)$ are zero-mean white Gaussian noise vectors and are assumed to be mutually independent.

Our goal is to obtain an estimate of $\mathbf{x}(k+1)$, which is the true state at time $k+1$, and Eqs. (2.3.3) and (2.3.4) can be used to obtain a recursion that yields the state estimate at time $k+1$, denoted as $\hat{\mathbf{x}}(k+1)$, by relating our target dynamic model (the dynamic case, Eq. (2.3.8)) with the MMSE estimator equation (the static case, Eq. (2.3.3)) as follows.

We can relate the static case to the dynamic case by letting the unconditional (prior) expectations in Eq. (2.3.3) represent the expectations *prior* to the availability of the measurement at time $k+1$. Similarly, the conditional (posterior) expectations in Eq. (2.3.3) will represent the expectations *posterior* to obtaining the measurement at time $k+1$. Therefore, from Eq. (2.3.3) we have the following equivalence for the prior mean:

$$\bar{\mathbf{x}} \rightarrow \bar{\mathbf{x}}(k+1) \triangleq \hat{\mathbf{x}}(k+1|k) \triangleq E[\mathbf{x}(k+1)|Z^k] \tag{2.3.10}$$

which is known as the **predicted state**. Similarly, for the measurement, let $\mathbf{z}$ in Eq. (2.3.3) represent the measurement taken at time $k+1$, $\mathbf{z}(k+1)$, with the prior mean

$$\bar{\mathbf{z}} \rightarrow \bar{\mathbf{z}}(k+1) \triangleq \hat{\mathbf{z}}(k+1|k) \triangleq E[\mathbf{z}(k+1)|Z^k] \tag{2.3.11}$$

which is also known as the **predicted measurement**. Let $\hat{\mathbf{x}}$ in Eq. (2.3.3) represent the estimate of $\mathbf{x}(k+1)$ in Eq. (2.3.8) (our desired quantity). We can compute the estimate *posterior* to time $k+1$ using the following representation:

$$\hat{\mathbf{x}} \rightarrow \hat{\mathbf{x}}(k+1) \triangleq \hat{\mathbf{x}}(k+1|k+1) \triangleq E[\mathbf{x}(k+1)|Z^{k+1}] \tag{2.3.12}$$

Now, using the above equivalences, we obtain the following dynamic estimation algorithm.

Predicted State:

We can substitute the state equation in Eq. (2.3.8) into Eq. (2.3.10) to obtain the predicted state:

$$\begin{aligned}\hat{\mathbf{x}}(k+1|k) &\triangleq E[\mathbf{x}(k+1)|Z^k] \\ &= E[F(k)\mathbf{x}(k) + \mathbf{v}(k)|Z^k] \\ &= \boxed{F(k)\hat{\mathbf{x}}(k|k)}\end{aligned} \tag{2.3.13}$$

Recall that the process noise $\mathbf{v}(k)$ is white and zero mean, so its expected value is zero.

Predicted Measurement:

Likewise, to obtain an expression for the predicted measurement, we can substitute the measurement equation in Eq. (2.3.9) into Eq. (2.3.11) as follows:

$$\begin{aligned}\hat{\mathbf{z}}(k+1|k) &\triangleq E[\mathbf{z}(k+1)|Z^k] \\ &= E[H(k+1)\mathbf{x}(k+1) + \mathbf{w}(k+1)|Z^k] \\ &= \boxed{H(k+1)\hat{\mathbf{x}}(k+1|k)}\end{aligned} \tag{2.3.14}$$

where again, since the measurement noise $\mathbf{w}(k+1)$ is white and zero mean, its expected value is zero.

Covariance Matrices:

Now that we have expressions for the predicted state and measurement, $\bar{\mathbf{x}}$ and $\bar{\mathbf{z}}$, respectively, from Eq. (2.3.3), we still need the covariance matrices P_{xz} and P_{zz} (whose expressions are given in Eqs. (2.3.6) and (2.3.7), respectively) in order to compute $\hat{\mathbf{x}}$. The **state prediction error** can be obtained by subtracting the predicted state in Eq. (2.3.13) from the true state

at time $k+1$:

$$\begin{aligned}\tilde{\mathbf{x}}(k+1|k) &\triangleq \mathbf{x}(k+1) - \hat{\mathbf{x}}(k+1|k) \\ &= F(k)\tilde{\mathbf{x}}(k|k) + \mathbf{v}(k),\end{aligned} \tag{2.3.15}$$

and similarly, the **measurement prediction error** can be obtained by subtracting the predicted measurement in Eq. (2.3.14) from the true measurement received at time $k+1$:

$$\begin{aligned}\tilde{\mathbf{z}}(k+1|k) &\triangleq \mathbf{z}(k+1) - \hat{\mathbf{z}}(k+1|k) \\ &= H(k+1)\tilde{\mathbf{x}}(k+1|k) + \mathbf{w}(k+1).\end{aligned} \tag{2.3.16}$$

The covariance between the state and the measurement, P_{xz} is, using Eq. (2.3.16):

$$\begin{aligned}P_{xz} &\triangleq E[\tilde{\mathbf{x}}(k+1|k)\tilde{\mathbf{z}}(k+1|k)^T|Z^k] \\ &= E\left[\tilde{\mathbf{x}}(k+1|k)\left[H(k+1)\tilde{\mathbf{x}}(k+1|k) + \mathbf{w}(k+1)\right]^T|Z^k\right] \\ &= E\left[\tilde{\mathbf{x}}(k+1|k)\tilde{\mathbf{x}}(k+1|k)^T|Z^k\right]H(k+1)^T \\ &\triangleq P(k+1|k)H(k+1)^T\end{aligned} \tag{2.3.17}$$

where $P(k+1|k)$ is the **state prediction covariance**, which can be computed as follows:

$$\begin{aligned}P(k+1|k) &\triangleq E[\tilde{\mathbf{x}}(k+1|k)\tilde{\mathbf{z}}(k+1|k)^T|Z^k] \\ &= F(k)E[\tilde{\mathbf{x}}(k|k)\tilde{\mathbf{x}}(k|k)^T|Z^k]F(k)^T + E[\mathbf{v}(k)\mathbf{v}(k)^T] \\ &\triangleq F(k)P(k|k)F(k)^T + Q(k)\end{aligned} \tag{2.3.18}$$

where recall that $Q(k)$ is the process noise covariance.

The **measurement prediction covariance** P_{zz} can be computed similarly to P_{xz} and is given as:

$$\begin{aligned} P_{zz} &\triangleq H(k+1)P(k+1|k)H(k+1)^T + R(k+1) \\ &\triangleq S(k+1) \end{aligned} \tag{2.3.19}$$

where recall that $R(k)$ is the measurement noise covariance.

Now that we have expressions for the quantities given in Eq. (2.3.3), we can rewrite the expression for the MMSE estimator $\hat{\mathbf{x}}$ using Eqs. (2.3.10), (2.3.11), (2.3.17), and (2.3.19) as follows:

$$\boxed{\hat{\mathbf{x}}(k+1|k+1) = \hat{\mathbf{x}}(k+1|k) + P(k+1|k)H(k+1)^T S(k+1)^{-1}\left[\mathbf{z}(k+1) - \hat{\mathbf{z}}(k+1|k)\right]} \tag{2.3.20}$$

which is also known as the **updated state estimate** and is our final estimate for the true state at time $k+1$. The Kalman Filter utilizes these equations to recursively estimate the current state, and these equations are summarized in Table 2.1.

Table 2.1: Kalman Filter (KF) equations.

Prediction:	
State prediction	$\hat{\mathbf{x}}(k+1\|k) = F(k)\hat{\mathbf{x}}(k\|k)$
State prediction covariance	$P(k+1\|k) = F(k)P(k\|k)F(k)^T + Q(k)$
Update:	
Measurement residual	$\tilde{\mathbf{z}}(k+1\|k) = \mathbf{z}(k+1) - H(k+1)\hat{\mathbf{x}}(k+1\|k)$
Measurement residual covariance	$S(k+1) = H(k+1)P(k+1\|k)H(k+1)^T + R(k+1)$
Kalman Filter gain	$W(k+1) = P(k+1\|k)H(k+1)^T S(k+1)^{-1}$
Updated state estimate	$\hat{\mathbf{x}}(k+1\|k+1) = \hat{\mathbf{x}}(k+1\|k) + W(k+1)\tilde{\mathbf{z}}(k+1\|k)$
Updated state covariance	$P(k+1\|k+1) = P(k+1\|k) - W(k+1)S(k+1)W(k+1)^T$

2.3.2 Extended Kalman Filter (EKF)

While the Kalman Filter is designed more for linear system dynamics, there are scenarios (e.g., during a maneuver), where the underlying target dynamics are nonlinear. The Extended Kalman Filter was developed based on the Kalman Filter to account for these nonlinearities by using Taylor series expansion so that it is effectively a nonlinear version of the Kalman Filter. The filter equations are slightly modified from the KF as shown in Table 2.2; the items shown in blue font are the terms that differ from the original Kalman Filter. In the EKF, the state prediction is no longer a linear function of the previous state, but a nonlinear (assumed to be known) function.

Table 2.2: Extended Kalman Filter (EKF) equations.

Prediction:	
State prediction	$\hat{\mathbf{x}}(k+1\|k) = f(\hat{\mathbf{x}}(k\|k))$
State prediction covariance	$P(k+1\|k) = F(k)P(k\|k)F(k)^T + Q(k)$
Update:	
Measurement residual	$\tilde{\mathbf{z}}(k+1\|k) = \mathbf{z}(k+1) - H(k+1)\hat{\mathbf{x}}(k+1\|k)$
Measurement residual covariance	$S(k+1) = H(k+1)P(k+1\|k)H(k+1)^T + R(k+1)$
Kalman Filter gain	$W(k+1) = P(k+1\|k)H(k+1)^T S(k+1)^{-1}$
Updated state estimate	$\hat{\mathbf{x}}(k+1\|k+1) = \hat{\mathbf{x}}(k+1\|k) + W(k+1)\tilde{\mathbf{z}}(k+1\|k)$
Updated state covariance	$P(k+1\|k+1) = P(k+1\|k) - W(k+1)S(k+1)W(k+1)^T$
State transition matrix	$F(k) = \left.\frac{\partial f}{\partial \mathbf{x}}\right\|_{\hat{\mathbf{x}}(k\|k)}$
Measurement matrix	$H(k+1) = \left.\frac{\partial h}{\partial \mathbf{x}}\right\|_{\hat{\mathbf{x}}(k+1\|k)}$

2.3.3 Interacting Multiple Model (IMM) Estimator

The IMM estimator considers multiple target motion models by running multiple filters that use the different motion models. The overall state estimate from the IMM estimator is obtained by a weighted sum of the filter outputs, where the weights are model probabilities

computed by the IMM estimator. This estimator is used in our simulations for the maneuvering target and runs the two filters (described in the previous subsections) in parallel: a Kalman Filter (KF) using the discretized CWNA model, and an Extended Kalman Filter (EKF) using the NCT model. The IMM estimator makes use of the total probability theorem to combine the state estimates generated by each independent filter. The algorithm for the IMM estimator is detailed in [2] and will be summarized here as well.

Figure 2.2 provides an overview of the IMM algorithm for a single cycle using two filters. The output of each filter from the previous time step $k-1$ is mixed using the mixing probabilities to produce the "mixed initial conditions". These mixed initial conditions are then input into each filter, where they are are updated using the current measurement to produce updated filter outputs, and the likelihood of each mode (target model) is also computed. These mode likelihoods can then be used to update the mixing probabilities at the current time k, which can then be used to combine the updated filter outputs to produce a final state estimate and covariance at the current time k. While Figure 2.2 is depicted for two filters, this algorithm is can easily accommodate more than two filters by running the additional filters in parallel to the two filters depicted here.

The mathematical equations for the IMM algorithm are as follows. To fuse the individual results from r different filters that are employed in parallel, the IMM algorithm computes mixing probabilities for each possible mode (target model). The mixing probability for each mode at time k is defined as follows:

$$\begin{aligned}\mu_{i|j}(k-1|k-1) &\triangleq P\left[M_i(k-1)|M_j(k), Z^{k-1}\right] \\ &= \frac{1}{\bar{c}_j} P\left[M_j(k)|M_i(k-1), Z^{k-1}\right] P\left[M_i(k-1)|Z^{k-1}\right] \\ &= \frac{1}{\bar{c}_j} p_{ij}\mu_i(k-1), \qquad i,j = 1,...,r,\end{aligned} \tag{2.3.21}$$

where $M_j(k)$ is defined as the event that model j is in effect at time k, and the normalizing

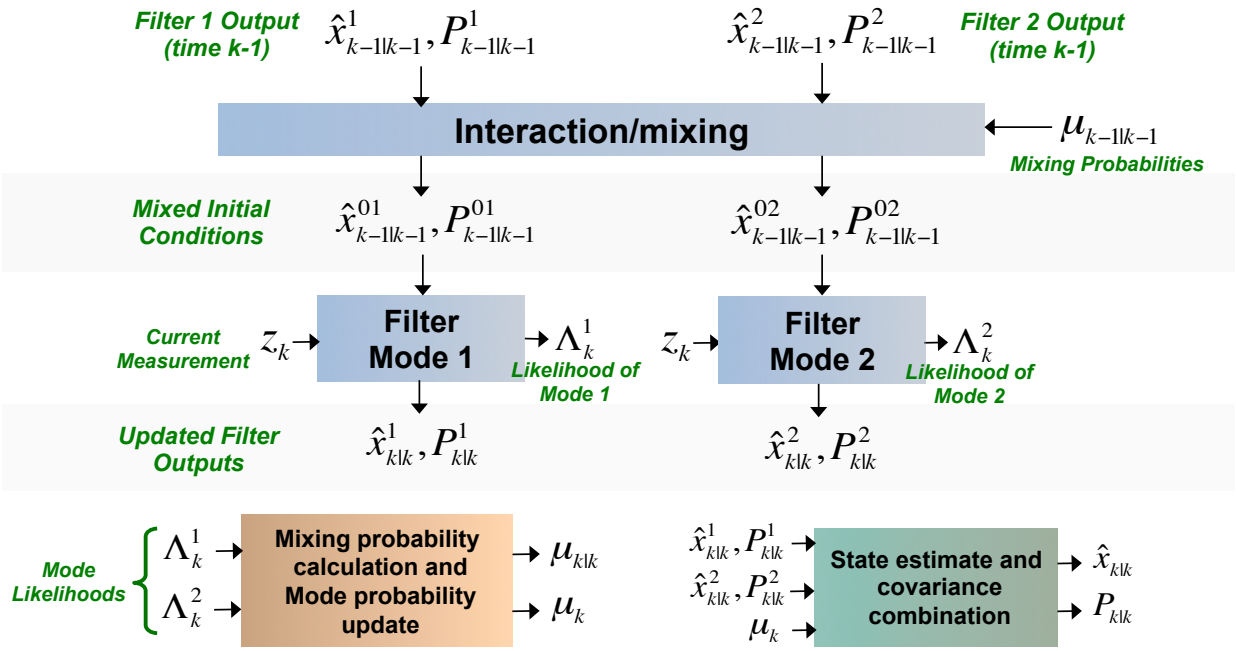


Figure 2.2: IMM Estimator for two filters (one cycle) [2].

constant $\bar{c}_j$ is given by:

$$\bar{c}_j = \sum_{i=1}^{r} p_{ij}\mu_i(k-1), \qquad j = 1, ..., r. \tag{2.3.22}$$

The "mixing" of the filters can then be performed by first computing the mixed initial state for the filter matched to $M_j(k)$, and its initial covariance:

$$\hat{x}^{0j}(k-1|k-1) = \sum_{i=1}^{r} \hat{x}_i(k-1|k-1)\mu_{i|j}(k-1|k-1), \qquad j = 1, ..., r \tag{2.3.23}$$

$$\begin{aligned} P^{0j}(k-1|k-1) = \sum_{i=1}^{r} \mu_{i|j}(k-1|k-1)\Big\{ & P^i(k-1|k-1) \\ & + \left[\hat{x}^i(k-1|k-1) - \hat{x}^{0j}(k-1|k-1)\right] \\ & \cdot \left[\hat{x}^i(k-1|k-1) - \hat{x}^{0j}(k-1|k-1)\right]^T \Big\}, \quad j = 1, ..., r \end{aligned} \tag{2.3.24}$$

The initial state and covariance in Eqs. (2.3.23) and (2.3.24) are then input into the filter matched to $M_j(k)$, which then updates these initial estimates using the current measurement

$z(k)$.

To combine the filter outputs, the mode probabilities are updated as follows:

$$\mu_j(k) = \frac{1}{c} \cdot \Lambda_j(k)\bar{c}_j, \tag{2.3.25}$$

where

$$c = \sum_{j=1}^{r} \Lambda_j(k)\bar{c}_j \tag{2.3.26}$$

and

$$\Lambda_j(k) = \mathcal{N}\left[z(k); \hat{z}^j\left[k|k-1; \hat{x}^{0j}(k-1|k-1)\right], S^j\left[k; P^{0j}(k-1|k-1)\right]\right], \quad j = 1, ..., r. \tag{2.3.27}$$

Once the mode probabilities are updated, the filter outputs can be combined as follows to yield the final state estimate and covariance at time $k+1$:

$$\hat{x}(k|k) = \sum_{j=1}^{r} \hat{x}^j(k|k)\mu_j(k); \tag{2.3.28}$$

$$P(k|k) = \sum_{j=1}^{r} \mu_j(k+1)\Big\{P^j(k|k) + \left[\hat{x}^j(k|k) - \hat{x}(k|k)\right]\left[\hat{x}^j(k|k) - \hat{x}(k|k)\right]^T\Big\}. \tag{2.3.29}$$

2.3.4 Coordinate Conversion

As stated earlier, since we are simulating a radar system for collecting our sensor measurements, the target's position is typically reported in polar or spherical coordinates. Two common approaches for dealing with the different coordinate systems are provided in [23]:

1. Convert the measurements to a Cartesian frame of reference and utilize a KF.

2. Employ an EKF.

The accuracy of the measurement conversion is dependent on the accuracy of the original measurements. For certain levels of measurement errors, the bias of the errors after conver-

sion is significant, thus requiring some form of debiasing compensation [23]. We can compute the bias in the converted measurement errors through the following conventional analysis.

Let us assume that the range measurement r_m and the azimuth angle (bearing) measurement A_m are defined with respect to the true range r and true bearing A as follows:

$$r_m = r + \tilde{r}, \qquad A_m = A + \tilde{A}, \tag{2.3.30}$$

where $\tilde{r}$ and $\tilde{A}$ are noise that are assumed to be independent and zero-mean with standard deviations σ_r and σ_A, respectively. The polar coordinates can be converted into Cartesian coordinates by the standard conversion:

$$\begin{aligned} x_m &= r_m \cos A_m = (r + \tilde{r}) \cos(A + \tilde{A}); \\ y_m &= r_m \sin A_m = (r + \tilde{r}) \sin(A + \tilde{A}). \end{aligned} \tag{2.3.31}$$

We are interested in the converted measurement error statistics so that we may debias the converted measurements and utilize the appropriate measurement covariance matrix when employing the KF. We can compute the bias and covariance of the errors exactly with the following assumptions: 1) the errors in the polar measurements are zero-mean and normally distributed, and 2) we have knowledge of the true state (r, A). If we condition the true mean and covariance on the true state (which we know from the second assumption), we can compute the expressions for the mean and covariance exactly. However, since in practice the true state is unknown, we can instead condition the mean and covariance on the *measured state*. The resulting equations for the mean μ and covariance R are as follows [23]:

$$\mu \triangleq \begin{bmatrix} E[\tilde{x}|r_m, A_m] \\ E[\tilde{y}|r_m, A_m] \end{bmatrix} = \begin{bmatrix} r_m \cos A_m (e^{-\sigma_A^2} - e^{-\sigma_A^2/2}) \\ r_m \sin A_m (e^{-\sigma_A^2} - e^{-\sigma_A^2/2}) \end{bmatrix} \tag{2.3.32}$$

$$
\begin{aligned}
R^{11} &\triangleq \mathrm{var}(\tilde{x}|r_m, A_m) \\
&= r_m^2 e^{-2\sigma_A^2}[\cos^2 A_m(\cosh 2\sigma_A^2 - \cosh \sigma_A^2) + \sin^2 A_m(\sinh 2\sigma_A^2 - \sinh \sigma_A^2)] \\
&\quad + \sigma_r^2 e^{-2\sigma_A^2}[\cos^2 A_m(2\cosh 2\sigma_A^2 - \cosh \sigma_A^2) + \sin^2 A_m(2\sinh 2\sigma_A^2 - \sinh \sigma_A^2)]
\end{aligned}
$$

$$
\begin{aligned}
R^{22} &\triangleq \mathrm{var}(\tilde{y}|r_m, A_m) \\
&= r_m^2 e^{-2\sigma_A^2}[\sin^2 A_m(\cosh 2\sigma_A^2 - \cosh \sigma_A^2) + \cos^2 A_m(\sinh 2\sigma_A^2 - \sinh \sigma_A^2)] \\
&\quad + \sigma_r^2 e^{-2\sigma_A^2}[\sin^2 A_m(2\cosh 2\sigma_A^2 - \cosh \sigma_A^2) + \cos^2 A_m(2\sinh 2\sigma_A^2 - \sinh \sigma_A^2)]
\end{aligned} \tag{2.3.33}
$$

$$
\begin{aligned}
R^{12} &\triangleq \mathrm{cov}(\tilde{x}, \tilde{y}|r_m, A_m) \\
&= \sin A_m \cos A_m e^{-4\sigma_A^2}[\sigma_r^2 + (r_m^2 + \sigma_r^2)(1 - e^{\sigma_A^2})] \\
&= R^{21}
\end{aligned}
$$

These equations for the mean μ and covariance R are derived from expanding Eq. (2.3.31), and using the following set of identities (assuming zero-mean Gaussian errors in the polar measurements):

$$
\begin{aligned}
E[\cos \tilde{A}] &= e^{-\sigma_A^2/2} \\
E[\sin \tilde{A}] &= 0 \\
E[\cos^2 \tilde{A}] &= \frac{1}{2}(1 + e^{-2\sigma_A^2}) \\
E[\sin^2 \tilde{A}] &= \frac{1}{2}(1 - e^{-2\sigma_A^2}) \\
E[\sin \tilde{A} \cos \tilde{A}] &= 0
\end{aligned} \tag{2.3.34}
$$

The debiased conversion is then given by:

$$
\begin{bmatrix} x^{dc} \\ y^{dc} \end{bmatrix} = \begin{bmatrix} r_m \cos A_m \\ r_m \sin A_m \end{bmatrix} - \mu, \tag{2.3.35}
$$

where μ is given in Eq. (2.3.32). Simulation results from [23] demonstrate that using converted measurements after debiasing with the KF outperforms the EKF. Therefore, the converted measurements with debiasing approach will be used in our simulations when utilizing the KF is desired. This approach will be denoted as CMKF-D (Converted Measurements Kalman Filter with Debiasing).

2.3.5 Recursive Best Linear Unbiased Estimator (BLUE) Filter

However, the converted measurements approach with debiasing clearly has its flaws. The true errors are dependent on the true state, while the CMKF-D computes the error statistics based on the measurements, so the resulting filter is by no means optimal. Zhao et al. note this in [3] and derive an optimal recursive filter in the linear MMSE sense for systems with linear dynamics and nonlinear measurements. The algorithm is shown in Figure 2.3. In Figure 2.3, both the prediction and update equations for this filter are provided, and the framework is similar to that of the Kalman Filter with a few modifications. The main deviations from the Kalman Filter are how the measurement error covariance matrix S, the gain filter K, and the predicted measurement, are computed.

For comparison purposes, this filter is adopted for use with tracking the ballistic target. The CMKF-D filter is used to help track the maneuvering target. Since the focus of this thesis is not on state estimate generation but rather on multisensor fusion, both tracking approaches (the CMKF-D and recursive BLUE filters) will be used in our simulations to demonstrate the overall efficacy of the fusion techniques irregardless of which state estimation approach is selected.

1. Prediction:

$$\bar{\mathbf{x}}_k = [\bar{x}, \dot{\bar{x}}, \bar{y}, \dot{\bar{y}}, \bar{z}, \dot{\bar{z}}]' = F_{k-1}\hat{\mathbf{x}}_{k-1} + \Gamma_{k-1}\bar{w}_{k-1}$$

$$\bar{P} = F_{k-1}P_{k-1}F'_{k-1} + \Gamma_{k-1}Q_{k-1}\Gamma'_{k-1}$$

$$\bar{r} = \sqrt{\bar{x}^2 + \bar{y}^2 + \bar{z}^2}, \qquad \bar{r}_1 = \sqrt{\bar{x}^2 + \bar{y}^2}$$

$$\alpha = \left(\frac{\mu_2\sigma_r^2}{\bar{r}^2} + \frac{\mu_3\bar{z}^2}{\bar{r}_1^2} + \frac{\mu_3\sigma_r^2\bar{z}^2}{\bar{r}_1^2\bar{r}^2} \right)$$

$$\alpha_1 = (\lambda_2\mu_2 - \lambda_1^2\mu_1^2)\bar{x}^2 + \lambda_3\mu_2\bar{y}^2$$

$$\alpha_2 = (\lambda_2\mu_2 - \lambda_1^2\mu_1^2)\bar{y}^2 + \lambda_3\mu_2\bar{x}^2$$

$$\alpha_3 = (\mu_2 - \mu_1^2)\bar{z}^2 + \mu_3(\bar{x}^2 + \bar{y}^2)$$

$$\alpha_4 = (\mu_2(\lambda_2 - \lambda_3) - \lambda_1^2\mu_1^2)\bar{x}\bar{y}$$

$$\alpha_5 = (\lambda_1(\mu_2 - \mu_3) - \lambda_1\mu_1^2)\bar{z}$$

$$S(1,1) \approx \lambda_2\mu_2\bar{P}(1,1) + \lambda_3\mu_2\bar{P}(3,3) + \alpha(\lambda_2\bar{x}^2 + \lambda_3\bar{y}^2) + \alpha_1$$

$$S(2,2) \approx \lambda_2\mu_2\bar{P}(3,3) + \lambda_3\mu_2\bar{P}(1,1) + \alpha(\lambda_3\bar{x}^2 + \lambda_2\bar{y}^2) + \alpha_2$$

$$S(3,3) \approx \mu_2\bar{P}(5,5) + \mu_3\left(\bar{P}(1,1) + \bar{P}(3,3) + \mu_2\sigma_r^2\frac{\bar{z}^2}{\bar{r}^2} + \mu_3\sigma_r^2\frac{\bar{r}_1^2}{\bar{r}^2} \right) + \alpha_3$$

$$S(1,2) = S(2,1) \approx (\lambda_2 - \lambda_3)(\mu_2\bar{P}(1,3) + \alpha\bar{x}\bar{y} + \alpha_4$$

$$S(1,3) = S(3,1) \approx \lambda_1(\mu_2 - \mu_3)\left(\bar{P}(1,5) + \sigma_r^2\frac{\bar{x}\bar{z}}{\bar{r}^2} \right) + \alpha_5\bar{x}$$

$$S(2,3) = S(3,2) \approx \lambda_1(\mu_2 - \mu_3)\left(\bar{P}(3,5) + \sigma_r^2\frac{\bar{y}\bar{z}}{\bar{r}^2} \right) + \alpha_5\bar{y}$$

$$K_k = \mu_1[\lambda_1\bar{P}(:,1), \lambda_1\bar{P}(:,3), \bar{P}(:,5)]S^{-1}$$

2. Update:

$$\hat{\mathbf{x}}_k = \bar{\mathbf{x}}_k + K_k(\mathbf{z}_k - \mu_1[\lambda_1\bar{x}, \lambda_1\bar{y}, \bar{z}]')$$

$$P_k = \bar{P}_k - K_kSK'_k$$

Constants:

$$\lambda_1 = e^{-\sigma_\theta^2/2}, \qquad \lambda_2 = \tfrac{1}{2}(1 + e^{-2\sigma_\theta^2}), \qquad \lambda_3 = \tfrac{1}{2}(1 - e^{-2\sigma_\theta^2})$$

$$\mu_1 = e^{-\sigma_\phi^2/2}, \qquad \mu_2 = \tfrac{1}{2}(1 + e^{-2\sigma_\phi^2}), \qquad \mu_3 = \tfrac{1}{2}(1 - e^{-2\sigma_\phi^2})$$

Figure 2.3: Algorithm for the recursive BLUE filter presented in [3].

2.4 Network Loss/Delays

After the sensors generate the state estimates, in our target tracking scenario the state estimates are sent over long-haul communication links, where network delays and packet losses may occur. Due to reporting time requirements (e.g., a report of the target state may be due once per second), packets that are delayed are effectively considered to be lost since a report must be made, and the fusion center cannot wait for the delayed packet to arrive. Therefore, packet losses will also be simulated, and its effects on multisensor fusion will be evaluated in Chapter 5.

For the simulations involving packet losses, we adopt the TCP model used by Rao et al. in [1]. The TCP model presented in [1] considers a simple loss model where packets are independently lost with some probability p. After each packet is sent, the sender waits for a time-out period T_{TO} to receive an acknowledgment from the receiver. The packet will then be re-sent if no acknowledgment is received within the set time-out period. Typically, T_{TO} is on the order of several times the round-trip time of the connection, and over long-haul networks, this could be on the order of seconds, which is the same as that of the time-window allowed for the correlation and fusion algorithms to run [1]. Rao et al. derived an expression for the probability that a message will be successfully delivered within some time window W as follows.

Let T_{EE}^{T} denote the time at which a message is received at the receiver using TCP, and let T_L represent the latency of the connection. The probability that a message will be delivered after exactly i losses is $p^i(1-p)$, which corresponds to the time $T_{EE}^{T} = iT_{TO} + T_L$. Then, the expected time at which the message is received using TCP is:

$$E\left[T_{EE}^{T}\right] = T_L + \sum_{i=0}^{\infty} ip^i(1-p)T_{TO} = T_L + \frac{p}{1-p}T_{TO}, \tag{2.4.1}$$

and the second moment is given by

$$E\left[\left(T_{EE}^{T}\right)^{2}\right] = T_L^2 + 2T_{TO}T_L\frac{p}{1-p} + T_{TO}^2\frac{p(1+p)}{(1-p)^3}. \tag{2.4.2}$$

These expressions can be used to derive the probability that a message will be successfully delivered within the time-window $[T_L, T_L + W]$ by applying Chebyshev's inequality of the second order, $\mathbf{P}\{X > \delta\} \leq \frac{E[X^2]}{\delta^2}$, as follows:

$$\begin{aligned}\mathbf{P}\left\{T_{EE}^{T} - T_L < W\right\} &= 1 - \mathbf{P}\left\{T_{EE}^{T} - T_L > W\right\} \\ &\geq 1 - \frac{E\left[(T_{EE}^{T} - T_L)^2\right]}{W^2} \\ &= 1 - \frac{T_{TO}^2 p(1+p)}{W^2(1-p)^3},\end{aligned} \tag{2.4.3}$$

For the simulations in Chapter 5, a time-out period of $T_{TO} = 0.3$ secs will be used, with the window W being set to 1 second to match the scan rate of the sensors, which will also be 1 second.

2.5 Summary of the Target Simulation Setup

To summarize, two different targets will be simulated:

1. Maneuvering Target: *Travels at a constant zero or nonzero turn rate.*

2. Ballistic Coast Target: *The only force acting on this target is gravity.*

The maneuvering target uses two dynamic models: the discretized CWNA model for when the turn rate is zero so that the target is traveling in a linear fashion, and the NCT model for when the turn rate is a nonzero constant. Since the horizontal and vertical motions for the maneuvering are assumed to independent (and are therefore decoupled), only a 2D radar measurement measuring the distance to (radius) and the bearing of the target (azimuth) relative to the sensor is required. Furthermore, due to the two different models (i.e., modes)

being employed for the maneuvering target, the use of an IMM filter is desirable, utilizing both a KF and an EKF (with converted debiased measurements).

The ballistic target uses a single dynamic model where it assumes that the only force acting on the target is gravity. The spherical Earth model is employed to model the gravitational acceleration. Since the horizontal and vertical motions are not independent in our ballistic target motion model (see Eq. (2.1.32)), a radar measurement that includes an elevation measurement is needed in order to estimate the true target state of the ballistic target. A recursive BLUE filter is used to estimate the ballistic target state.

To simulate network loss, the state estimates from individual sensors are dropped with some probability p. If a packet (i.e., state estimate) is dropped, the missing state estimate will then be estimated at the fusion center using an assumed target model to predict the missing estimate from a previously received state estimate. The models and algorithms used for each target in our simulations are summarized in Table 2.3.

Table 2.3: Summary of the models and algorithms used in the simulation of tracking the two targets.

	Maneuvering Target	**Ballistic Coast Target**
Dynamic Model	1) Discretized CWNA Model 2) Nearly Coordinated Turn (NCT) Model	Gravitational Acceleration only with Spherical Earth Model
Sensor Measurement Model	2D Radar Measurement: ⁕ Radius ⁕ Azimuth with state-dependent noise	3D Radar Measurement: ⁕ Radius ⁕ Azimuth ⁕ Elevation with state-dependent noise
State Estimate Generation	IMM Filter with: 1) KF with debiased converted measurements 2) EKF with debiased converted measurements	Recursive BLUE Filter for spherical (3D) measurements
Network Loss	Probability of loss following a uniform distribution $\sim U[0,1]$	

It is noted, however, that alternate tracking approaches could be used (different sensor measurement models and/or different state estimation approaches). For example, we could also elect to utilize an EKF with debiased converted measurements with the ballistic target, but recall that the focus of this thesis is primarily on multisensor fusion. Different tracking approaches will be used in our simulations to show the overall effectiveness of the fusion techniques despite these differing approaches. In other words, we prefer to have more diversity in our simulations so that fuser performance is not solely demonstrated for a single type of simulation.

2.6 Simulations (Target Trajectories and Estimates)

Here we show plots of an example target trajectory and corresponding state estimates for each simulated target. Note that these trajectories/state estimates are used for our test target when comparing fuser performance in Chapter 3. The sampling/scan rate of the sensors is one sample per second.

2.6.1 Sensor Locations

Since our measurement noise is dependent upon the location of the target with respect to the sensor, two arbitrary sensor locations were selected around the world: Alaska, and Greenland, with the exact latitudes and longitudes given in Table 2.4. These latitudes and longitudes were then converted to the ECI coordinates for use in all simulations.

Table 2.4: Sensor locations and corresponding ECI coordinates.

	Latitude	**Longitude**	**ECI coordinates** (x, y, z)
Sensor 1 (Alaska)	52.7373°N	174.09143°E	(-3757.92, 889.624, 5076.18)
Sensor 2 (Greenland)	76.5311113°N	68.7030563°W	(356.553, -1442.17, 6202.79)

2.6.2 Example 1: Maneuvering Target

The initial state of the maneuvering target, $\mathbf{x}_{MT}(0)$, in Cartesian coordinates (with the positions x_0 and y_0 in km, the velocities $\dot{x}_0$ and $\dot{y}_0$ in km/s, and the turn rate Ω_0 in radians) is set to [19]:

$$\begin{aligned}\mathbf{x}_{MT}(0) &= [x_0 \quad \dot{x}_0 \quad y_0 \quad \dot{y}_0 \quad \Omega_0]^T \\ &= [0 \quad 0.1 \quad 5 \quad 0.1 \quad 0]^T\end{aligned} \tag{2.6.1}$$

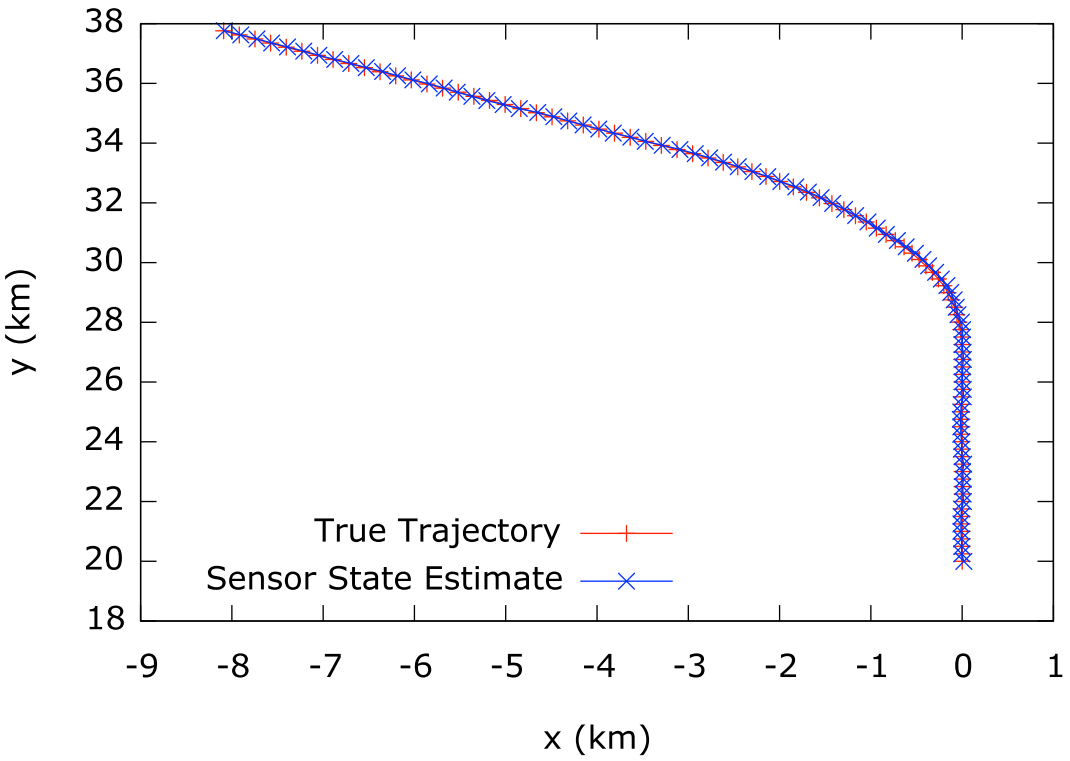
The example target travels straight (following the Discretized CWNA model) until time $t = 50$s. At $t = 50$s, the target begins to take a left turn at a turn rate of $2°$/s for 50s (following the NCT model), and then continues straight again until $t = 150$s. Figure 2.4 shows a plot of the true trajectory and the state estimates, as well as the Root-Mean-Squared (RMS) error of the state estimates for one of the sensors.

2.6.3 Example 2: Ballistic Coast Target

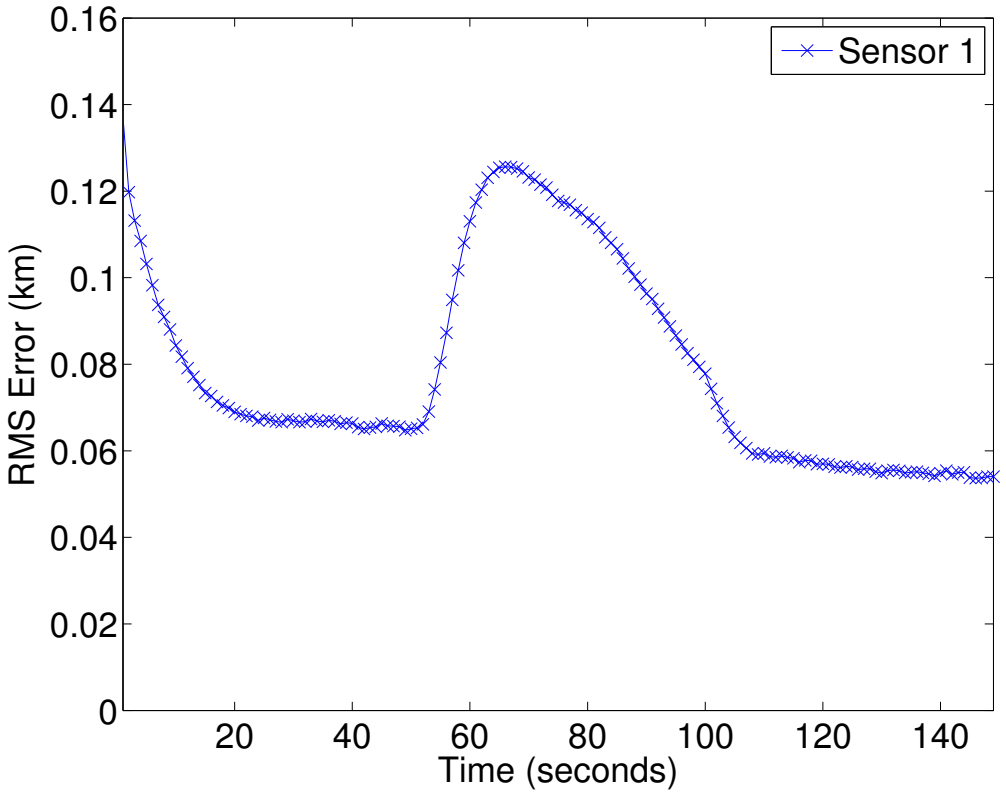
The initial state of the target, $\mathbf{x}_{BT}(0)$, is set to the following (in km for position, and km/s for velocity):

$$\begin{aligned}\mathbf{x}_{BT}(0) &= [x_0 \quad \dot{x}_0 \quad y_0 \quad \dot{y}_0 \quad z_0 \quad \dot{z}_0]^T \\ &= [113.75 \quad 0.94 \quad 3950 \quad 3.33 \quad 6150 \quad -6.0125]^T\end{aligned} \tag{2.6.2}$$

For the ballistic target, checks were performed before all simulations to verify that the simulated target was indeed within the expected range for a ballistic target; it should be above the reentry point (100 km above the Earth), but less than the maximum altitude for a ballistic target at approximately 1200 km [4]. Figure 2.5 shows a plot of the trajectory as a function of its x-y-z coordinates, and Figure 2.6 shows the actual altitude of the ballistic target above the Earth.



(a) Maneuver Trajectory with overlaid corresponding sensor state estimates.



(b) State estimate RMS error

Figure 2.4: Maneuvering Target

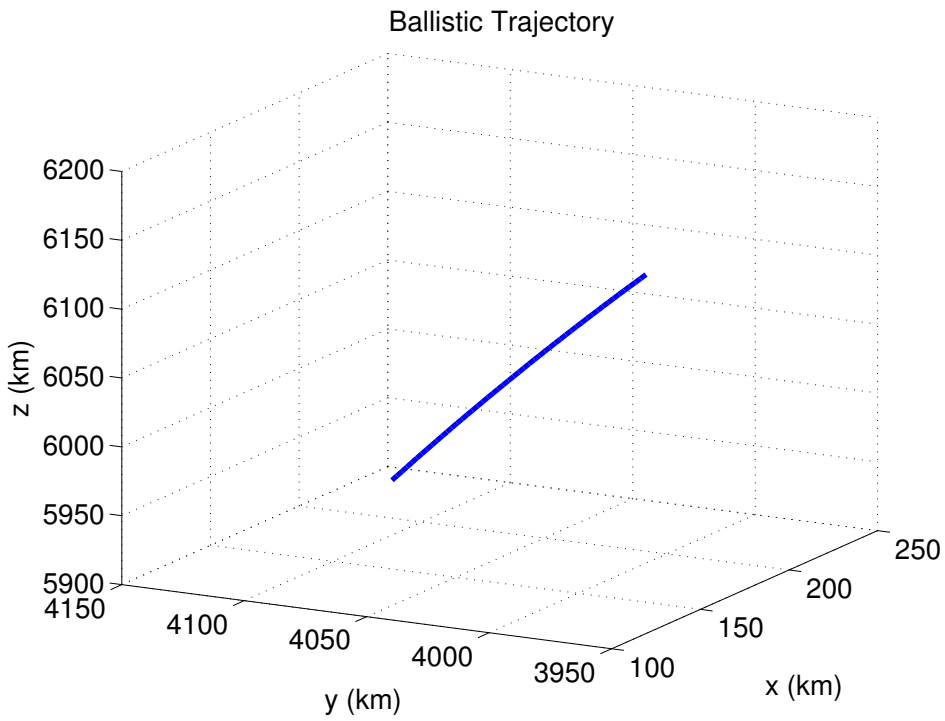


Figure 2.5: Ballistic Trajectory in ECI coordinates.

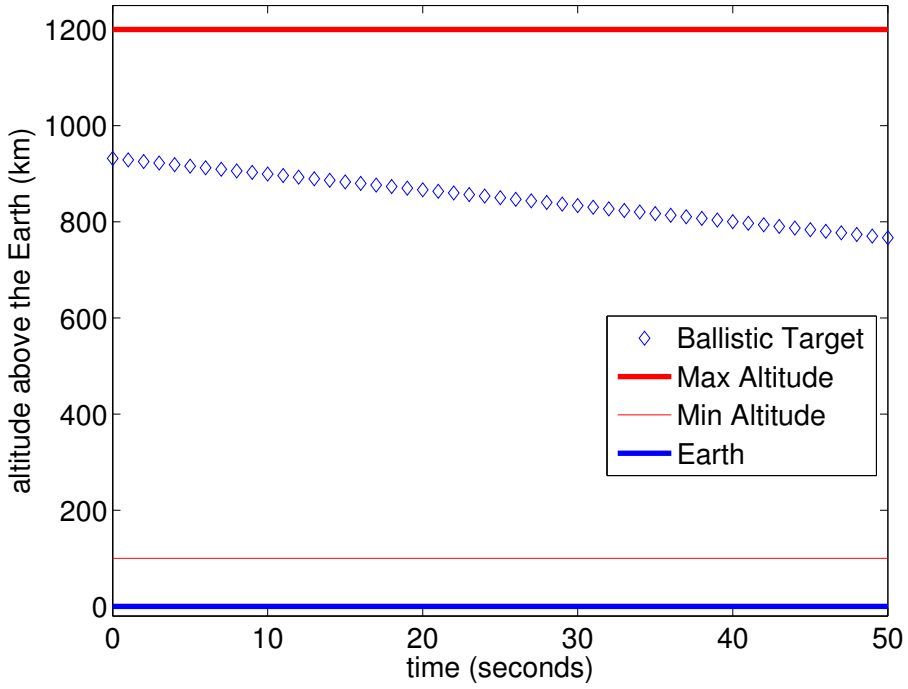
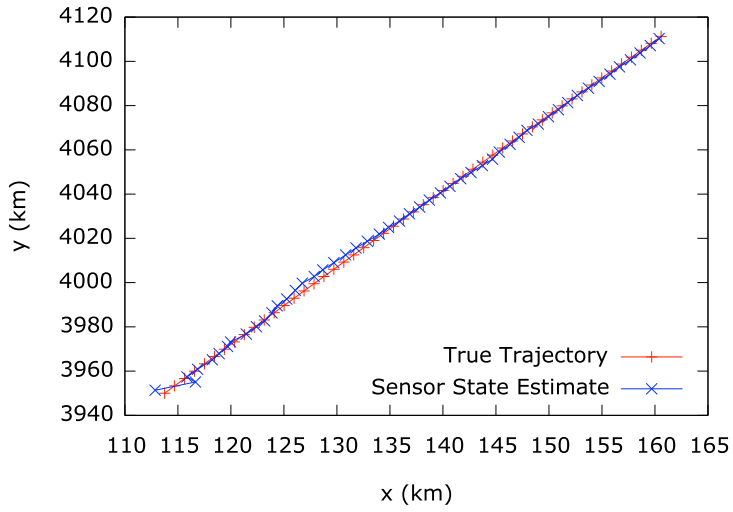
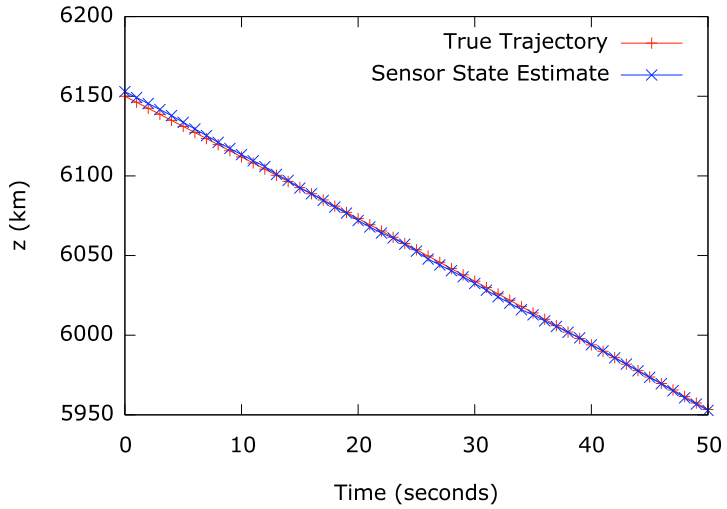


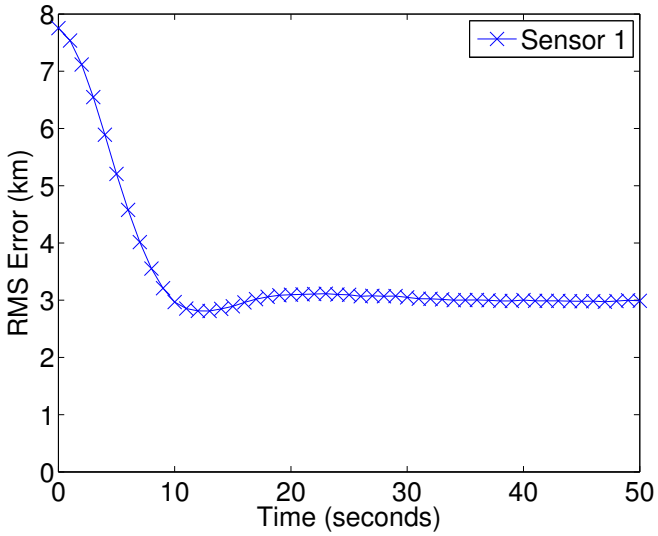
Figure 2.6: Ballistic Target altitude in km above the Earth (at 0 km). Also shown is the minimum and maximum altitude for a ballistic target (100 km and 1200 km, respectively [4]).



(a) Ballistic x-y Trajectory with overlaid corresponding sensor state estimates.



(b) Ballistic z Trajectory with overlaid corresponding sensor state estimates.



(c) State estimate RMS error

Figure 2.7: Ballistic Target Trajectory with overlaid state estimates and corresponding RMS error in position.

Chapter 3

Prior Work in Multisensor Fusion

After each sensor sends its state estimates to the fusion center, the state estimates from all sensors are correlated to specific targets and combined to produce more accurate global target state estimates. The correlation process is also sometimes referred to as data association, which is actually not a straightforward process and can introduce errors that will also affect the outputs after the fusion. However, in this work, a single target is assumed; therefore, data association is not investigated as the focus here is primarily on fusers. In this chapter we provide descriptions of two popular linear fusion methods for comparison with several nonlinear fusers.

3.1 Linear Fusers

Two linear fusers will be described in this section, both of which are popular in target tracking. The most commonly utilized linear fuser is the Linear Minimum Variance (LMV) fuser, which requires the (typically unknown) error cross-covariance between sensors for optimality. The second method presented here, called the Covariance Intersection(CI) fuser, attempts to bypass the need for knowing the cross-covariance between sensors through an alternate formulation of the fusion problem.

3.1.1 Linear Minimum Variance (LMV) Fuser

An optimal state fuser (in the linear minimum variance sense) is derived in [24]. The algorithm is as follows. Suppose we received an N_o-dimensional state estimate from each of the N sensors, each of which we will denote as $\hat{\mathbf{x}}_i$, where i is the sensor index ($i = 1, ..., N$). Each state estimate also has an associated error covariance estimate $\hat{P}_{ii}$, which is an estimate of the true error covariance $P_{ii} = E[(\mathbf{x} - \hat{\mathbf{x}}_i)(\mathbf{x} - \hat{\mathbf{x}}_i)^T]$, where $\mathbf{x}$ is the true target state.

The fused state estimate $\hat{\mathbf{x}}_F$ is therefore defined as

$$\hat{\mathbf{x}}_F = A_1\hat{\mathbf{x}}_1 + A_2\hat{\mathbf{x}}_2 + \cdots + A_N\hat{\mathbf{x}}_N, \tag{3.1.1}$$

and the optimal matrix weights A_i for $i = 1, 2, ..., N$ are computed by

$$A = (B^T\Sigma^{-1}B)^{-1}B^T\Sigma^{-1}, \tag{3.1.2}$$

where $\Sigma = [P_{ij}]_{i,j=1,2,...,N}$ is an $NN_o \times NN_o$ symmetric positive definite matrix (where $P_{ij} = E[(\mathbf{x} - \hat{\mathbf{x}}_i)(\mathbf{x} - \hat{\mathbf{x}}_j)^T]$), and $A = [A_1, A_2, ..., A_N]^T$, $B = [I_{N_o}, ..., I_{N_o}]^T$ are both $NN_o \times N_o$ matrices with I_{N_o} being an $N_o \times N_o$ identity matrix. The corresponding error covariance of this fused estimate, P_F, is given by

$$P_F = (B^T\Sigma^{-1}B)^{-1}. \tag{3.1.3}$$

However, P_{ij} for $i \neq j$ is typically unknown, so P_{ij} is typically set to $\mathbf{0}_{N_o \times N_o}$, but the result will be suboptimal.

3.1.2 Covariance Intersection (CI) Algorithm

Another sensor fusion method is the covariance intersection (CI) algorithm. The intuition behind this approach comes from a geometric interpretation of the problem. If one were to plot the covariance ellipses for P_F (defined as the locus of points $\{\mathbf{y} : \mathbf{y}^T P_F^{-1}\mathbf{y} = c\}$ where

c is some constant), the ellipses of P_F are found to always lie within the intersection of the ellipses for P_1 and P_2 for all possible choices of P_{12} [25]. The intersection is characterized by the convex combination of sensor covariances (for two sensors, as an example):

$$P_F = (\omega_1 P_1^{-1} + \omega_2 P_2^{-1})^{-1} \tag{3.1.4}$$

and the corresponding sensor fusion for the CI algorithm is

$$\hat{x}_F = P_F \left(\omega_1 P_1^{-1}\hat{x}_1 + \omega_2 P_2^{-1}\hat{x}_2\right), \quad \omega_1 + \omega_2 = 1 \tag{3.1.5}$$

where $\omega_1, \omega_2 > 0$ are weights to be determined (e.g., by minimizing the determinant of P_F, which is related to minimizing the volume of space implied by the covariance matrix).

Recently, Wang and Li [26] proposed a fast CI algorithm where the weights are found based on an information-theoretic criterion so that ω_1 and ω_2 can be determined analytically as follows:

$$\omega_1 = \frac{D(p_1, p_2)}{D(p_1, p_2) + D(p_2, p_1)} \tag{3.1.6}$$

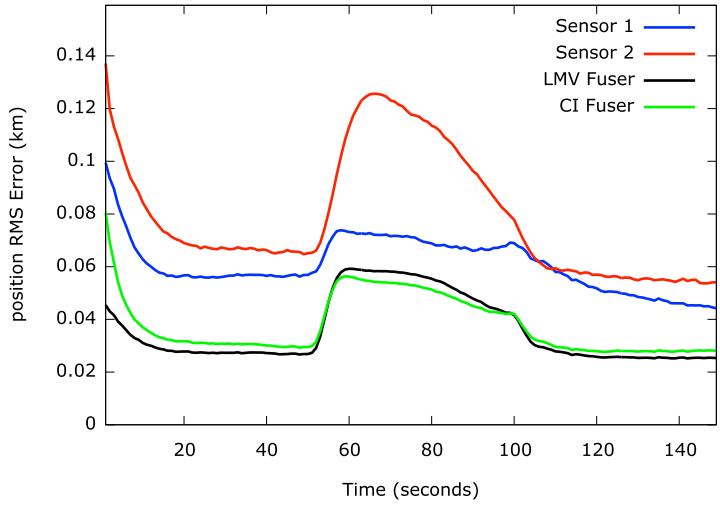
where $D(p_A, p_B)$ is the Kullback-Leibler (KL) divergence from $p_A(\cdot)$ to $p_B(\cdot)$, and $\omega_2 = 1 - \omega_1$. When the underlying estimates are Gaussian, the KL divergence can be computed as:

$$D(P_i, P_j) = \frac{1}{2}\left[\ln\frac{|P_j|}{|P_i|} + d_x^T P_j^{-1} d_x + Tr(P_i P_j^{-1}) - k\right] \tag{3.1.7}$$

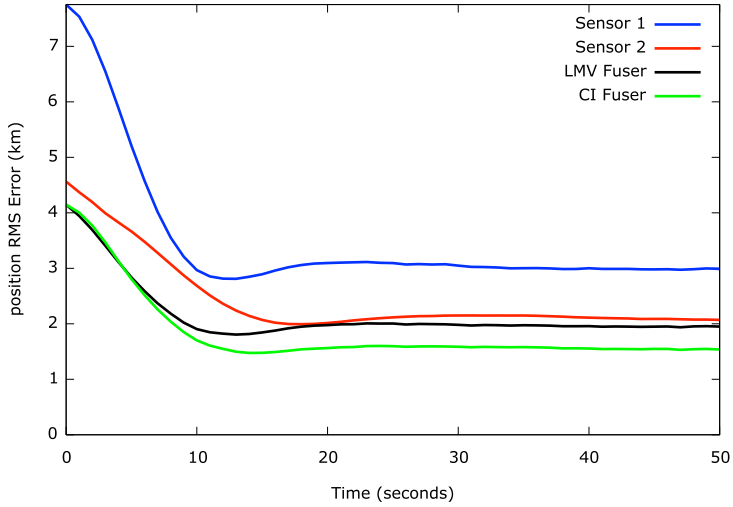
where $d_x = \hat{x}_i - \hat{x}_j$, k is the dimensionality of $\hat{x}_i$, and $|\cdot|$ denotes the determinant. Note that in the case where $P_1 = P_2$, we have $\omega_1 = \omega_2 = 0.5$, and the resulting fused estimate will be equivalent to that of the LMV fuser but with an inflated error covariance matrix (increased by a factor of two).

3.1.3 Simulations (Linear Fusers)

The simulation results shown below in Figure 3.1 demonstrate the benefits of fusing results from multiple sensors over individual sensors by examining the RMS error of each linear fuser and sensor. Figure 3.1 shows the error for two individual sensors, along with the error for the LMV and CI fusers for the maneuvering target and the ballistic target. In both cases, the LMV and CI fusers are able to fuse the sensor data to obtain more accurate results.



(a) Position RMS error for the Maneuvering Target.



(b) Position RMS error for the Ballistic Target.

Figure 3.1: RMS error in position for the (a) Maneuvering and (b) Ballistic Target. The LMV and CI fusers outperform the individual sensors.

3.2 Cross-Covariance Across Sensors

A major issue encountered with fusing sensor tracks is dealing with these state estimate error correlations across sensors. In particular, the cross-covariance, which is a measure of such correlations, may be nonzero due to a variety of factors such as common process noise and correlated measurement noise across sensors. The cross-covariance is a key component in several fusion strategies such as the LMV fuser, among others, (e.g., the best linear unbiased estimation (BLUE) and optimal weighted least squares (WLS) fusion rules [27]), but explicit estimation of the cross-covariance can be quite involved [23]. There are generally two approaches to the fusion of state estimates; one approach attempts to fuse estimates with unavailable cross-covariance, and the other approach requires knowledge of the cross-covariance [26]. In the case of unavailable cross-covariance, one may try to estimate the cross-covariance or fuse the data without it.

Learning approaches may bypass the need to compute the cross-covariance by implicitly incorporating the correlations into the fuser function that is estimated from measurements. It is noted that the cross-covariance could actually be computed off-line if the underlying dynamic system were linear and time-invariant [28]; however, in many target tracking applications, the underlying system is often nonlinear. The aim in this section is to investigate the benefits of estimating and utilizing the cross-covariance.

3.2.1 Methods for Estimating Cross-Covariance

Several methods for dealing with the cross-covariance are described in [29] and include estimation by time averaging, fusion using pseudo-measurements, etc. We can actually estimate the inter-sensor covariance in a similar manner to the state estimate error covariance if we have information from the sensors such as the filter gain and measurement matrices, which we will do in this section for comparison purposes. The method for estimating the inter-sensor covariance (i.e., cross-covariance) is as follows.

We have the following discrete-time state equation and measurement equations:

$$\begin{aligned} x(k) &= F(k, k-1)x(k-1) + v(k, k-1) \\ z_i(k) &= H_i(k)x(k) + w_i(k) \end{aligned} \tag{3.2.1}$$

where i is the index of the sensor. The state estimate from sensor i is given as:

$$\hat{x}_i(k|k) = \hat{x}_i(k|k-1) + W_i(k)[z_i(k) - H_i(k)\hat{x}_i(k|k-1)] \tag{3.2.2}$$

where $\hat{x}_i(k|k-1)$ is the predicted state estimate and $W_i(k)$ is the filter gain.

For the case where there are two sensors, the cross-covariance between the state estimates from the two sensors can be predicted by using the cross-covariance at the previous time step, $P_{12}(k-1|k-1)$, the assumed transition matrix F, and the assumed process noise covariance Q:

$$P_{12}(k|k-1) = F(k-1)P_{12}(k-1|k-1)F(k-1)^T + Q(k-1) \tag{3.2.3}$$

We can update the cross-covariance estimate using the following derivation [30]:

$$\begin{aligned} P_{12}(k|k) &\triangleq E[(x(k) - \hat{x}_1(k|k))(x - \hat{x}_2(k|k))^T] \\ &= E[(x(k) - (\hat{x}_1(k|k-1) + W_1(k)[z_1(k) - H_1(k)\hat{x}_1(k|k-1)])) \\ &\quad \cdot (x(k) - (\hat{x}_2(k|k-1) + W_2(k)[z_2(k) - H_2(k)\hat{x}_2(k|k-1)]))^T] \\ &= E[((x(k) - \hat{x}_1(k|k-1)) - W_1(k)[H_1(k)x(k) + w_1(k) - H_1(k)\hat{x}_1(k|k-1)])) \\ &\quad \cdot ((x(k) - \hat{x}_2(k|k-1)) - W_2(k)[H_2(k)x(k) + w_2(k) - H_2(k)\hat{x}_2(k|k-1)]))^T] \\ &= E[((x(k) - \hat{x}_1(k|k-1)) - W_1(k)H_1(k)x(k) - W_1(k)w_1(k) + W_1(k)H_1(k)\hat{x}_1(k|k-1))) \\ &\quad \cdot ((x(k) - \hat{x}_2(k|k-1)) - W_2(k)H_2(k)x(k) - W_2(k)w_2(k) + W_2(k)H_2(k)\hat{x}_2(k|k-1)]))^T] \\ &= E[((I - W_1(k)H_1(k))(x(k) - \hat{x}_1(k|k-1)) - W_1(k)w_1(k))) \\ &\quad \cdot ((I - W_2(k)H_2(k))(x(k) - \hat{x}_2(k|k-1)) - W_2(k)w_2(k)))^T] \end{aligned}$$

$$= E[((I - W_1(k)H_1(k))(x(k) - \hat{x}_1(k|k-1)))((I - W_2(k)H(_2k))(x(k) - \hat{x}_2(k|k-1)))^T] + E[W_1(k)w_1(k)(W_2(k)w_2(k))^T]$$

$$= (I - W_1(k)H_1(k))E[(x(k) - \hat{x}_1(k|k-1))(x(k) - \hat{x}_2(k|k-1))^T](I - W_2(k)H_2(k))^T + W_1(k)E[w_1(k)w_2(k)^T]W_2(k)^T$$

$$= \boxed{(I - W_1(k)H_1(k))P_{12}(k|k-1)(I - W_2(k)H_2(k))^T}, \tag{3.2.4}$$

which uses the assumption that the measurement noise is zero-mean and independent across sensors so that several cross-terms in this inter-sensor covariance derivation drop out. Eq. (3.2.4) can be used to update the predicted cross-covariance computed using Eq. (3.2.3), and this updated estimate, $P_{12}(k|k)$, can be utilized directly in the LMV fuser in Eq. (3.1.1), also noting that $P_{12}(k) = P_{21}(k|k)^T$.

3.2.2 Simulations (Cross-Covariance Estimation)

We can test the benefits of estimating and using the inter-sensor covariance by incorporating it into the LMV Fuser and comparing the results. The cross-covariance is first initialized to zero. Figure 3.2 shows the error for the LMV fuser with and without using the inter-sensor covariance estimate, and the CI fuser is shown as well. While the LMV fuser with the inter-sensor covariance estimate initially provides the lowest error , it seems to converge to the CI fuser error as time goes on, thus indicating that the CI fuser may be performing optimally for the ballistic target simulations.

Ballistic Coast Target

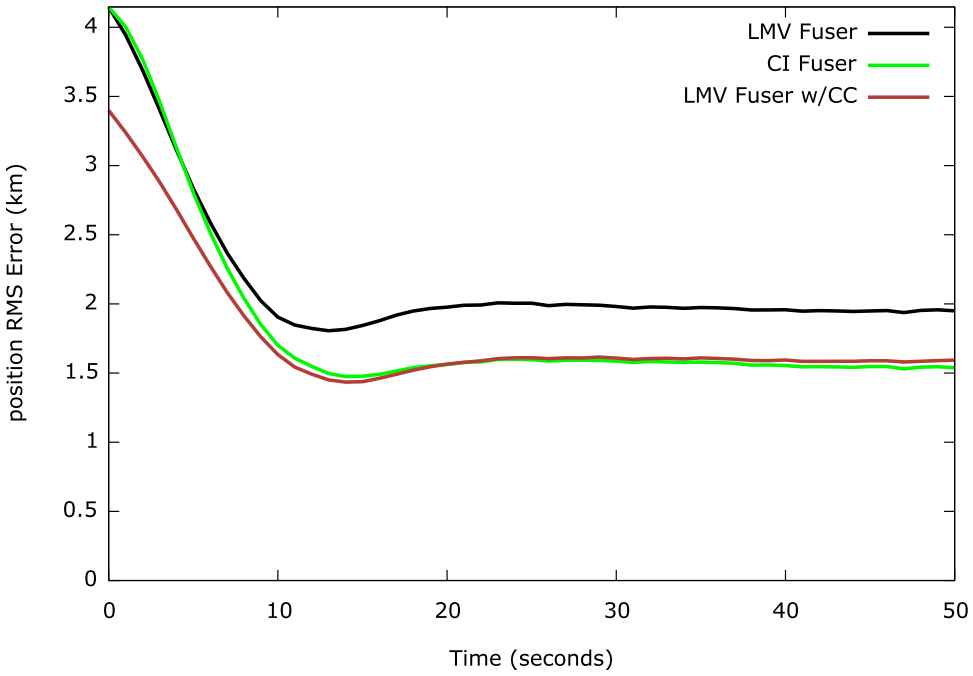


Figure 3.2: Position RMS error in km for the maneuvering target, comparing the LMV fuser with (labeled 'LMV Fuser w/CC' – the 'CC' stands for Cross-Covariance) and without (labeled 'LMV Fuser') the cross-covariance, and the CI fuser.

Simulations were only performed for the ballistic target since the maneuvering target has multiple modes, thus making it more difficult to estimate the inter-sensor covariance using the method described earlier.

3.3 Nonlinear Fusers

There are a number of nonlinear fusers that can be trained to combine state estimates, since many types of regression analysis methods exist that can be used to learn or compute the parameters of the fusing function we wish to estimate. It is noted that in most applications, field tests may be performed using the sensor networks to collect test measurements (i.e., the training data). We propose to use these measurements collected *a priori* to learn ways to fuse the data; in particular, we investigate the use of Support Vector Regression (SVR),

the Nadaraya-Watson (NW) Estimator, the Nearest Neighbor (NN) Projective Fuser, and artificial neural networks (ANNs) for multisensor fusion.

The use of nonlinear fusion methods for sensor fusion has been explored in previous works. Rao examined the use of the NN Projective Fuser for generic sensor fusion in [31], the Nadaraya-Watson estimator for sensor fusion in [32], and Artificial Neural Networks in [33] and found favorable results. Therefore, these nonlinear fusers will be compared along with SVR for nonlinear fusion. Support Vector Machines have previously been used in multisensor data fusion for classifying targets [34–36]. ANNs and SVR take different approaches to approximating the desired function; ANNs are typically designed to minimize the error between the estimated function and the target function and possess the capability of modeling arbitrary mappings [37], as long as a sufficient number of training samples are available from the same distribution and the ANN has a sufficient number of nodes and layers. SVR attempts to instead minimize the error bound between the estimated and target function to try and achieve better generalization [38]. ANNs have also been proposed for target tracking applications; e.g., for improving data association [39], filtering [40–42], and measurement fusion [43], to name a few. Chowdhury [44] and Fong et al. [45] proposed using ANNs for sensor fusion in target tracking, where the neural networks are used to determine the weights for linearly combining sensor state estimates. Neural networks have also been applied for sensor fusion in many other applications, such as automatic target recognition [46–48], audio-visual speech recognition [49], image fusion [50, 51], fermentation process control [52], edge map fusion [53], multimedia analysis [54], and weather [55]. All of these learning-based methods provide us with the ability to use nonlinear functions for fusing the data, which can potentially yield better results than with linear fusion.

3.3.1 Support Vector Regression (SVR)

Support Vector Regression was originally formulated by Vapnik [56] using the ϵ-insensitive loss function. In this work, we investigate the use of a type of SVR called ν-SVR, which is

an extension of Vapnik's well-known ϵ-SVR [56]. We will begin with a brief description of ϵ-SVR.

The regression estimate can be written as [57]

$$f(\mathbf{x}) = (\mathbf{w} \cdot \mathbf{x}) + b, \quad \mathbf{w}, \mathbf{x} \in \Re^N, b \in \Re, \tag{3.3.1}$$

Given a set of training data $(\mathbf{x}_1, y_1), ..., (\mathbf{x}_l, y_l) \in \Re^N \times \Re$ where $\mathbf{x}_i$ are the inputs and y_i are the desired targets, the coefficients $\mathbf{w}$ and b in Eq. (3.3.1) are found by minimizing the regularized risk functional

$$\|\mathbf{w}\|^2/2 + C\left(\frac{1}{l}\sum_{i=1}^{l}|y_i - f(\mathbf{x}_i)|_\epsilon\right) \tag{3.3.2}$$

where $|y_i - f(\mathbf{x}_i)|_\epsilon$ is the ϵ-insensitive loss function devised by Vapnik [56]:

$$|y - f(x)|_\epsilon = \max\{0, |y - f(x)| - \epsilon\}. \tag{3.3.3}$$

Minimizing the first term in the regularized risk functional (Eq. (3.3.2)) will lead to a smoother function, and the second term represents minimization of the error between the target data and the predicted function (although errors less than some ϵ will not be penalized).

The minimization of Eq. (3.3.2) is equivalent to the following constrained optimization problem:

$$\begin{aligned} &\text{minimize} \quad f(\mathbf{w}, \xi, \xi^*) = \frac{1}{2}\|\mathbf{w}\|^2 + C \cdot \frac{1}{l}\sum_{i=1}^{l}(\xi_i + \xi_i^*) \\ &\text{subject to} \begin{cases} y_i - \langle \mathbf{w}, \mathbf{x}_i \rangle - b \leq \epsilon + \xi_i \\ \langle \mathbf{w}, \mathbf{x}_i \rangle + b - y_i \leq \epsilon + \xi_i^* \\ \xi_i, \xi_i^* \quad \geq 0 \end{cases} \end{aligned} \tag{3.3.4}$$

The parameter ϵ represents the desired accuracy of the approximation and must be specified before training, but one may simply want ϵ to be as low as possible without having to commit to a specific level of accuracy *a priori*. Scholkopf et. al. present a modification of this ϵ-SVR algorithm in [57] where they introduce a new variable, ν, so that one may automatically minimize ϵ. The optimization function for ν-SVR for a constant $\nu \geq 0$ is given by

$$\underset{\mathbf{w},\xi,\xi^*,\epsilon}{\text{minimize}} \quad f(\mathbf{w},\xi,\xi^*,\epsilon) = \frac{1}{2}\|\mathbf{w}\|^2 + C \cdot \left(\nu\epsilon + \frac{1}{l}\sum_{i=1}^{l}(\xi_i + \xi_i^*)\right), \tag{3.3.5}$$

subject to the same constraints as in Eq. (3.3.4). As a result, in this approach the ν variable has some nice properties if the resulting ϵ is nonzero: 1) $0 \leq \nu \leq 1$ can be used to control the number of errors as it is shown in [57] that in this case, ν is an upper bound on the fraction of errors, and 2) ν is the lower bound on the fraction of support vectors (SVs), so in using ν-SVR, the user has the capability to control the number of SVs and errors.

3.3.2 Nadaraya-Watson (NW) Estimator

The Nadaraya-Watson estimator, or kernel regression, is a nonparametric regression technique where the regression function does not take on a specific form, but rather it is estimated based on available data (e.g., the 'training data' collected from field tests). The Nadaraya-Watson estimator is given by

$$\hat{f}(\mathbf{x}) = \frac{\sum_{i=1}^{l} K\left(\mathbf{x} - \mathbf{x}_i\right) y_i}{\sum_{i=1}^{l} K\left(\mathbf{x} - \mathbf{x}_i\right)} \tag{3.3.6}$$

where $\mathbf{x}$, for example, is a vector containing the state estimates that we wish to fuse, $\mathbf{x}_i$ is the i^{th} training sample, y_i is the target value for the i^{th} sample, and $K(\cdot)$ is a *kernel function*, e.g., a 'Gaussian' kernel, which is given by

$$K(\mathbf{x} - \mathbf{x}_i) = \exp(-\|\mathbf{x} - \mathbf{x}_i\|^2/2\sigma^2). \tag{3.3.7}$$

Each fused state (e.g., the position in the x-coordinate) is computed separately using Eq. (3.3.6).

3.3.3 Nearest Neighbor (NN) Projective Fuser

The general method for projective fusers are to first compute the error regressions of the sensors from the collected training data. The estimator that corresponds to the lower envelope of regressions is then projected as the fused estimate. It was shown in [58] that for the Nearest Neighbor (NN) Projective Fuser, as the sample size goes to infinity, the fuser is at least as good as the best subset of sensors in a probabilistic sense. A description of the NN Projective Fuser from [58] will be provided here for convenience.

To apply the NN Projective Fuser, the space of the input data $X_1, X_2, ..., X_l$ is first partitioned into a set of Voronoi regions $V(X_1), V(X_2), ..., V(X_l)$, such that

$$V(X_j) = \{X : \|X - X_j\| < \|X - X_k\| \text{ for all } k = 1, 2, ..., l; k \neq j\} \tag{3.3.8}$$

where $\|\cdot\|$ is the Euclidean metric. (Note that points that are equidistant from more than one sample point can be arbitrarily assigned to one of the regions).

Let $NN(X) = k$ such that $X \in V(X_k)$ for some k, which is the Voronoi cell that the data point X belongs to. For the cell $V(X_{NN(X)})$ that contains X, we identify the estimator that achieves the lowest empirical error at the sample point $X_{NN(X)}$ by defining the *estimator index* of X as follows:

$$i_{NN(X)} = \underset{i=1,2,...,N}{\arg\min} \left[f(X_{NN(X)}) - \hat{f}_i(X_{NN(X)}) \right]^2. \tag{3.3.9}$$

where $\hat{f}_i$ for $i = 1, 2, ..., N$ are given estimators of the true function, f. $i_{NN(X)}$ is essentially the index of the estimator that achieves the least empirical error at the sample point $X_{NN(X)}$

nearest to X. Then, the NN Projective Fuser is defined as:

$$\hat{f}_{NN}(X, \hat{f}_1(X), ..., \hat{f}_N(X)) = \hat{f}_{i_{NN}(X)}(X). \tag{3.3.10}$$

In other words, $\hat{f}_{NN}(X)$ is equal to the $\hat{f}_i(X)$ that achieves the least empirical error at the nearest sample point to X. So, given a test point, one first determines the Voronoi region that contains it, and the estimator that has the lowest error in that region is used as the final predictor.

3.3.4 Artificial Neural Network (ANN) Fuser

We consider a three-layer feedforward neural network, whose overall architecture is shown in Fig. 3.3. This network consists of an input layer, a hidden layer, and an output layer,

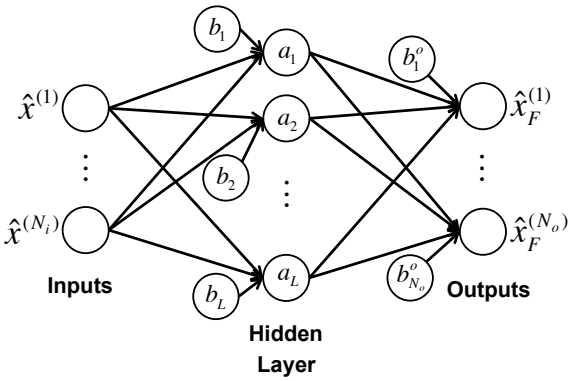


Figure 3.3: Example architecture of a simple three-layer neural network.

interconnected by weights (to be determined) which are represented by the arrows between the layers. The inputs $\hat{x}^{(1)}, ..., \hat{x}^{(N_i)}$, for example, can be the state estimates from the sensors, and the outputs $\hat{x}_F^{(1)}, ..., \hat{x}_F^{(N_o)}$ are the global (fused) state estimates. There is also a bias unit that is connected to each node in addition to the input nodes.

The nodes in the hidden layer are referred to as hidden nodes. The output of the j^{th} hidden node, a_j, is given by

$$a_j = g_1\left(\sum_{i=1}^{N_i} w_{ij}\hat{x}^{(i)} + b_j\right) \tag{3.3.11}$$

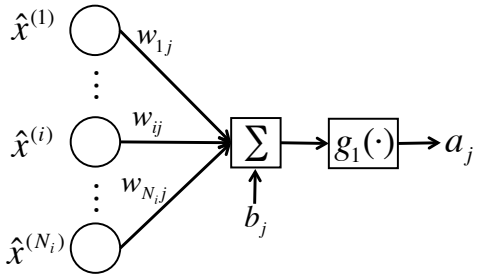


Figure 3.4: Node function diagram. The inputs are multiplied by weights for that hidden node, summed, and then passed through a function to produce a hidden node output, a_j.

where the parameters w_{ij} and b_j are typically called the weights and biases, respectively. $\hat{x}^{(i)}$ is an input feature (e.g., a state estimate from a sensor), and $g_1(\cdot)$ is a nondecreasing function called the activation function, which is typically a bounded function such as the sigmoid. A simple diagram illustrating this node function is shown in Fig. 3.4. If we concatenate all of the hidden node outputs a_j into a vector $\mathbf{a} = [a_1, ..., a_L]^T$, then we can write the hidden node outputs as:

$$\mathbf{a} = g_1(W_H^T \hat{\mathbf{x}} + \mathbf{b}_H) \tag{3.3.12}$$

where $W_H = [w_{ij}]_{N_i \times L}$ is the matrix of weights that are multiplied by the inputs $\hat{\mathbf{x}} = [\hat{x}^{(1)}, ..., \hat{x}^{(N_i)}]^T$, and $\mathbf{b}_H = [b_1, ..., b_L]^T$ is a vector of the biases for each hidden node. The fused output of our network, $\hat{\mathbf{x}}_F$, which is an N_o-dimensional vector, is then given by

$$\hat{\mathbf{x}}_F = g_2(W_o^T \mathbf{a} + \mathbf{b}_o) \tag{3.3.13}$$

where $W_o = [w_{ij}^o]_{L \times N_o}$ is another weight matrix, $\mathbf{b}_o = [b_1^o, ..., b_{N_o}^o]^T$ is the a vector of biases for each output, and $g_2(\cdot)$ is another activation function.

3.3.5 Simulations (Nonlinear Fusers)

In this section, we show the baseline performance for all of the linear and nonlinear fusers described in this chapter for a maneuvering target and a ballistic target, using the models and algorithms described in Chapter 2 for our simulations. The test targets used in all simulations

are as described in Section 2.6 and will briefly be described again in this subsection for convenience. In order to train the fusers, we use the state estimates themselves as the inputs to train and test the fusers.

Preprocessing

For the ANN fuser, however, the state estimates will be preprocessed prior to training and testing. The state estimates, which (in our cases) are the estimated position and velocity of the target, may have fairly dissimilar ranges. For example, as can be seen from Figure 2.7a, the x-coordinate for the ballistic target has an approximate range of 110km to 165km, whereas the y-coordinate has an approximate range of 3950km to 4110km. For ANNs, when the relative magnitudes of the input variables vary to a large extent, the variables with larger magnitudes may mask the effect of the variables with smaller magnitudes. In other words, the typical sizes of the inputs may not accurately reflect their relative importance in determining the required outputs [59]. Therefore, we would like to scale the input variables so that they fall within the same range (e.g., from -1 to 1) using a technique called Min-Max Normalization [60]. In this approach, we use the following equation to scale the input variable x:

$$x_{scaled} = \frac{(d_{max} - d_{min})(x - x_{min})}{x_{max} - x_{min}} + d_{min} \tag{3.3.14}$$

where x_{min} and x_{max} are the minimum and maximum values, respectively, in the training data for this input variable (e.g., the estimated x-coordinate), and $[d_{min}, d_{max}]$ is our desired range, which we will set to $[-1, 1]$.

In our preprocessing, prior to training we apply Eq. 3.3.14 to both the inputs (the state estimates) and target values (the true state) and store the x_{min} and x_{max} values for the inputs and the targets separately, for each variable. When testing, we will apply Eq. 3.3.14 using the same x_{min} and x_{max} values stored from training to the test data, and to obtain the correct outputs, we will apply the inverse operation of Eq. 3.3.14 to the outputs of the neural network using the x_{min} and x_{max} values of the target values stored from the training data.

However, it is noted that in employing this preprocessing technique, it may limit the range for which the training and testing sets may differ. We would prefer to have the capability to have largely differing training and testing sets, which may not be feasible when employing this normalization technique. This subject matter will be explored further in Chapter 4.

Example 1: Maneuvering Target

The initial states of the training targets are randomly generated from a normal distribution with the mean set to the initial state of the test target (repeated here for convenience, with the position in km and velocity in km/s):

$$\begin{aligned} \mathbf{x}_{MT}(0) &= [x_0 \quad \dot{x}_0 \quad y_0 \quad \dot{y}_0 \quad \Omega_0]^T \\ &= [0 \quad 0.1 \quad 5 \quad 0.1 \quad 0]^T, \end{aligned} \tag{3.3.15}$$

with a standard deviation of 100m and 5m/s for the initial position and velocities, respectively, in both the x and y coordinates. The time at which the maneuver starts and ends, as well as the turn rate, are also randomly generated from a normal distribution where the mean for the start and stop time is 50s and 100s, respectively, both with a standard deviation of 5s, and the mean for the turn rate is 2°/s with a standard deviation of 0.1°/s.

Simulations were run for the cases where there are two and four sensors tracking the target. The CI algorithm described in Section 3.1.2 is currently only defined for two sensors, so results using the CI algorithm are only shown in the two sensors case. The resulting position Root-Mean-Squared (RMS) error for each fuser is in shown in Figure 3.5 for the two sensors. In Figure 3.5, it can be seen that the ANN and SVR Fusers are able to outperform the linear fusers.

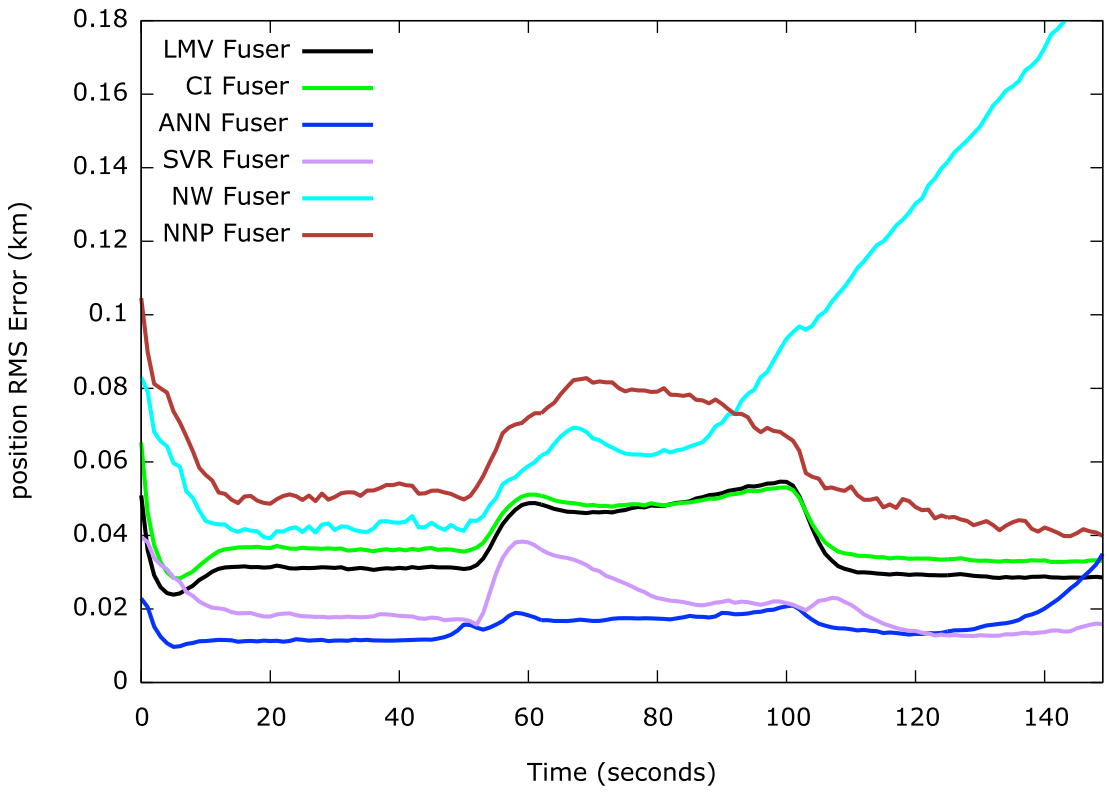


Figure 3.5: Linear and Nonlinear Fuser Performance for the Maneuvering Target.

Example 2: Ballistic Coast Target

The initial states of the training and test targets are randomly generated from a normal distribution with the mean set to the following (in km for position, and km/s for velocity) :

$$\begin{aligned} \mathbf{x}_{BT}(0) &= [x_0 \quad \dot{x}_0 \quad y_0 \quad \dot{y}_0 \quad z_0 \quad \dot{z}_0]^T \\ &= [113.75 \quad 0.94 \quad 3950 \quad 3.33 \quad 6150 \quad -6.0125]^T \end{aligned} \tag{3.3.16}$$

with a standard deviation of 100m and 5m/s for the position and velocity, respectively, in the x and y coordinates. The RMS error in the position for the different fusers is shown in Fig. 3.6 for two sensors. Twenty trajectories (each containing 100 samples) were generated and used for training the nonlinear fusers.

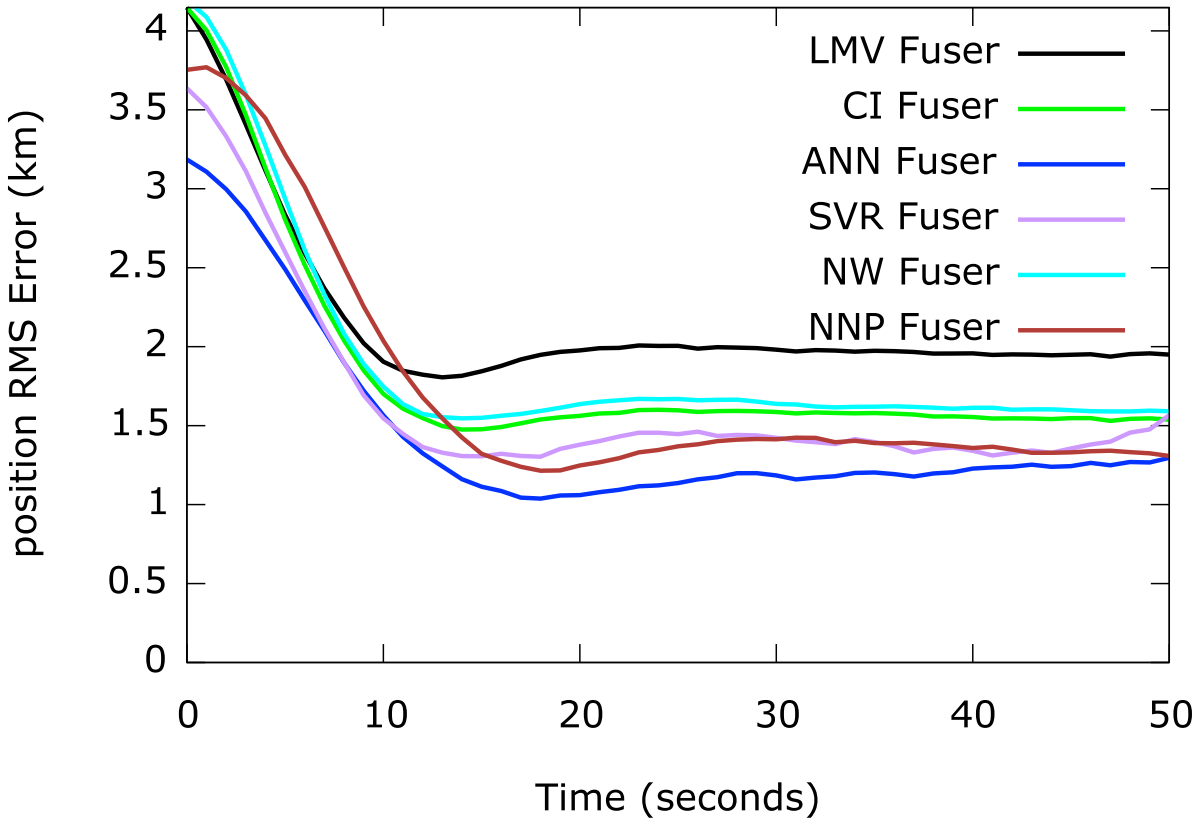


Figure 3.6: Linear and Nonlinear Fuser Performance for the Ballistic Target.

In these simulations, the ANN and SVR fusers are clearly able to outperform the linear fusers, in addition to the NW fuser and the NN Projective fuser. The ANN fuser is the best performing of all the nonlinear fusers in both the maneuvering target and ballistic target cases, so the ANN fuser was selected for further investigation for use in target tracking and will be discussed in greater detail in Chapter 4.

Chapter 4

Training the Artificial Neural Network (ANN) Fuser

In the previous chapter, it was shown through simulation that the ANN fuser was able to outperform other promising nonlinear fusers for both the maneuvering and ballistic coast targets. Therefore, we elected to proceed with the ANN fuser for further analysis. In this chapter, we intend to look deeper in two main topics: 1) ***Performance***; and 2) ***Practicality***. That is, can we further improve the ANN fuser performance, and improve its practicality? Specifically, the key questions that we investigate regarding these topics are:

1. **(Performance)**. Can we utilize additional information that may be available (e.g., the error covariance estimates, previous state estimates) to further improve our ANN fuser performance? (Subsections 4.1 and 4.4).

2. **(Practicality)**. How can we train the ANN fuser so that the training data need not be collected in such close proximity to the testing data, or can we use multiple fusers to cover the state space? (Subsections 4.2 and 4.3).

3. **(Performance)**. What parameters should we use for the ANN fuser (e.g., how many hidden nodes should we use) for the best performance? (Subsection 4.5).

4.1 Error Regularization

A learning algorithm is said to exhibit good generalization if it is able to perform accurately on new, unseen data. One way of analyzing generalization is to look at the generalization error. It is well known that the generalization error can be decomposed into two key components: bias and variance [61], where a model that is too simple will have a large bias, but a model that is too complex (i.e., flexible) will exhibit a large variance. In practice, there is always a trade-off between these two error components. The best generalization is obtained when we have the best compromise with a small bias and a small variance, and to find this optimum balance, it is necessary to control the effective complexity of the model [59]. There are primarily two approaches to controlling model complexity in neural networks. One is called structural stabilization [59], where the number of adaptive parameters in the network is varied (e.g., by varying the number of hidden nodes or the number of connections), with a preliminary analysis presented in Section 4.5. The other approach makes use of *regularization*, where additional information is introduced, typically in the form of a penalty for model complexity (e.g., restricting the function to be smooth). Regularization techniques for neural networks have been widely studied over the past few decades, and it has been shown to improve the generalization capabilities of neural networks. In this section we propose a weighted error regularization based on the inherent noise properties of the data and demonstrate improved performance over the baseline ANN fuser. Our proposed method is also compared to other forms of error regularization.

4.1.1 Existing Methods for Regularization in ANNs

Typically, the ANN weights are determined by minimizing an error function, e.g., the sum of squared errors:

$$S(\mathbf{w}) = \sum_{i=1}^{m} (t_i - f(\mathbf{x}_i, \mathbf{w}))^2 \tag{4.1.1}$$

where $\mathbf{w}$ is a vector of the ANN weights, m is the number of training samples, t_i is the (known) target for the i^{th} training sample, and $f(\mathbf{x}_i, \mathbf{w})$ is the ANN output for the i^{th} training sample. In regularization, smoother mappings are encouraged with the addition of a penalty term to the error function. The error function then typically takes the form

$$\tilde{S}(\mathbf{w}) = S(\mathbf{w}) + \lambda\Omega \tag{4.1.2}$$

The parameter λ controls the degree to which the penalty term Ω influences the form of the solution [59]. Several methods of regularization are as follows.

Weight Decay/Ridge Regression

Weight decay, also known as ridge regression, is one of the simplest forms of regularizers:

$$\tilde{S}(\mathbf{w}) = \sum_{i=1}^{m}(t_i - f(\mathbf{x}_i, \mathbf{w}))^2 + \lambda\mathbf{w}^T\mathbf{w} \tag{4.1.3}$$

It is known as *weight decay* since this function encourages the weights to go to zero, unless supported by the data [59], which effectively limits the model complexity (by promoting a smoother function) to avoid over-fitting. However, this introduces another problem where now another parameter needs to be selected: the regularization coefficient λ. This value can be chosen through methods such as cross-validation [37].

Bayesian Regularization

McKay [62] proposed a Bayesian framework that actually introduces two *hyperparameters*, α and β, which are also used to trade off between the minimizing the sum of squared errors or the sum of squared weights, but the trade-off is adaptive depending on the network parameters and corresponding errors. (Note that α and β are referred to as *hyperparameters* since they control the distribution of other parameters (weights and biases), so they are labeled as hyperparameters to distinguish them from the actual parameters of interest).

This approach also avoids the need for cross-validation to determine appropriate values for additional parameters. The cost function can be written as:

$$\begin{aligned}\tilde{S}(\mathbf{w}) &= \beta \sum_{i=1}^{m} (t_i - f(\mathbf{x}_i, \mathbf{w}))^2 + \alpha \mathbf{w}^T \mathbf{w} \\ &= \beta \cdot E_d + \alpha \cdot E_w\end{aligned} \tag{4.1.4}$$

where E_d is the sum of squared errors, and E_w is the sum of squared weights.

The Levenberg-Marquardt (LM) method can be used to train the neural network, where α and β are updated at each iteration according to the following algorithm:

$$\gamma = r - \alpha \cdot Tr(\nabla^2 S(\mathbf{w})^{-1}) \tag{4.1.5}$$

$$\beta = (m - \gamma)/(2E_d) \tag{4.1.6}$$

$$\alpha = \gamma/(2E_w) \tag{4.1.7}$$

where r is the number of parameters (the number of weights and biases), m is the number of training patterns, and $Tr(\nabla^2 S(\mathbf{w})^{-1})$ is the trace of the inverse Hessian matrix of $S(\mathbf{w})$.

Training with Noise

Several works (e.g., [63, 64]) have demonstrated that in practice, injecting noise into the training data oftentimes yields better generalization results. In [65], Bishop showed that training with noise is actually closely related to regularization. To employ this method, noise is simply added into duplicates of the available training data.

4.1.2 Training the ANN

Before going into further details on the proposed regularization, we will provide some background on actually training an ANN. When the target outputs are known, a well-known approach to determining the neural network parameters is called backpropagation. Back-

propagation is based on gradient descent; the weights are initialized with random values and are iteratively updated to reduce the error (according to some user-defined error function like the mean-squared error). Once the network parameters are learned (from training data), new inputs can simply be fed into the neural network to obtain the fused outputs.

The Levenberg-Marquardt (LM) algorithm [66] is a backpropagation method that is used in this work to train the ANNs as it can be implemented efficiently and is considered to be one of the faster training methods with relatively good performance. It is a second-order method in that it uses both the first and second derivatives of the error function to find a set of optimum weights. The formulation is as follows. Assume we want to minimize some function $S(\mathbf{w})$ with respect to a r-dimensional parameter vector $\mathbf{w}$. Then the Gauss-Newton method for updating $\mathbf{w}$, which is an iterative method that uses the first and second derivatives of a function to find a point where the derivative is zero, would be:

$$\Delta\mathbf{w} = -[\nabla^2 S(\mathbf{w})]^{-1}\nabla S(\mathbf{w}) \tag{4.1.8}$$

where $\nabla^2 S(\mathbf{w})$ and $\nabla S(\mathbf{w})$ are the Hessian and the gradient, respectively, of $S(\mathbf{w})$. If we let $S(\mathbf{w})$ be a sum of squares function over m training patterns, e.g.,

$$\begin{aligned} S(\mathbf{w}) &= \sum_{k=1}^{m}(y^{(k)} - f(\hat{x}^{(k)}, \mathbf{w}))^2 \\ &= \sum_{k=1}^{m}(e_k(\mathbf{w}))^2 = \mathbf{e}(\mathbf{w})^T\mathbf{e}(\mathbf{w}) \end{aligned} \tag{4.1.9}$$

where $e_k(\mathbf{w}) = y^{(k)} - f(\hat{x}^{(k)}, \mathbf{w})$ and $\mathbf{e}(\mathbf{w}) = [e_1(\mathbf{w}), ..., e_m(\mathbf{w})]^T$, then the elements of the gradient of $S(\mathbf{w})$ can be written as:

$$[\nabla S(\mathbf{w})]_j = 2\sum_{i=1}^{m} e_i(\mathbf{w})\frac{\partial e_i(\mathbf{w})}{\partial w_j} \tag{4.1.10}$$

and the elements of the Hessian are given by

$$[\nabla^2 S(\mathbf{w})]_{jk} = 2\sum_{i=1}^{m}\left(\frac{\partial e_i(\mathbf{w})}{\partial w_j}\frac{\partial e_i(\mathbf{w})}{\partial w_k} + e_i(\mathbf{w})\frac{\partial^2 e_i(\mathbf{w})}{\partial w_j \partial w_k}\right) \tag{4.1.11}$$

If we assume that the second-order derivative terms (the second term in Eq. (4.1.11)) are approximately zero, then we obtain an approximation of the Hessian:

$$[\nabla^2 S(\mathbf{w})]_{jk} \approx 2\sum_{i=1}^{m} J_{ij} J_{ik} \tag{4.1.12}$$

where $J_{ij} = \partial e_i(\mathbf{w})/\partial w_j$ and likewise, $J_{ik} = \partial e_i(\mathbf{w})/\partial w_k$, which are simply elements of the Jacobian J of $\mathbf{e}(\mathbf{w})$. Therefore, we can rewrite the weight update equation in Eq. (4.1.8) in terms of J as:

$$\Delta\mathbf{w} = -(J^T J)^{-1} J^T \mathbf{e}(\mathbf{w}) \tag{4.1.13}$$

where

$$J = \begin{bmatrix} \frac{\partial e_1(\mathbf{w})}{\partial w_1} & \cdots & \frac{\partial e_1(\mathbf{w})}{\partial w_r} \\ \vdots & \ddots & \vdots \\ \frac{\partial e_m(\mathbf{w})}{\partial w_1} & \cdots & \frac{\partial e_m(\mathbf{w})}{\partial w_r} \end{bmatrix} \tag{4.1.14}$$

The Levenberg-Marquardt modification to the Gauss-Newton method is:

$$\Delta\mathbf{w} = -(J^T J + \mu I)^{-1} J^T \mathbf{e}(\mathbf{w}) \tag{4.1.15}$$

where I is the identity matrix, and $\mu > 0$ is a damping factor, which is adjusted at each iteration of the weight update. If a step (i.e., weight update) results in an increased $S(\mathbf{w})$, then μ is multiplied by some factor β, and if a step results in a decreased $S(\mathbf{w})$, then μ is divided by β. The Jacobian of $\mathbf{e}(\mathbf{w})$ can be computed using the backpropagation approach as described in [66], where $\mathbf{w}$ is a vector containing all of the neural network parameters

more specifically as follows:

$$\mathbf{w} = [W_H(1,1), W_H(1,2), ..., W_H(L,N_i), \mathbf{b}_H(1), ..., \mathbf{b}_H(L), W_o(1,1), ..., \mathbf{b}_o(N_o)]^T. \quad (4.1.16)$$

If we have N_i network inputs, L hidden nodes, and N_o network outputs, then the dimension of $\mathbf{w}$ is $r = L(N_i + 1) + N_o(L + 1)$. In this work, the state estimates from each sensor are used as a network input, so $N_i = Ns$ if there are N sensors generating s-dimensional state estimates, and $N_o = s$ so that the neural network outputs an s-dimensional fused state estimate.

4.1.3 Proposed Weighted Error Regularization

It is noted, however, that the sensors also provide additional information regarding the state estimates: an estimate of its error covariance. It is an open question of how to best utilize this information, if at all, when designing or using these nonlinear fusers. It is proposed here that these error covariance estimates can be used when training the neural network to improve the neural network's generalization capability. If we assume that the state estimates from the sensors can be modeled as the true states plus noise, that is:

$$\hat{x}_i = \mathbf{x} + \mathbf{n}_i \quad (4.1.17)$$

where i is the sensor index, and $\mathbf{n}_i$ is zero-mean white Gaussian noise with covariance P_i, then in fusing these state estimates, the neural network will also transform and fuse the noise. We can therefore try to reduce the variance of the output in addition to the overall error by adding the output variance to the objective function that we minimize when determining the neural network parameters. But if one were to use a nonlinear activation function such as the tangent hyperbolic sigmoid function, $g_1(z) = \frac{2}{1+e^{-2z}} - 1$, then the variance becomes quite difficult to determine analytically. However, from [67], we have the following upper

bound on the variance of the function of a random variable X, if $X \sim \mathcal{N}(\mu, \sigma^2)$:

$$Var[g(X)] \leq \sigma^2 E[g'(X)]^2 \tag{4.1.18}$$

We can write the output $\hat{x}_F$ as a function of the noise $\mathbf{n} = [\mathbf{n}_1^T, ..., \mathbf{n}_N^T]^T$, where $\mathbf{n} \sim \mathcal{N}(\underline{0}, P)$. P is a block diagonal matrix where the blocks are P_1 through P_N:

$$P = \begin{bmatrix} P_1 & \mathbf{0}_{s\times s} & \cdots & \mathbf{0}_{s\times s} \\ \mathbf{0}_{s\times s} & P_2 & \cdots & \mathbf{0}_{s\times s} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0}_{s\times s} & \cdots & \mathbf{0}_{s\times s} & P_N \end{bmatrix} \tag{4.1.19}$$

Recall that s is the number of states in the true target state $\mathbf{x}$ (e.g., $s = 6$ for the ballistic coast target since we are interested in the position and velocity of the target in x-y-z coordinates). For the ANN fuser, if we let the activation function for each hidden node be $g_1(x) = \frac{2}{1+e^{-2x}} - 1$ and the output activation function $g_2(x) = x$, then the ANN fuser output would be given by

$$\begin{aligned} \hat{x}_F &= W_o^T \left(\frac{2}{1 + e^{-2(W_H^T \hat{x} + \mathbf{b}_H)}} - 1 \right) + \mathbf{b}_o \\ &= W_o^T \left(\frac{2}{1 + e^{-2(W_H^T (\mathbf{x}+\mathbf{n}) + \mathbf{b}_H)}} - 1 \right) + \mathbf{b}_o \\ &= W_o^T \left(\frac{2}{1 + e^{-2(W_H^T \mathbf{x} + \mathbf{b}_H)} e^{-2W_H^T \mathbf{n}}} - 1 \right) + \mathbf{b}_o \end{aligned} \tag{4.1.20}$$

Now if we let $\gamma = W_H^T \mathbf{n}$, which is simply a linear transformation of the Gaussian random variable $\mathbf{n} \sim \mathcal{N}(\underline{0}, P)$, then $\gamma \sim \mathcal{N}(\underline{0}, W_H^T P W_H)$. Therefore, we can write Eq. (4.1.18) as

$$Var[\hat{x}_F(\gamma)] \leq W_H^T P W_H \cdot E\left[\frac{\partial \hat{x}_F(\gamma)}{\partial \gamma} \right]^2 \tag{4.1.21}$$

However, even determining $E\left[\partial \hat{x}_F(\gamma)/\partial\gamma\right]$ is still quite complicated. Therefore, to simplify, only the term $W_H^T P W_H$ will be used to add to the error function for minimization. We can modify the LM algorithm described in Section 4.1.2 to incorporate this additional term using a tradeoff parameter λ:

$$S(\mathbf{w}) = \sum_{k=1}^{m} (y^{(k)} - f(\hat{x}^{(k)}, \mathbf{w}))^2 + \lambda \mathbf{w}^T \Sigma \mathbf{w} \tag{4.1.22}$$

where

$$\Sigma = \left[\begin{array}{c|c} P_R & \mathbf{0}_{NLs\times(r-NL)s} \\ \hline \mathbf{0}_{(r-NL)s\times NLs} & \mathbf{0}_{(r-NL)s\times(r-NL)s} \end{array}\right], \tag{4.1.23}$$

and $P_R \in \Re^{(NLs)\times(NLs)}$ is a block diagonal matrix with each block consisting of the matrix P, repeated L times (once for each hidden node):

$$P_R = \begin{bmatrix} P & \mathbf{0}_{Ns\times Ns} & \cdots & \mathbf{0} \\ \mathbf{0} & P & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \cdots & \mathbf{0} & P \end{bmatrix} \tag{4.1.24}$$

Recall that N is the number of sensors and s is the number of states, so each block diagonal matrix P is of size $Ns \times Ns$. Following the LM approach, when we update $\mathbf{w}$ with $\Delta\mathbf{w}$, we get the following update to our error function $S(\mathbf{w})$:

$$\begin{aligned} S(\mathbf{w}+\Delta\mathbf{w}) &= \sum_{k=1}^{m} (y^{(k)} - f(\hat{x}^{(k)}, \mathbf{w}+\Delta\mathbf{w}))^2 + \lambda(\mathbf{w}+\Delta\mathbf{w})^T\Sigma(\mathbf{w}+\Delta\mathbf{w}) \\ &= \sum_{k=1}^{m} (e_k(\mathbf{w}+\Delta\mathbf{w}))^2 + \lambda(\mathbf{w}+\Delta\mathbf{w})^T\Sigma(\mathbf{w}+\Delta\mathbf{w}) \\ &= \left[\mathbf{e}(\mathbf{w}+\Delta\mathbf{w})\right]^T \mathbf{e}(\mathbf{w}+\Delta\mathbf{w}) + \lambda(\mathbf{w}+\Delta\mathbf{w})^T\Sigma(\mathbf{w}+\Delta\mathbf{w}) \\ &\approx \left[\mathbf{e}(\mathbf{w}) + \nabla\mathbf{e}(\mathbf{w})\Delta\mathbf{w}\right]^T \left[\mathbf{e}(\mathbf{w}) + \nabla\mathbf{e}(\mathbf{w})\Delta\mathbf{w}\right] \\ &\qquad + \lambda(\mathbf{w}+\Delta\mathbf{w})^T\Sigma(\mathbf{w}+\Delta\mathbf{w}) \end{aligned} \tag{4.1.25}$$

where in the fourth line of Eq. (4.1.25), we approximate and substitute $\mathbf{e}(\mathbf{w}+\Delta\mathbf{w})$ with its first-order Taylor expansion about $\mathbf{w}$. We know that at the minimum of $S(\mathbf{w}+\Delta\mathbf{w})$, the gradient with respect to $\Delta\mathbf{w}$ is zero, so we have:

$$\begin{aligned}\frac{\partial S(\mathbf{w}+\Delta\mathbf{w})}{\partial\Delta\mathbf{w}} &\approx 2\left[\nabla\mathbf{e}(\mathbf{w})\right]^T\left[\mathbf{e}(\mathbf{w})+\nabla\mathbf{e}(\mathbf{w})\Delta\mathbf{w}\right]+2\lambda\Sigma(\mathbf{w}+\Delta\mathbf{w})\\ &= 2J^T\mathbf{e}(\mathbf{w})+2J^TJ\Delta\mathbf{w}+2\lambda\Sigma\mathbf{w}+2\lambda\Sigma\Delta\mathbf{w}\\ &= 0\end{aligned} \tag{4.1.26}$$

where J is the Jacobian of $\mathbf{e}(\mathbf{w})$. The weight update equation becomes:

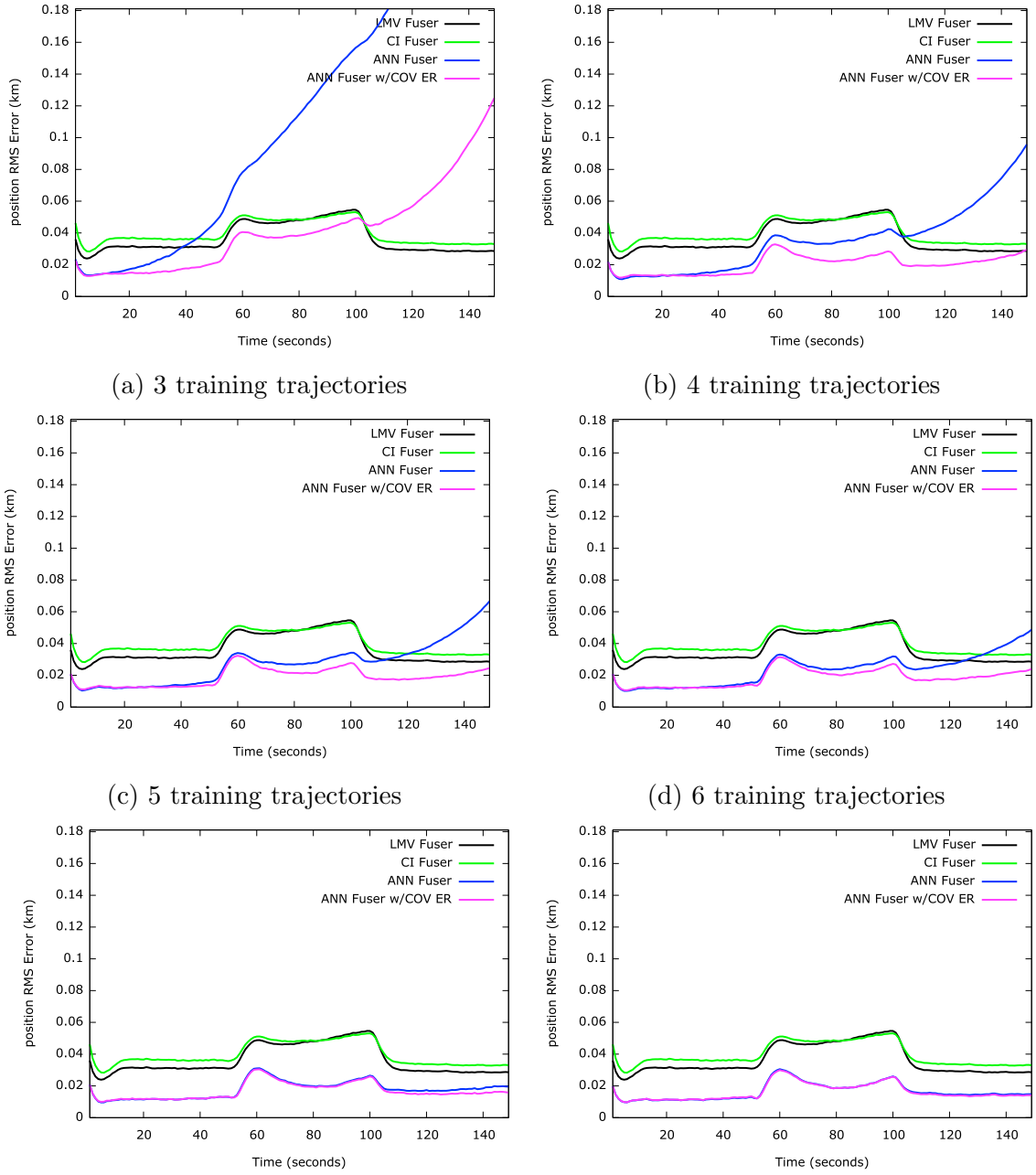
$$\Delta\mathbf{w} = -(J^TJ+\lambda\Sigma)^{-1}(J^T\mathbf{e}(\mathbf{w})+\lambda\Sigma\mathbf{w}) \tag{4.1.27}$$

and with the LM modification to include the damping factor, we obtain the final weight update equation which incorporates the error covariance estimates into the training using the LM method:

$$\boxed{\Delta\mathbf{w} = -(J^TJ+\lambda\Sigma+\mu I)^{-1}(J^T\mathbf{e}(\mathbf{w})+\lambda\Sigma\mathbf{w})} \tag{4.1.28}$$

4.1.4 Simulations (Regularization)

Figures 4.1 and 4.3 show a comparison between the baseline ANN Fuser (without error regularization), and with the proposed error regularization for the maneuvering and ballistic target, respectively. The number of training trajectories was varied from 3, 4, 5, 6, 10, to 20, and it can be seen that in using the proposed error regularization, the number of training trajectories required for obtaining better performance than the linear fusers is reduced. We can also compare the proposed error regularization to the existing regularization algorithms described in Section 4.1.1. Figures 4.2 and 4.4 show these comparison results; it can be seen in these plots that the proposed error regularization scheme also outperforms alternate existing regularizers in this application.



(a) 3 training trajectories

(b) 4 training trajectories

(c) 5 training trajectories

(d) 6 training trajectories

(e) 10 training trajectories

(f) 20 training trajectories

Figure 4.1: ANN Fuser performance with (denoted by 'w/COV ER', which stands for 'with Covariance Error Regularization') and without the proposed error regularization for the maneuvering target. Each subfigure shows the position RMS error for a different number of training trajectories.

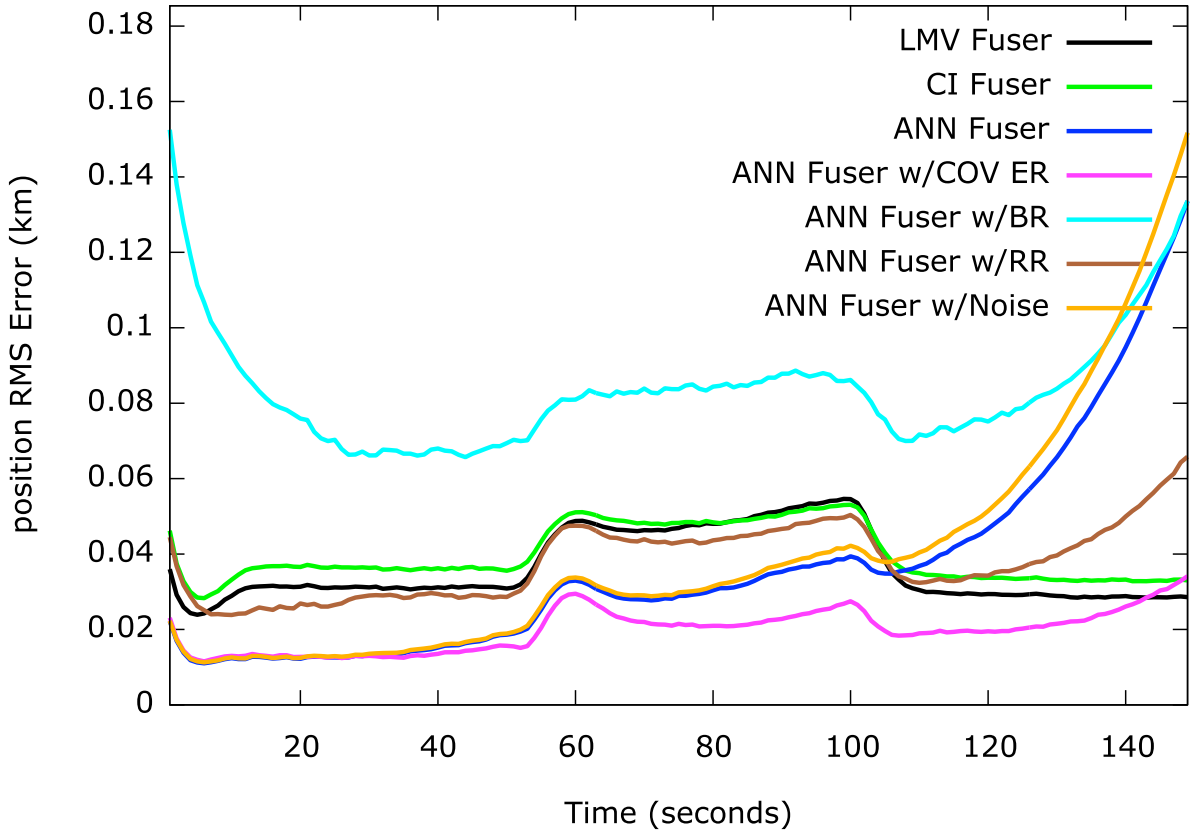
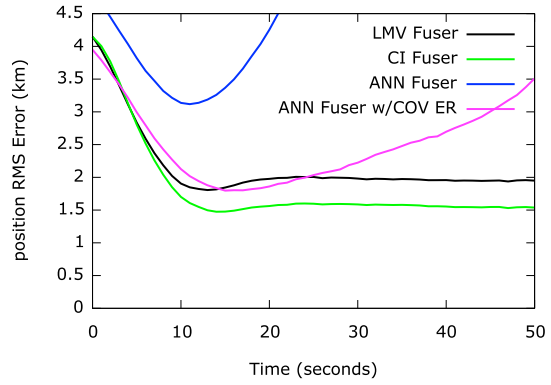
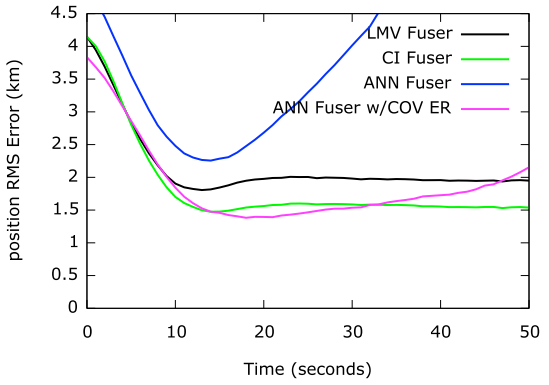


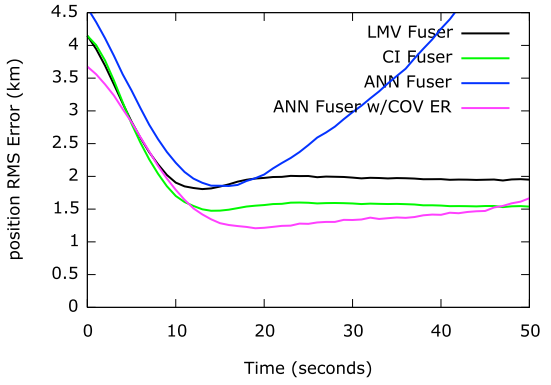
Figure 4.2: Fuser performance using various methods of regularization with the manuevering target, using 4 training trajectories. BR: Bayesian Regularization, RR: Ridge Regression, and COV ER: Covariance Error Regularization.



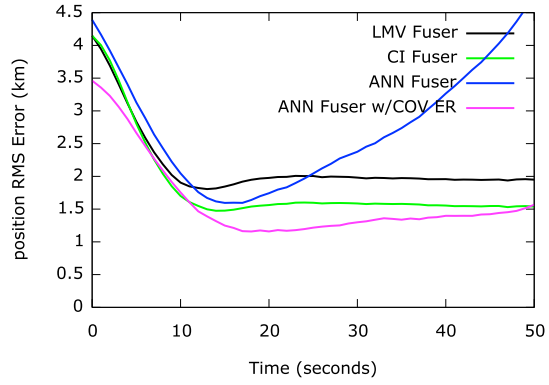
(a) 3 training trajectories



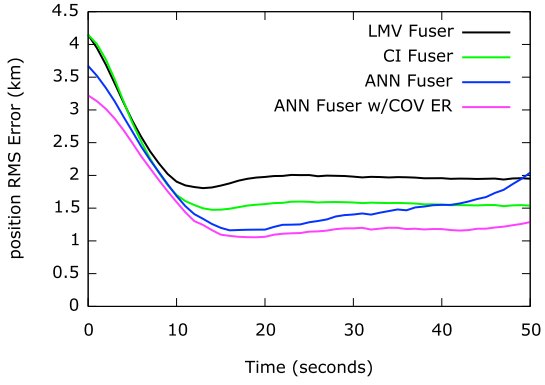
(b) 4 training trajectories



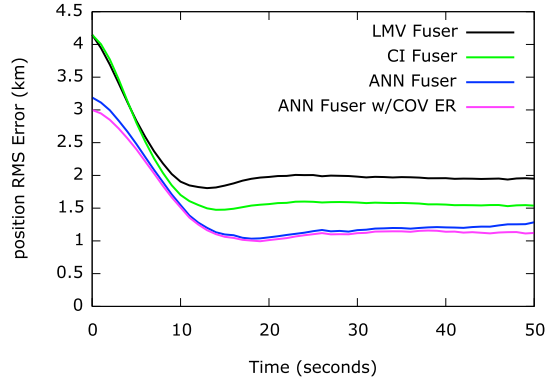
(c) 5 training trajectories



(d) 6 training trajectories



(e) 10 training trajectories



(f) 20 training trajectories

Figure 4.3: ANN Fuser performance with (denoted by 'w/COV ER', which stands for 'with Covariance Error Regularization') and without the proposed error regularization for the ballistic target. Each subfigure shows the position RMS error for a different number of training trajectories.

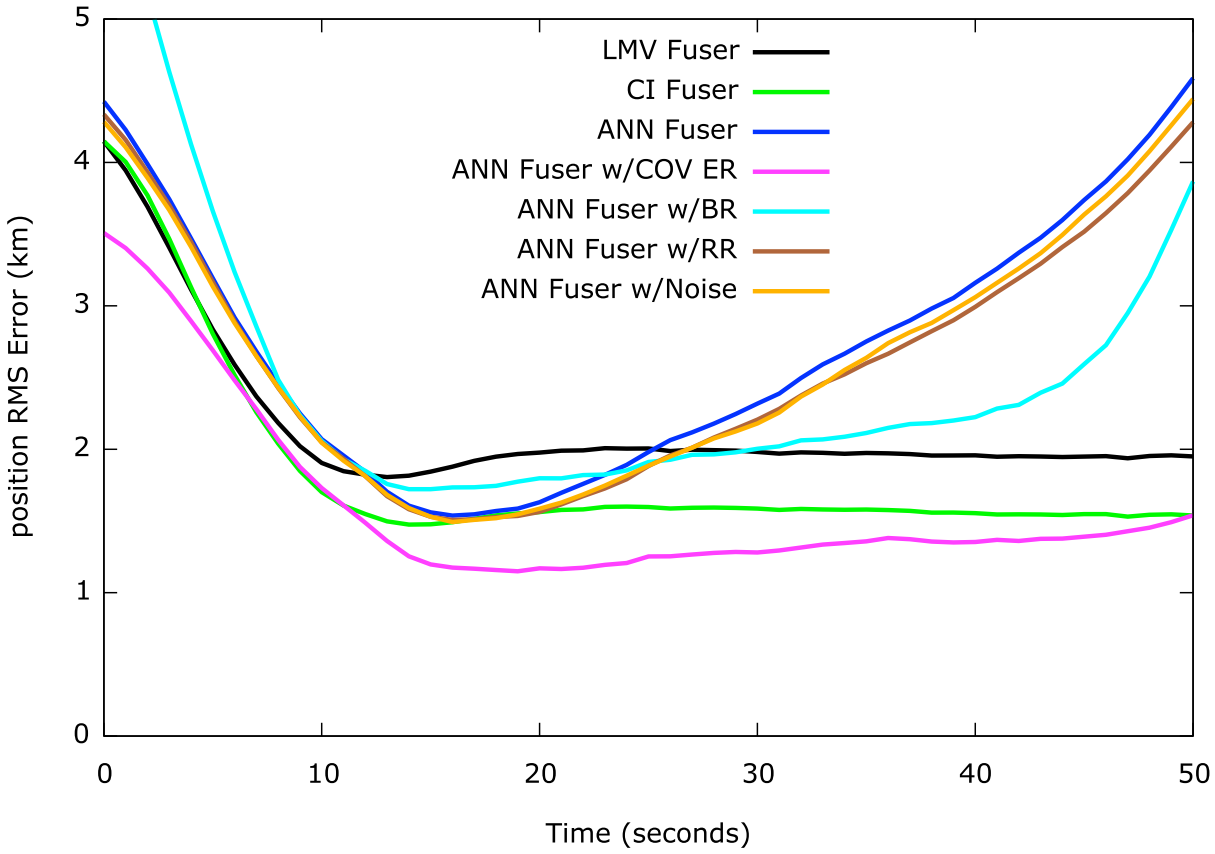


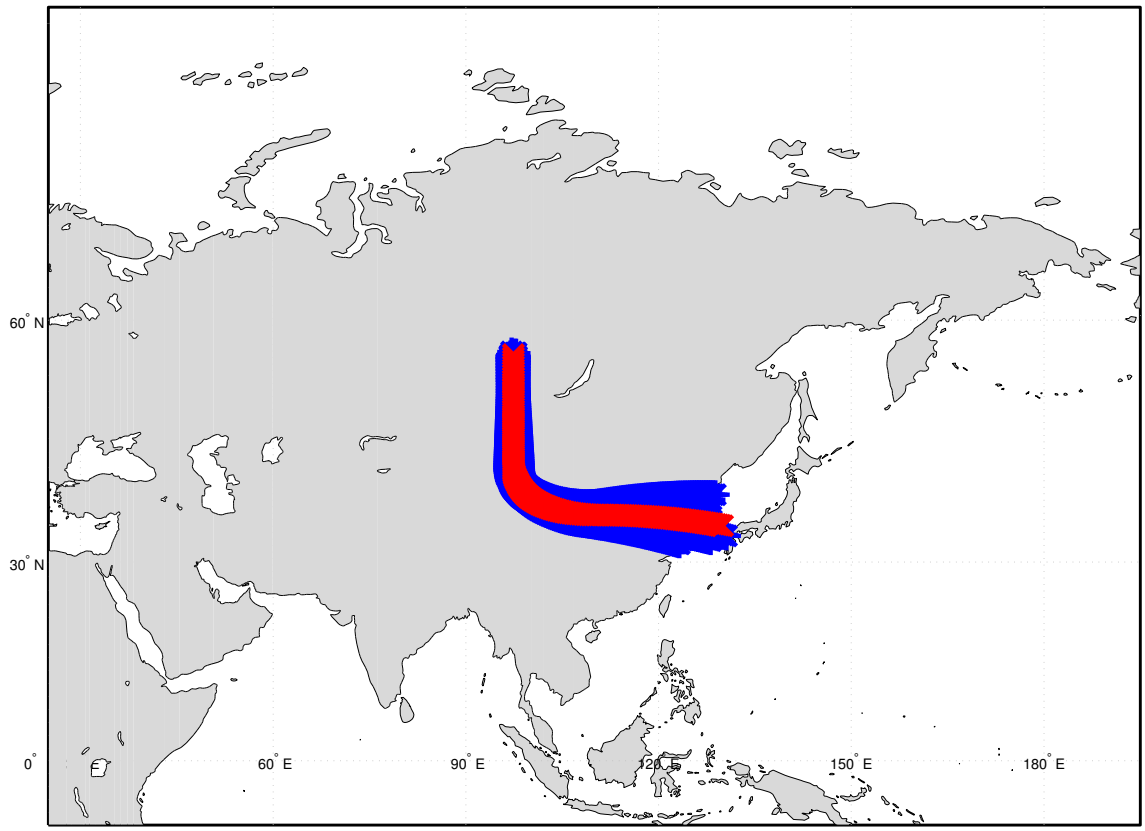
Figure 4.4: Fuser performance using various methods of regularization with the ballistic target, using 6 training trajectories. BR: Bayesian Regularization, RR: Ridge Regression, and COV ER: Covariance Error Regularization.

4.2 Significantly Different Training and Testing Sets

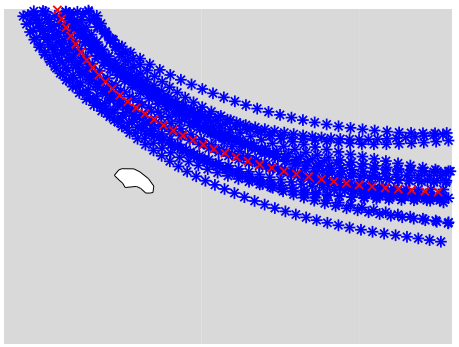
While we are able to reduce the number of training trajectories needed to obtain better performance than the linear fusers using the proposed error regularization, we still have the issue of requiring that the training trajectories be close to the test trajectory due to the use of the commonly utilized "Min-Max" normalization scheme described in Section 3.3.5. We can convert the locations of the trajectory samples to the corresponding latitude and longitude and plot these points on a world map to demonstrate the proximity. The training and testing samples for the simulated maneuvering and ballistic targets are shown in Figures 4.5 and 4.6, respectively, where the blue points indicate the training samples, and the red

points indicate the test samples. It can be seen that the training samples here are indeed very close in location to the testing samples, but such a scenario may not be feasible in a real-world situation.

We would like to be able to use test trajectories that are not in such close proximity to the training data in order to outperform linear fusers. We therefore propose the use of an alternate normalization scheme for target tracking, where instead of normalizing each dimension of the input to the ANN to be in some range such as $[-1, 1]$ (as is typically performed), we restrict each data sample to a smaller range by normalizing the data sample by its own $\mathcal{L}_2$-norm. Since the resulting fused estimate should be within the same range as the original sample, we can use that same $\mathcal{L}_2$-norm from that data sample to un-normalize it after fusion. With this approach, we will demonstrate that we can now fuse samples from test trajectories that can be a relatively long distance away from the actual training trajectories, as is evidenced by the new maneuvering and ballistic target maps in Figures 4.7 and 4.8. Simulation results based on this alternative normalization approach are shown in Section 4.2.1 for both the maneuvering target and the ballistic target.

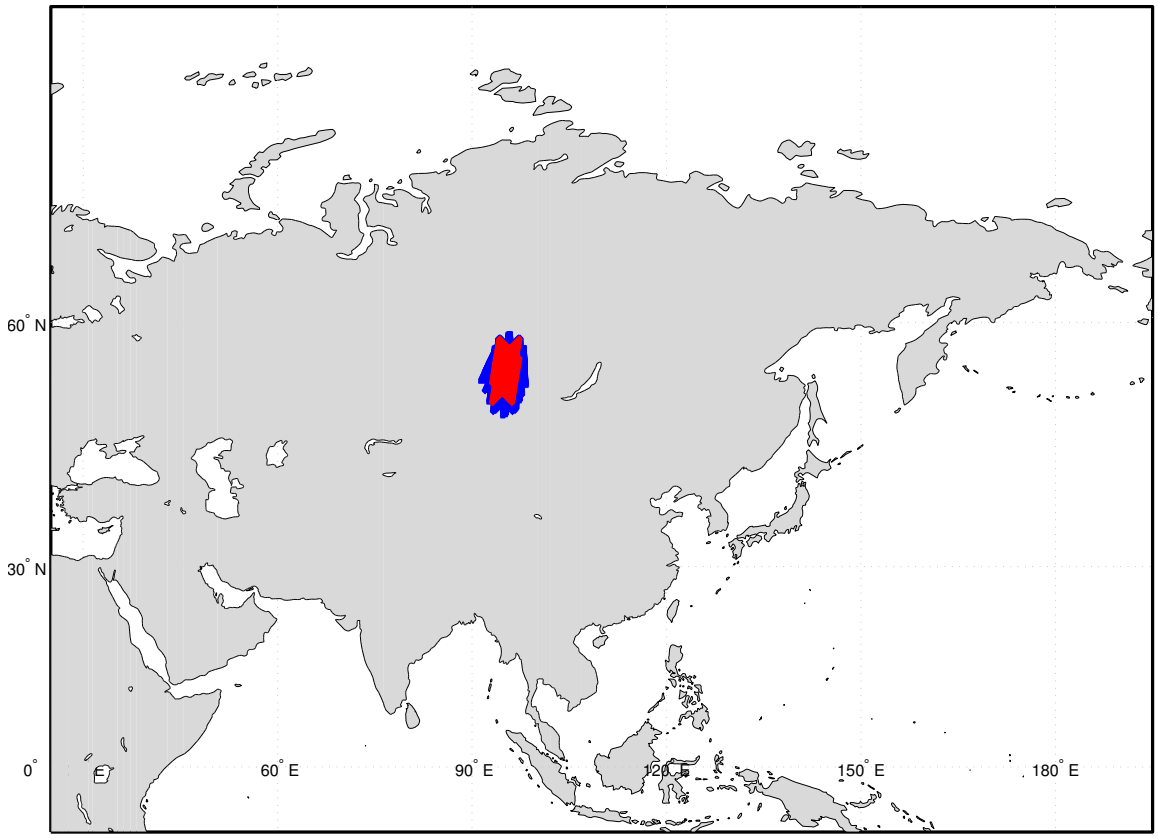


(a)

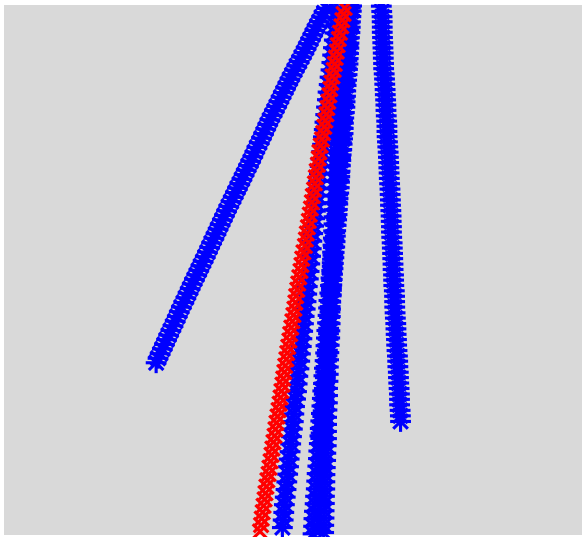


(b) Close-up view of (a)

Figure 4.5: Original Trajectory Map for the maneuvering target. The blue points indicate the training samples, and the red points indicate the test samples.



(a)



(b) Close-up view of (a)

Figure 4.6: Original Trajectory Map for the ballistic target. The blue points indicate the training samples, and the red points indicate the test samples.

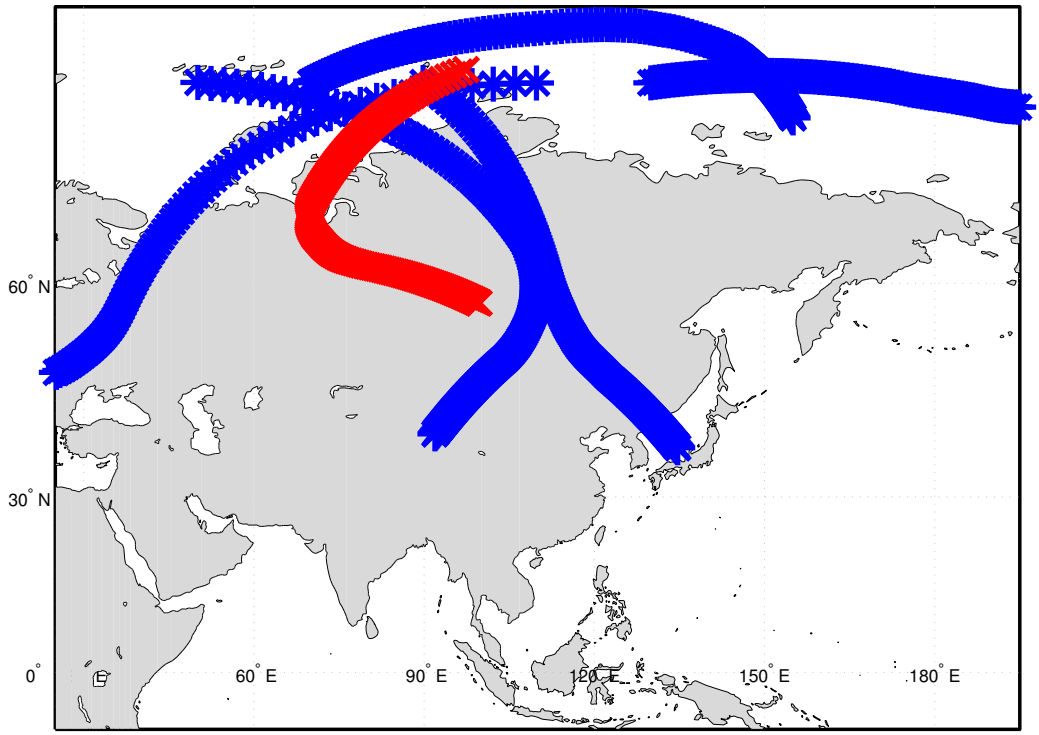


Figure 4.7: Updated Trajectory Map for the maneuvering target. The blue points indicate the training samples, and the red points indicate the test samples.

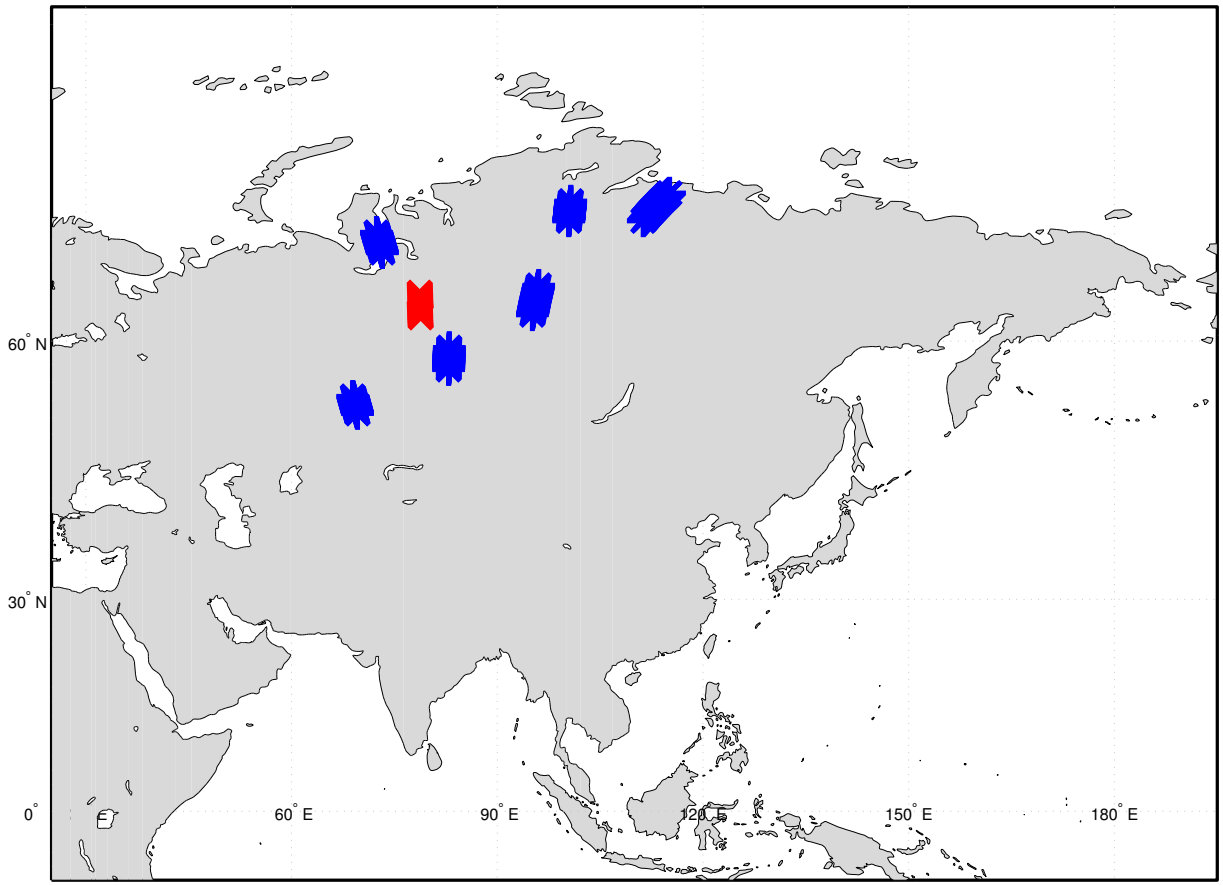


Figure 4.8: Updated Trajectory Map for the ballistic target. The blue points indicate the training samples, and the red points indicate the test samples.

4.2.1 Simulations (Different Training/Testing Sets)

Example 1: Maneuvering Target

For the maneuvering target, we increased the standard deviation of the initial training position to 2000m (previously 100m), and the standard deviation of the initial training velocity to 200m (previously 5m/s).

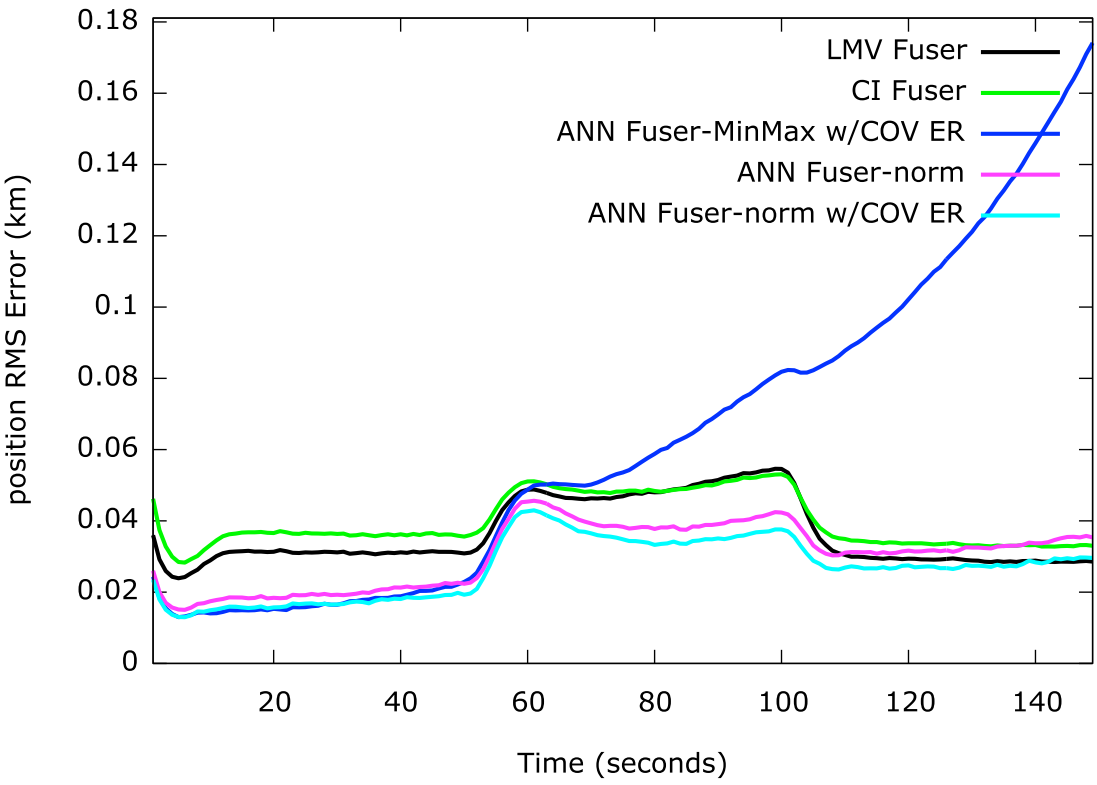
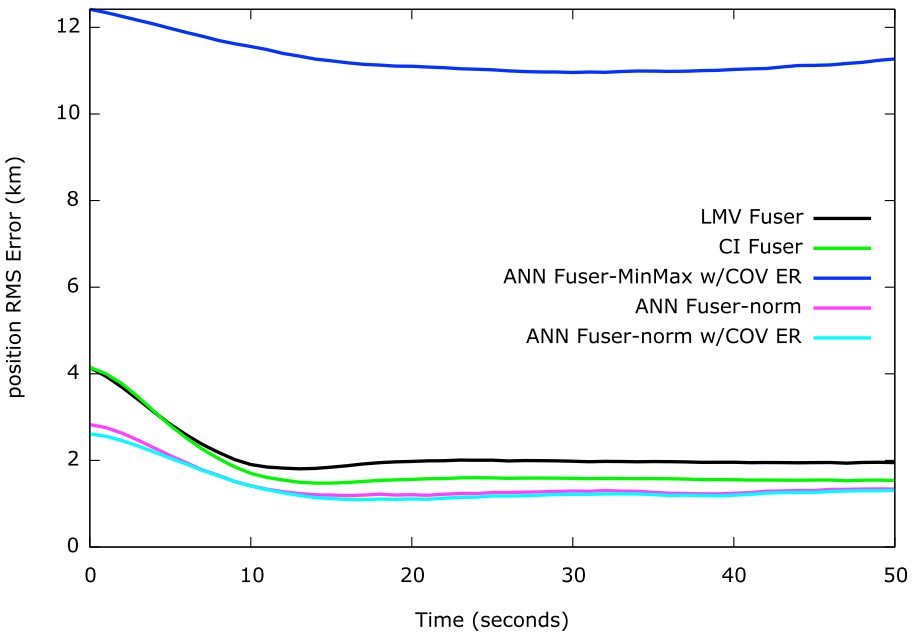


Figure 4.9: The LMV, CI, and ANN fusers using 6 training trajectories at locations similar to that shown in Figure 4.7. The blue line in (a) is the resulting error from utilizing the Min-Max Normalization scheme and the proposed error regularization, the magenta line, called "ANN Fuser-norm", is the resulting error from utilizing the new normalization scheme, and the cyan line labeled "ANN Fuser -norm w/COV ER" line is the resulting error from utilizing the new normalization scheme with the proposed error regularization.

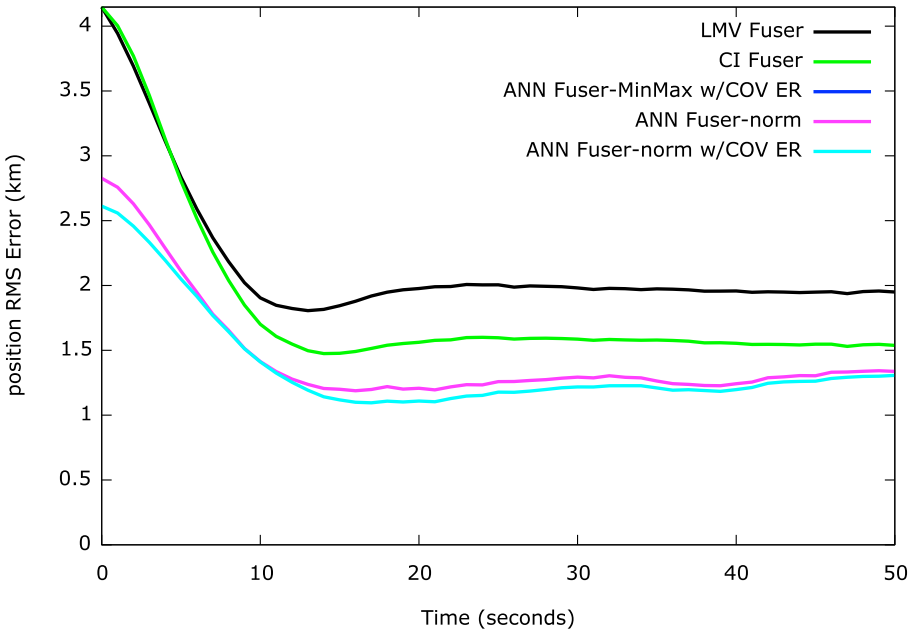
Example 2: Ballistic Coast Target

For the ballistic target, the standard deviation of the initial training position was increased to 500km (previously 0.1km), and the standard deviation of the initial training velocity was increased to 1km/s (previously 0.01km/s). Simulations were run, testing 3 variants of the ANN fuser: 1) "ANN Fuser-MinMax w/Cov ER", which is the ANN Fuser with the Min-Max Normalization scheme utilizing the proposed error regularization (with the error covariance estimates), 2) "ANN Fuser-norm", which is the ANN Fuser with the new normalization scheme, and 3) "ANN Fuser-norm w/COV ER", which is the ANN Fuser combining the new normalization scheme and the proposed error regularization. The resulting RMS error in the

target's position in shown in Figure 4.10. Even with the proposed error regularization, with the Min-Max normalization scheme the error is quite high (nearly three times that of the linear fusers), but the with the alternate normalization scheme described in this section, the ANN fusers are again able to outperform the linear fusers.



(a)



(b) Close-up view of (a)

Figure 4.10: The LMV, CI, and ANN fusers using 6 training trajectories at locations similar to that shown in Figure 4.8. The blue line in (a) is the resulting error from utilizing the Min-Max Normalization scheme and the proposed error regularization, the magenta line (seen more clearly in (b)), called "ANN Fuser-norm", is the resulting error from utilizing the new normalization scheme, and the cyan line (also seen more clearly in (b)) labeled "ANN Fuser -norm w/COV ER" line is the resulting error from utilizing the new normalization scheme with the proposed error regularization.

In addition, it can be seen in Figure 4.10 that the proposed error regularization is able to provide only a slightly reduced error over the baseline ANN fuser; the impact is not as significant as shown when using Min-Max normalization. If we reduce the number of training trajectories to 4 (shown in Figure 4.11), we still do not see as much of a benefit in using the proposed error regularization with the $\mathcal{L}_2$-normalization scheme as we did with the Min-Max normalization. Overall, it is still recommended that the proposed error regularization scheme is utilized regardless of the normalization scheme used since it has been shown to yield the lowest errors of all the schemes presented thus far and does not add any additional computational cost during testing.

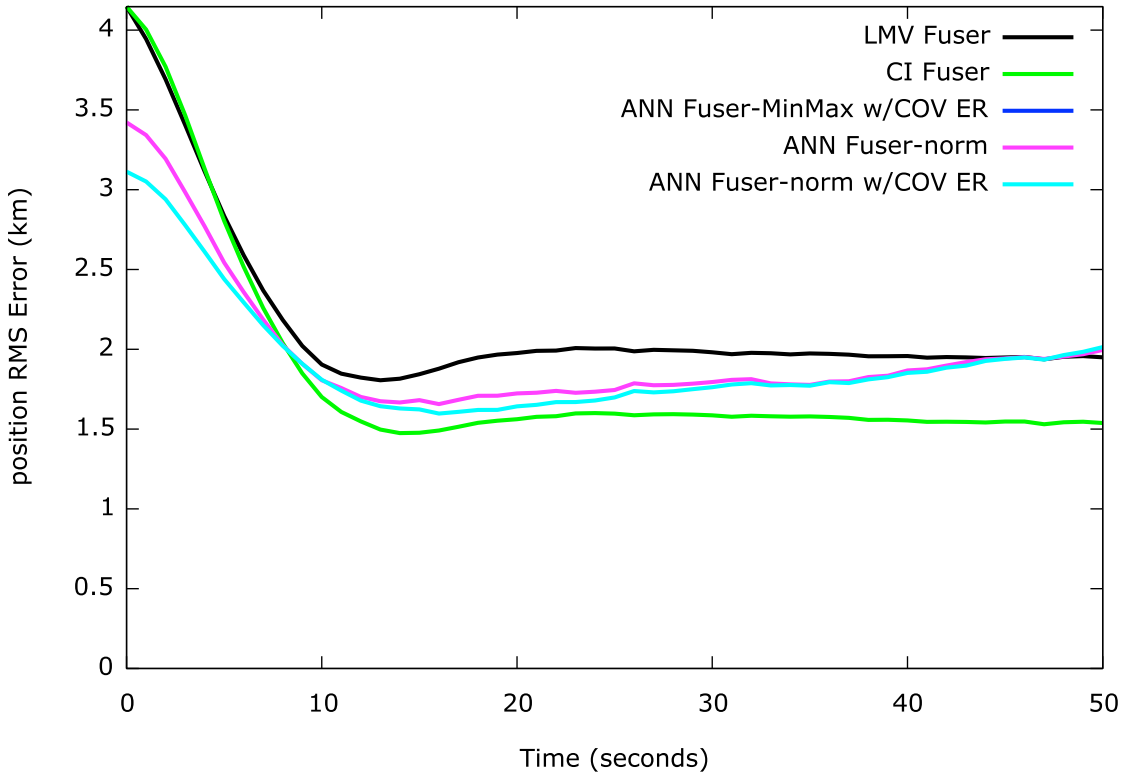


Figure 4.11: The LMV, CI, and ANN fusers using 4 training trajectories

4.3 Multiple Fusers

Since there are essentially two different modes for the maneuvering target, it may make sense to utilize different fusers based on whether or not the target is performing a maneuver (as determined by evaluating whether or not the estimated turn rate is above a certain threshold, e.g., 0.005°/s, from the sensors). For the ballistic target, since there is only one basic mode, we tested the utilization of different fusers depending on the region in which the target is located.

4.3.1 Simulations (Multiple Fusers)

For the maneuvering target, two separate ANN fusers were trained. One ANN fuser was trained only using data collected from when the target was traveling in a straight line (no maneuver), and another ANN fuser was trained only using data that was collected during a maneuver (i.e., when the turn rate is greater than zero). Then during testing, if the estimated average turn rate (as estimated by the sensors) was above a certain threshold ($\geq 0.005°$/s), the maneuver-specific ANN fuser was used to fuse the data, and if the estimated average turn rate was below that threshold (indicating that it was likely that a maneuver was not being performed), the non-maneuver-specific ANN fuser was used to fuse the testing data.

Figure 4.12 shows the position RMS error if a single ANN fuser is used compared to using multiple fusers – a maneuver-specific and non-maneuver-specific ANN fuser used separately, depending on the estimated target state. From Figure 4.12, it can be seen that using multiple fusers for the different motions (maneuvering and traveling straight) helped reduce the overall error, thus suggesting that it may make more sense to train different fusers if there are differing target motions. Note that during testing, the threshold for determining which ANN fuser to use was strictly set; if the estimated average turn rate was $\geq 0.005°$/s, then the maneuver-specific ANN fuser was used, otherwise, the non-maneuver-specific ANN fuser was used.

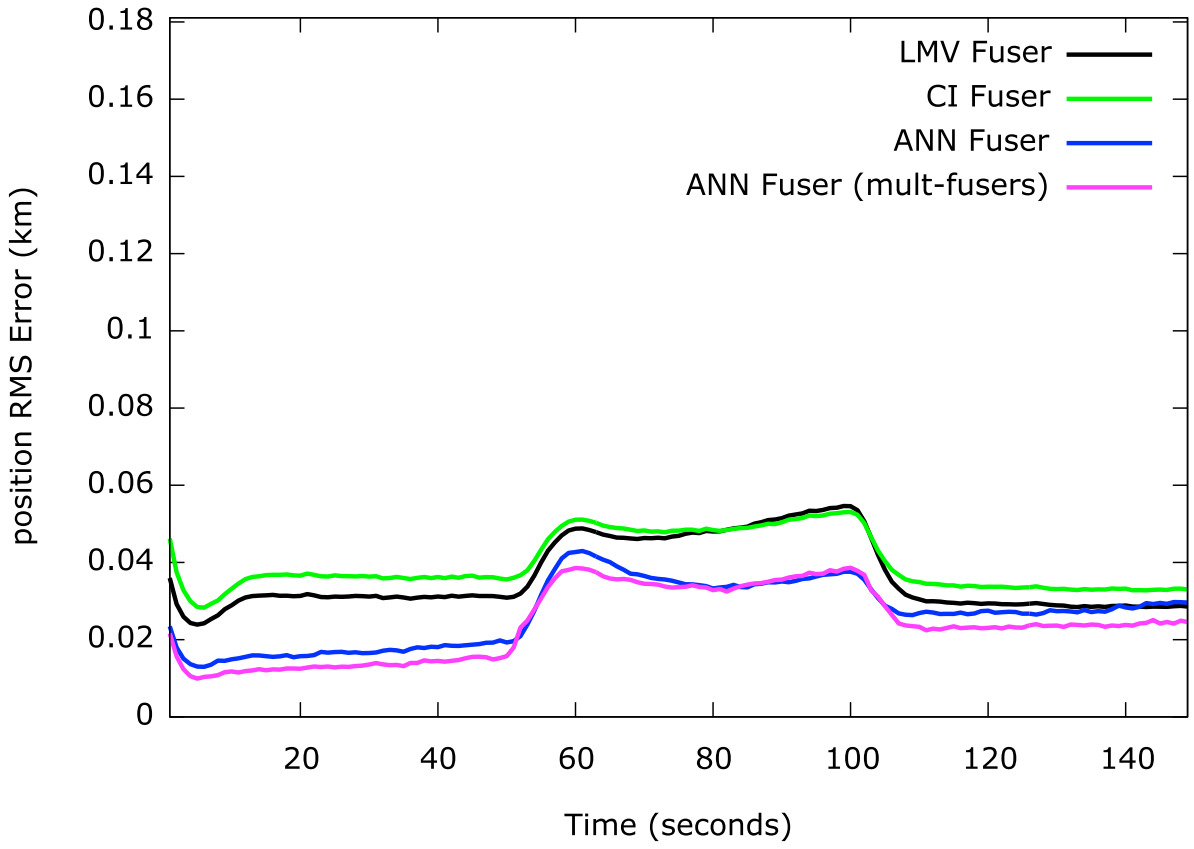


Figure 4.12: RMS error for the ANN fuser with the maneuvering target, comparing the utilization of a single fuser versus two fusers (one for each mode of the target).

For the ballistic target, a different approach was used since there is only one motion model assumed here. Instead of training motion-model-specific fusers, we trained fusers based on location. We first divided the 3D training space (for position only) into eight cubic regions (by dividing the x-y-z space into two regions for each coordinate). We then trained an individual ANN fuser for each region, using the training data that was found in that region. When testing, the test state estimates were used to determine the appropriate region that the target was located in (at time k), and then the fuser for that region was used to fuse that data. Figure 4.13 shows the resulting position RMS error for the ballistic target for a single fuser and for multiple fusers. In this example, 50 training trajectories were used in order to increase the amount of data that may be found within each region.

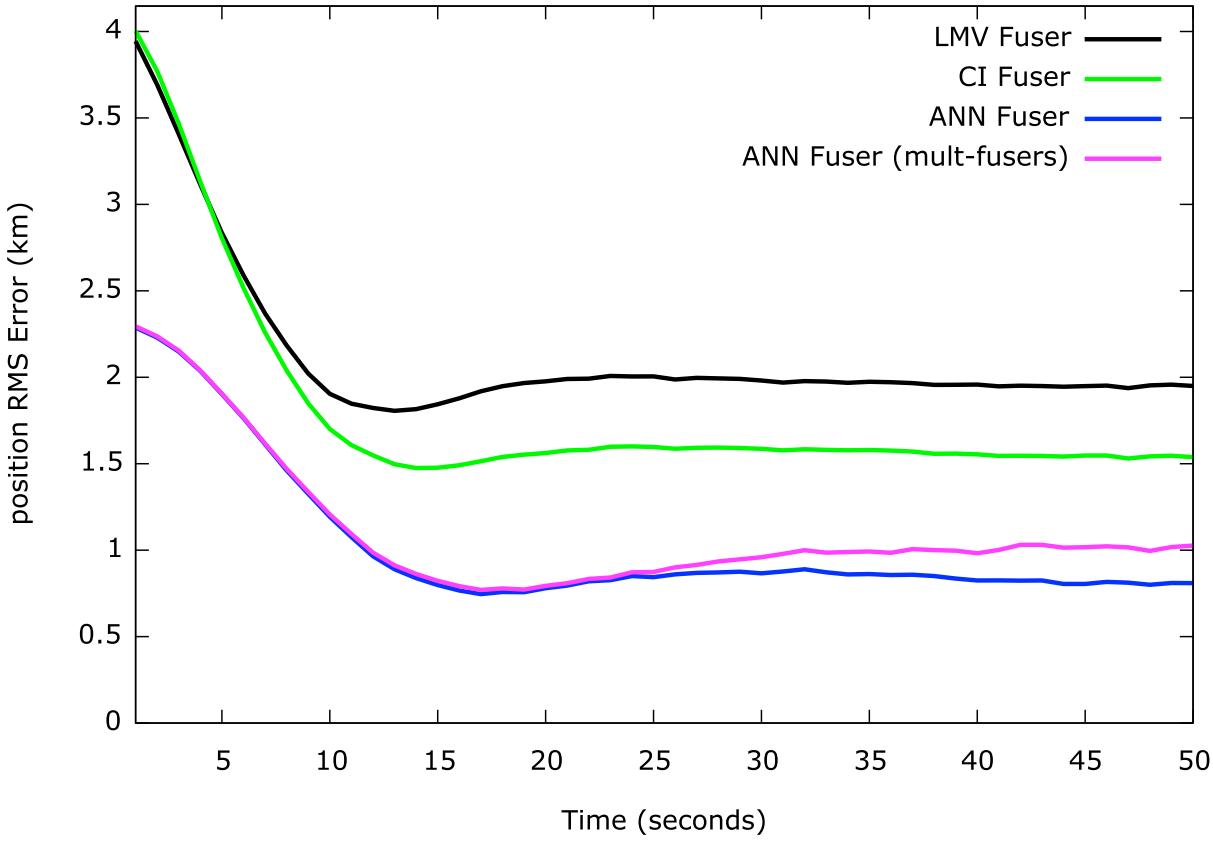


Figure 4.13: RMS error for the ANN fuser with the ballistic target, comparing the utilization of a single fuser versus multiple location-dependent fusers with 50 training trajectories.

For the ballistic target, even though 50 training trajectories are used to train the ANN fusers, by necessity the use of multiple fusers reduces the overall amount of training data available for each individual fuser. As can be seen in Figure 4.13, the single fuser (blue line) outperforms the set of multiple fusers (magenta line), thus indicating that more training data may be more beneficial to the ANN fuser than being within in a closer proximity to the target. This may be also due to the normalization scheme presented in Section 4.2, where each data point is normalized individually, regardless of where other data points are located (cf. Min-Max normalization), so the need to train based on location is diminished. In Figure 4.13, we can also see that the error for multiple fusers and the single fuser seem to coincide until it reaches the end of the trajectory. If we look at the average number of trajectories used to train the ANN fuser used for each point in the test trajectory (shown in Figure

4.14), we can see that towards the end of the testing trajectory, there are fewer training data available in that region for the ANN fuser that was used. Therefore, the error begins to increase due to there being less training data available. From these simulations we may conclude that if 50 training trajectories are indeed available, it may be more constructive to utilize all of the training data for training a single fuser versus dividing up the training data for multiple fusers.

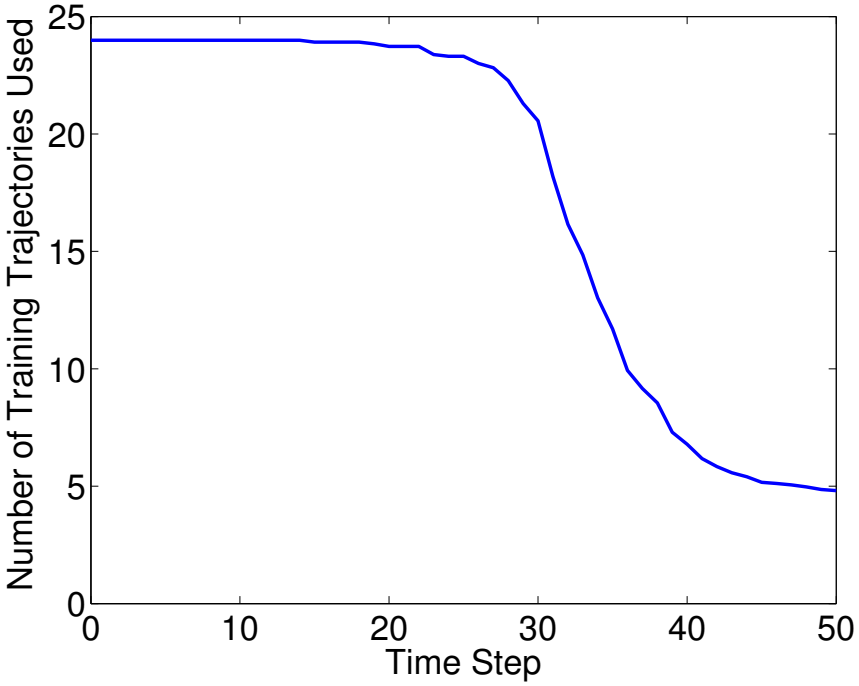


Figure 4.14: Ballistic Target: Average number of training trajectories used to train the ANN fuser used at a particular time step for the testing trajectory.

Also, similarly to the maneuvering target, the thresholds for determining which ANN fuser to use were also strictly set; the boundaries of each region did not overlap.

From this brief study, it appears that utilizing more than one fuser depending on the mode of the target (i.e., for different motion models) may be more beneficial, while training based on location (for our setup utilizing the normalization scheme presented in Section 4.2) does not appear to be of much assistance in reducing the error. It is also noted that the use of fuzzy (instead of strict) thresholds for determining which ANN fusers to employ may

provide better results during transition periods, which may be of interest in future work.

4.4 Input Features

In the previous simulations, the fused estimate at time k was computed using only the state estimates at time k as the fuser inputs. As an example, the input vector (up until this point) for the maneuvering target for two sensors has been:

$$\hat{\mathbf{x}}_k = \begin{bmatrix} \hat{\xi}_k^{(1)} & \hat{\eta}_k^{(1)} & \hat{\xi}_k^{(2)} & \hat{\eta}_k^{(2)} \end{bmatrix}^T \tag{4.4.1}$$

where $\hat{\xi}_k^{(i)}$ is the state estimate for coordinate ξ at time k for sensor i, and similarly $\hat{\eta}_k^{(i)}$ is the state estimate for coordinate η at time k for sensor i.

In this section we investigate whether or not it would be beneficial to incorporate additional information such as previous state estimates (e.g., at time $k-1$, $k-2$, etc.) to produce the fused estimates at time k. According to our assumed target model (Eq. (2.1.1)), the current state of the target is a function of the previous state plus noise. Therefore, we may benefit from utilizing previous state estimates as inputs into the ANN fuser to potentially furnish the ANN with additional information regarding the current state. Two separate experiments were run using the previous state information in the following way:

(1) Appended all of the previous states (e.g., the previous position and velocities)

(2) Appended a function of the previous states (e.g., the previous position plus the velocities)

For experiments (1) and (2), for the given example above (the maneuvering target with two

sensors), the input vector $\hat{\mathbf{x}}_k$ was appended with:

$$\mathbf{a}_{k,(1)} = \begin{bmatrix} \hat{\xi}^{(1)}_{k-1} \\ \hat{\dot{\xi}}^{(1)}_{k-1} \\ \hat{\eta}^{(1)}_{k-1} \\ \hat{\dot{\eta}}^{(1)}_{k-1} \\ \hat{\xi}^{(2)}_{k-1} \\ \hat{\dot{\xi}}^{(2)}_{k-1} \\ \hat{\eta}^{(2)}_{k-1} \\ \hat{\dot{\eta}}^{(2)}_{k-1} \end{bmatrix}, \qquad \mathbf{a}_{k,(2)} = \begin{bmatrix} \hat{\xi}^{(1)}_{k-1} + T \cdot \hat{\dot{\xi}}^{(1)}_{k-1} \\ \hat{\eta}^{(1)}_{k-1} + T \cdot \hat{\dot{\eta}}^{(1)}_{k-1} \\ \hat{\xi}^{(2)}_{k-1} + T \cdot \hat{\dot{\xi}}^{(2)}_{k-1} \\ \hat{\eta}^{(2}_{k-1} + T \cdot \hat{\dot{\eta}}^{(2)}_{k-1} \end{bmatrix} \tag{4.4.2}$$

for experiments (1) and (2), respectively, where $\mathbf{a}_{k,(j)}$ represents the appended vector at time k for experiment j, and T is the time step from time $k-1$ to time k. Therefore, the new input vectors become:

$$\hat{\mathbf{x}}_{k,(1)} = \begin{bmatrix} \hat{\mathbf{x}}_k \\ \mathbf{a}_{k,(1)} \end{bmatrix}, \qquad \hat{\mathbf{x}}_{k,(2)} = \begin{bmatrix} \hat{\mathbf{x}}_k \\ \mathbf{a}_{k,(2)} \end{bmatrix}, \tag{4.4.3}$$

again, for experiments (1) and (2), respectively.

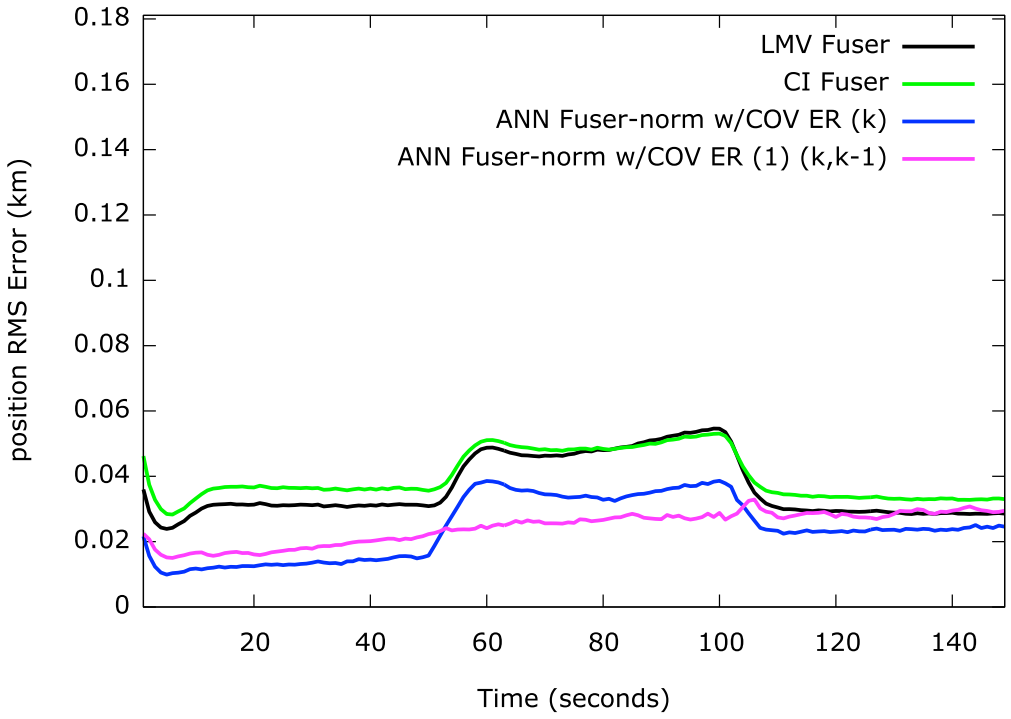
4.4.1 Simulations (Input Features)

Figure 4.15 shows the RMS error for the maneuvering target for Experiments (1) and (2), separately. As can be seen in Figure 4.15, in appending the input vector with a *function* of the previous states (i.e., the estimates of the current state based on previous states) as was done in Experiment (2), the performance can be slightly improved, while the Experiment (1) results in Figure 4.15a showed no improvement. In addition, in Figure 4.15b (Experiment (2)), it appears that using state estimates from two previous time steps (times $k-1$ and $k-2$) yields the best performance. Therefore, simulations will henceforth include appending

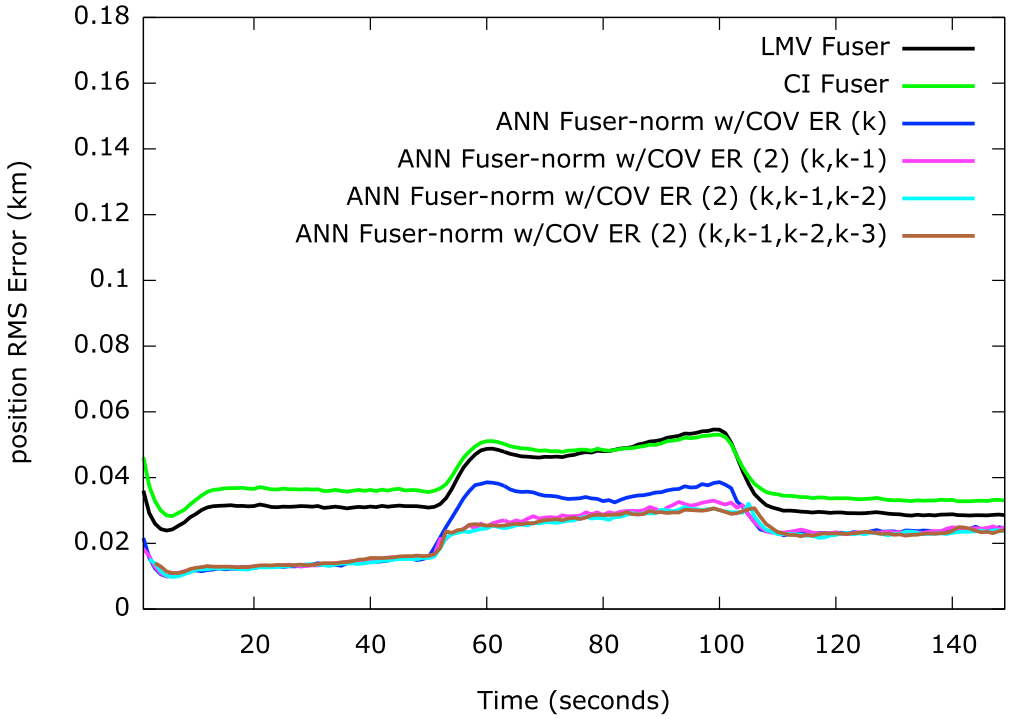
two previous state estimates to the input vector for the ANN fuser for the maneuvering target.

For the ballistic target, Figure 4.16 shows the error from using information for more than one time instant. It can be seen in Figure 4.16a, Experiment (1) performs particularly poorly for the ballistic target using Min-Max normalization. Experiment (1) uses the previous positions and velocities as additional inputs to the ANN fuser when computing the state at the current time. It is hypothesized that this is due to the previous velocities themselves being poor indicators of the current position, and when Min-Max normalization is used, the velocities are scaled to be just as "important" (i.e., have large values) as the previous and current positions in computing the current state, thus resulting in higher errors.

Interestingly enough, however, if utilizing the Min-Max normalization scheme, then using the current and previous time estimates from Experiment (2) indeed reduces the error (also shown in Figure 4.16a), but if utilizing the proposed normalization scheme (from Section 4.2), then using only the current state estimate results in the best performance (shown in Figure 4.16b). In Experiment (2), the previous velocity estimates are multiplied by the time step and are added to previous position estimates to predict the current state. However, perhaps this is a poor way to predict the current state for the ballistic target. The Min-Max normalization scheme normalizes the inputs so that each input variable is scaled to within the same range, which may somewhat correct the poor prediction. Alternatively, the proposed normalization scheme would preserve the relative differences between the inputs and would thus retain the poor predictions. It is conjectured that this may be what is contributing to this difference in behavior for the ballistic target when using different normalization schemes.

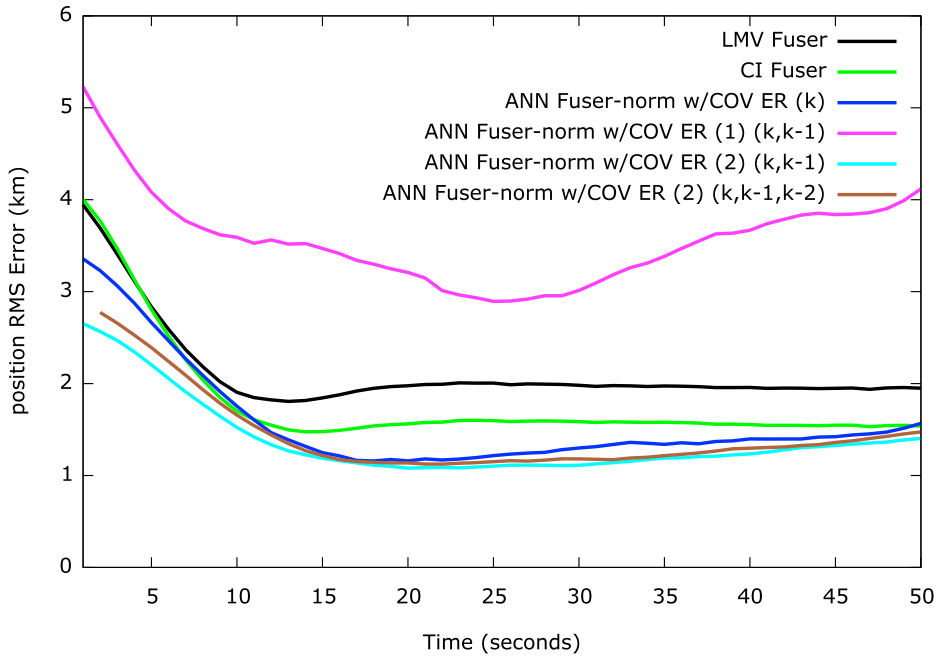


(a) Experiment (1): Using information from time k (the baseline), and time k and $k-1$.

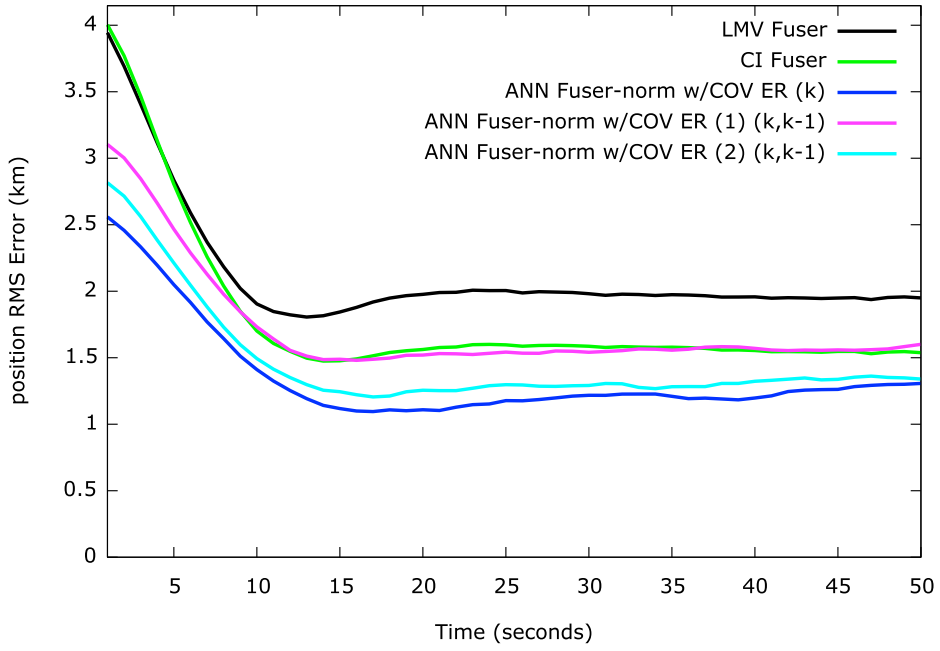


(b) Experiment (2): Using information from time k (the baseline), time k and $k-1$, times $(k, k-1, k-2)$, and times $(k, k-1, k-2, k-3)$.

Figure 4.15: ANN Fuser Performance for the Maneuvering Target using state estimates from more than one time step.



(a) Position RMS Error for the ANN Fuser using information from time k, time k and $k-1$, and time k, $k-1$, and $k-2$ with experiments (1) and (2), with the Min-Max normalization scheme.



(b) Position RMS Error for the ANN Fuser using information from time k, and time k and $k-1$ with experiments (1) and (2), with the proposed normalization scheme.

Figure 4.16: ANN Fuser Performance for the Ballistic Target using state estimates from more than one time step.

4.5 ANN Architecture Parameters

In this section, we will examine how the given architecture of the ANN fuser affects the performance. We can change the architecture of the ANN fuser by increasing or decreasing the number of hidden nodes and by increasing the number of hidden layers. For the number of hidden nodes, thus far the default number of hidden nodes used has been 20. In this section we show the average position RMS error when varying the number of hidden nodes and layers used in the ANN fuser.

4.5.1 Simulations (Hidden Layers)

In the past, training deep neural networks (i.e., neural networks with more than one hidden layer) was known to be hard and often yielded poor solutions when applying the standard learning strategy of randomly initializing the weights and applying gradient descent using backpropagation [68]. However, advances in neural networks within the last decade [68, 69] have shown that utilizing a proper initialization strategy for the weights of a deep ANN (e.g., a greedy layer-wise unsupervised training strategy) can yield better results than with a shallow ANN. For this set of experiments, we will therefore pre-train each layer by initially treating each hidden layer separately, setting the outputs of each hidden layer to the final target values. So for an ANN with two hidden layers, we use the following procedure:

1. Train the set of weights for the first hidden layer *only* by creating a temporary output, where this temporary output is set to the final target values (the true states). Compute the set of weights for the first hidden layer with the same backpropagation algorithm used for training an ANN with one hidden layer using these temporary outputs.

2. Discard the temporary outputs, and compute the actual outputs of the first hidden layer.

3. Use the actual outputs of the first hidden layer as the inputs into the second hidden layer. Train the weights for the second hidden layer by treating the network now as

an ANN with one hidden layer, where the inputs are the outputs of the first hidden layer, and the outputs are the true states.

4. The weights are now "initialized". Train the entire network now with both hidden layers using backpropagation.

Figures 4.17 and 4.18 show the results for the maneuvering and ballistic target, respectively. It can be seen that the addition of another layer, pretrained in the manner described, does not reduce the error. However, this should not preclude the use of investigating deep neural networks for use in multisensor fusion for target tracking. It is possible that alternate pretraining schemes may assist in reducing the overall error for a multi-layered ANN fuser.

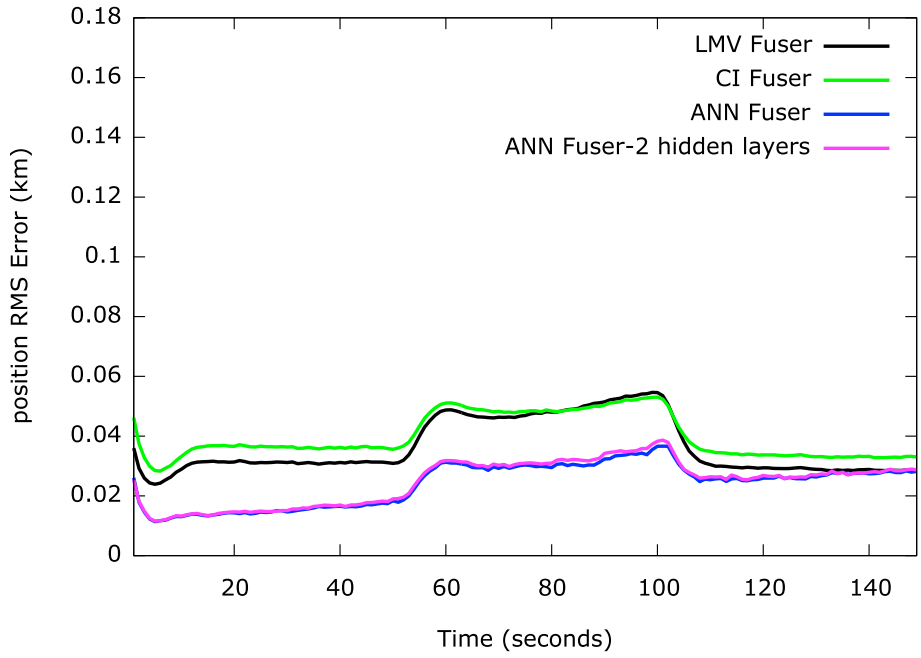


Figure 4.17: Maneuvering Target: average position RMS error for the ANN fuser using two hidden layers, with 10 nodes per layer.

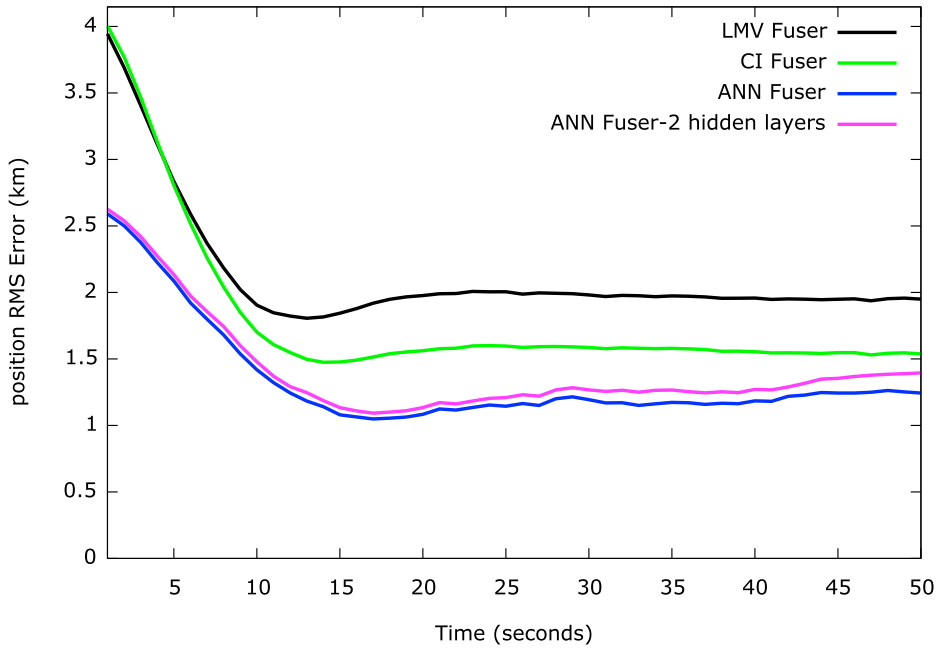


Figure 4.18: Ballistic Target: average position RMS error for the ANN fuser using two hidden layers, with 10 nodes per layer.

4.5.2 Simulations (Number of Hidden Nodes)

In varying the number of hidden nodes for these simulations, the number of hidden nodes does not appear to have a significant effect on the RMS error, so the average RMS error (averaged across time) is plotted instead. We found that for the ballistic target, the number of hidden nodes needed to achieve the lowest average RMS error in the position is greater (at 60 hidden nodes) than that needed for the maneuvering target (at 30 hidden nodes – likely due to the increased number of inputs needed). The ballistic target has an additional coordinate (and therefore, more input data) and also always follows a nonlinear trajectory, thus perhaps requiring the use of more hidden nodes. The number of hidden nodes for the maneuvering target in these simulations is constant regardless of whether it is performing a maneuver or traveling a straight course.

Example 1: Maneuvering Target

The average RMS error for the maneuvering target is shown in Figure 4.19. Based on these results, simulations henceforth will utilize 30 hidden nodes in the ANN fuser.

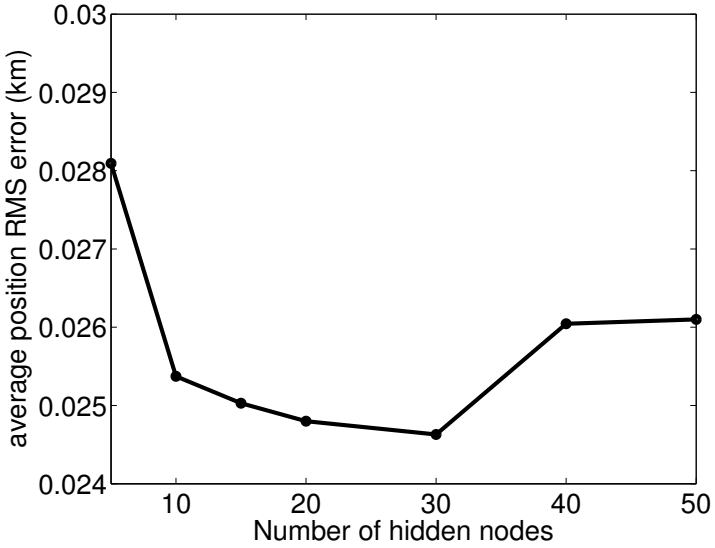


Figure 4.19: Average position RMS error for the ANN fuser with the maneuvering target, varying number of hidden nodes from 5 to 50.

Example 2: Ballistic Coast Target

The average RMS error for the ballistic target is shown in Figure 4.20. Based on these results, simulations henceforth will utilize 60 hidden nodes in the ANN fuser.

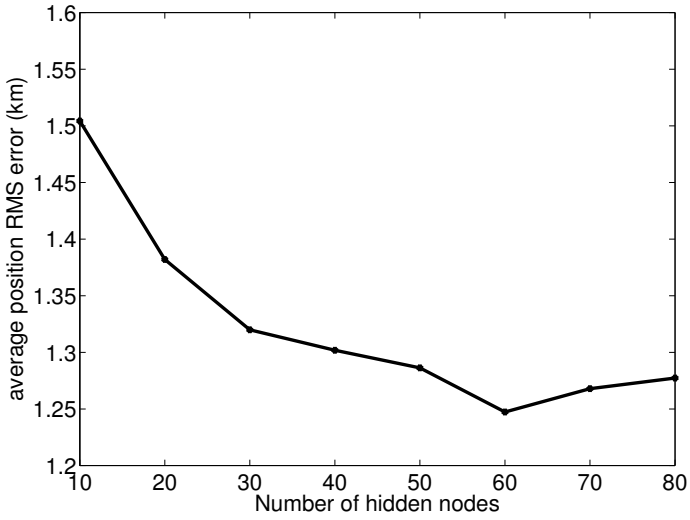


Figure 4.20: Average position RMS error for the ANN fuser with the ballistic target, varying number of hidden nodes from 10 to 80 in increments of 10.

4.6 Summary

Throughout this chapter, we explored two main topics for the ANN fuser: 1) Performance, and 2) Practicality. First, a new regularization scheme for the ANN fuser was presented, which exploited additional information that we have about the inputs: the error covariance estimates. Using the proposed regularization during the training of the ANN fuser allowed for fewer training data to be used while still maintaining accuracy, but still required the training data to be close to the testing data. Therefore, an alternate data normalization method is suggested for use with the ANN fuser, which was shown in this chapter to broaden the allowable training space while still obtaining error levels less than that of linear fusers. Several other studies were conducted, yielding the following conclusions:

- It may be more beneficial to utilize multiple ANN fusers if the target can undergo multiple motions.
- It is not necessary to break up the training data into regions to train location-dependent ANN fusers.

- It can also help to use information from previous time steps.
- The number of hidden nodes can be varied to further reduce the ANN fuser error.
- However, initial investigations in increasing the number of hidden layers did not show improvement.

Chapter 5

Effects of Imperfect Communications

Due to the long-haul sensor network, however, the fusion center may not receive all of the state estimates in time from the sensors as some packets may be lost and/or delayed. Given that there is only a finite time window in which we can wait for delayed packets due to reporting requirements, packets that are delayed beyond a certain threshold are effectively considered to be lost packets. The ANN fuser is designed in such a way that all of the inputs/state estimates are required in order for the ANN fuser to function as designed. However, in target tracking, there is typically some prior assumed knowledge of the target's dynamic system, so one can use this information to predict a future state from a previous state. So if a state estimate is missing, a replacement state estimate can be easily imputed from previously received state estimates through prediction using an assumed target motion model, and the error covariance additionally becomes inflated as a result. This predicted state estimate is then used in place of the missing state estimate. We can likewise use the predicted state in place of the missing input to the ANN fuser, but the ANN fusers are trained with data that have a certain distribution, and the distribution of the predicted state is different (i.e., has a higher error covariance) than that of the true state estimate, which may potentially result in poorer fuser performance. We investigate here the effects of packet losses/data replacements on the ANN fuser.

5.1 Experimental Methods

In all of these experiments, any missing data will be imputed from the most recently received state estimate. Experiments are run assuming that at least the first packet from all sensors is received so that we may actually impute subsequent missing data. Several methods for dealing with the missing data were tested for dealing with packet losses, and the following training approaches will be used as the baseline comparison:

- Train the ANN fuser on all of the available data, and test on incomplete data.
- Train different fusers with different combinations of missing data (where if the data is missing, it is replaced by imputed data using the previous state estimate). For example, if there are two sensors, a different fuser is trained for the various cases of missing sensor data: 1) Missing Sensor 1 Data, 2) Missing Sensor 2 Data, and 3) Missing Sensors 1 and 2. We shall assume that we eventually receive the missing data so that future data can be imputed.
- Augment the training data with duplications of the original training dataset with different random missing state estimates.

The aforementioned 'baseline' methods will be compared to the following proposed methods:

- Introduce an additional input to represent a "confidence" that we have in the inputs (i.e., the state estimates). For example, if the input is missing then we may say that we have zero to little confidence in this input. The error covariance estimates will be used as a measure of confidence.
- Train the neural network in such a way so it is somewhat "insensitive" or robust to changes in the input. This method will be detailed in the following section.

5.2 Artificial Neural Network (ANN) Sensitivity

The sensitivity of an ANN's output to perturbations in its input has been studied extensively in the past as one such method for assessing a neural network's performance or generalization ability. Sensitivity analysis in neural networks has been used for a number of different purposes primarily related to assessing the significance of model inputs, which is useful for applications such as selective learning [70], pruning the inputs [71, 72], and weight selection [73], just to name a few. Here, we will look into the use of the neural network's sensitivity measure in an effort to produce neural networks that are relatively robust against changes to the input by directly minimizing the sensitivity of the network.

Preliminary experiments suggest that solely minimizing the network sensitivity yields very large errors. This is likely due to the sensitivity and the network error being completely unrelated so that minimizing the network sensitivity does nothing to reduce the error between the neural network output and the ground truth. Therefore, two different approaches will be tested to see if using the sensitivity will help with reducing errors due to packet losses. Approach (1) will be to initialize the network by first minimizing the sensitivity and then subsequently minimizing the error, and Approach (2) will be to attempt to simultaneously minimize the sensitivity and error by adding the sensitivity measure to the objective function similarly to the regularization approach presented in Section 4.1.1. To reiterate (for convenience), the error function in regularization typically takes the form

$$\tilde{S}(\mathbf{w}) = S(\mathbf{w}) + \lambda\Omega, \tag{5.2.1}$$

where $S(\mathbf{w})$ is the sum of the squared errors, and recall that the parameter λ controls the degree to which the penalty term Ω influences the form of the solution [59]. We will let the penalty term in Eq. (5.2.1) be the squared sensitivity for Approach (2), or we shall minimize the squared sensitivity prior to minimizing the squared error for Approach (1).

Now, let us define the term 'sensitivity' as the measure of the change in an output given

a change in an input, or, in mathematical terms, we can define this as the partial derivative of an output, $\hat{x}_F^{(m)}$, over the partial derivative of an input, $\hat{x}^{(n)}$ as follows [71]:

$$V_n^m = \frac{\partial \hat{x}_F^{(m)}}{\partial \hat{x}^{(n)}}. \tag{5.2.2}$$

This sensitivity can then be expressed in terms of the network parameters as

$$\begin{aligned} \frac{\partial \hat{x}_F^{(m)}}{\partial \hat{x}^{(n)}} &= \sum_{j=1}^{L} w_{mj}^o \frac{\partial a_j}{\partial \hat{x}^{(n)}} \\ &= \sum_{j=1}^{L} w_{mj}^o a_j' w_{nj} \end{aligned} \tag{5.2.3}$$

where a_j is the output of the j^{th} hidden node (out of L total hidden nodes), and w_{mj}^o is the neural network weight that is multiplied by the j^{th} hidden node for the m^{th} output, and w_{nj} is the neural network weight that is multiplied by the n^{th} input for the j^{th} hidden node.

We can derive now a matrix expression for the network sensitivity. Following the notation given in Section 3.3.4 where the ANN fuser is first described, W_H is the matrix of weights that are multiplied by the inputs. Let W_o represent the weight matrix that is multiplied by the outputs from the hidden layer. If $\mathbf{a}$ is our vector of hidden node outputs, then let A be a diagonal matrix with $\mathbf{a}$ along the diagonal. If $\hat{\mathbf{x}}_F$ is our vector of outputs, then let X_F be a diagonal matrix with $\hat{\mathbf{x}}_F$ along the diagonal. The network sensitivities can then be written in matrix form as:

$$V = W_o^T A' W_H^T \tag{5.2.4}$$

where the prime ($'$) indicates the derivative with respect to a given input.

5.2.1 Minimizing the Squared Sensitivity

To minimize the optimization function for Approach (1) or (2) which involves the squared sensitivity, we need the derivative of the squared sensitivity so we can compute the Jacobian

used for updating the weights. The squared sensitivity v is given by

$$v \triangleq Tr(VV^T) = Tr(W_o^T A' W_H^T W_H A' W_o). \tag{5.2.5}$$

We are interested in computing the partial derivative of v with respect to the neural network parameters. Let us first start with computing the partial derivative with respect to W_H. Let $U = W_H A'$, so we have

$$v \triangleq g(U) = Tr(W_o^T U^T U W_o). \tag{5.2.6}$$

From [74], we have the chain rule for matrix derivatives:

$$\frac{\partial g(U)}{\partial [W_H]_{ij}} = Tr\left[\left(\frac{\partial g(U)}{\partial U}\right)^T \frac{\partial U}{\partial [W_H]_{ij}}\right]. \tag{5.2.7}$$

And plugging Eq. (5.2.6) into Eq. (5.2.7), we have:

$$\begin{aligned}
\frac{\partial g(U)}{\partial [W_H]_{ij}} &= Tr\left[\left(\frac{\partial Tr(W_o^T U^T U W_o)}{\partial U}\right)^T \cdot \frac{\partial U}{\partial [W_H]_{ij}}\right] \\
&= Tr\left[\left(2 \cdot U W_o W_o^T\right)^T \cdot \frac{\partial U}{\partial [W_H]_{ij}}\right] \\
&= 2 \cdot Tr\left[W_2 W_2^T U^T \cdot \boxed{\frac{\partial U}{\partial [W_H]_{ij}}}\right].
\end{aligned}$$

A general formula was found for computing $\frac{\partial g(U)}{\partial [W_H]_{ij}}$ in the above expression:

$$\boxed{\frac{\partial g(U)}{\partial [W_H]_{ij}} = 2 \cdot \left[a_j \sum_{h_1=1}^{L} [W_{o2}]_{jh_1} a_{h_1} [W_H]_{ih_1} + a'_j \hat{x}_i \sum_{h_2=1}^{L} \left([W_{o2}]_{jh_2} a_{h_2} \sum_{c=1}^{N_i} [W_H]_{cj} [W_H]_{ch_2}\right)\right]} \tag{5.2.8}$$

where $W_{o2} = W_o W_o^T$, a_j is the output of the j^{th} hidden node, and if we have $a_j \triangleq f(net_j)$ where $f(\cdot)$ is the activation function and net_j is the input to the j^{th} hidden node, then $a'_j \triangleq f'(net_j)$. We can also compute a general formula for the partial derivative with respect

to the biases that are added to the hidden node inputs, $\frac{\partial g(U)}{\partial b_j}$:

$$\boxed{\frac{\partial g(U)}{\partial b_j} = 2a'_j \sum_{h=1}^{L} [W_{o2}]_{jh} \left(\sum_{c=1}^{N_i} [W_H]_{cj} [W_H]_{ch} \right)} \tag{5.2.9}$$

And lastly, since we are just considering one hidden layer here, we need to compute the partial derivative of the squared sensitivity with respect to W_o:

$$\frac{\partial g(U)}{\partial W_o} = \frac{\partial Tr(W_o^T U^T U W_o)}{\partial W_o} = 2U^T U W_o = \boxed{2A' W_H^T W_H A' W_o} \tag{5.2.10}$$

5.3 Simulations (Data Loss)

In these simulations, the TCP model presented in Section 2.4 is used to compute the loss probability of a network employing the TCP protocol. The testing data is created by discarding state estimates at the TCP loss rate and predicting missing values. Multiple simulations were run with fixed losses to better examine the effects of packet losses on the fuser performance. The variants of the ANN fuser that were run under the presence of packet losses are summarized in Table 5.1.

It can be seen in Figure 5.1 that the LMV and CI Fuser error lines exhibit peaks at the locations where there were packets that were lost/delayed (either single or consecutive packets). These peaks are clearly more pronounced when compared to that of the ANN Fuser-SENS-trnLoss (the red line lower in the plot), which is also plotted on top of the error curve that it would have had (ANN Fuser-SENS-trnLOSS (ALL DATA Rx)) if all the data had been received. It is apparent that for the sensitivity Approach (1) to achieve relatively low error, the training data should also include missing packets, otherwise its performance is on par of that as the ANN Fuser without any modifications. Overall, the sensitivity initialization procedure appears to help improve the overall performance with relatively minimal impact to the error when there are packet losses especially compared to that of the linear fusers.

Table 5.1: ANN variants used in packet loss testing.

ANN Variant	Description
ANN Fuser-CI	Uses a "confidence" input
ANN Fuser-trnLoss:	Augment the training data with duplications of the original training dataset with different random missing state estimates
ANN Fuser-SENS	(Sensitivity – Approach (1)) Initialize network by minimizing the squared sensitivity
ANN Fuser-MISS	Train different fusers with different combinations of missing data
ANN Fuser-SENS-LAMBDA	(Sensitivity – Approach (2))Incorporate a penalty term of the squared sensitivity into the optimization function
ANN Fuser-SENS-trnLoss	(Sensitivity – Approach (1)) Use the same input data as ANN Fuser-trnLoss

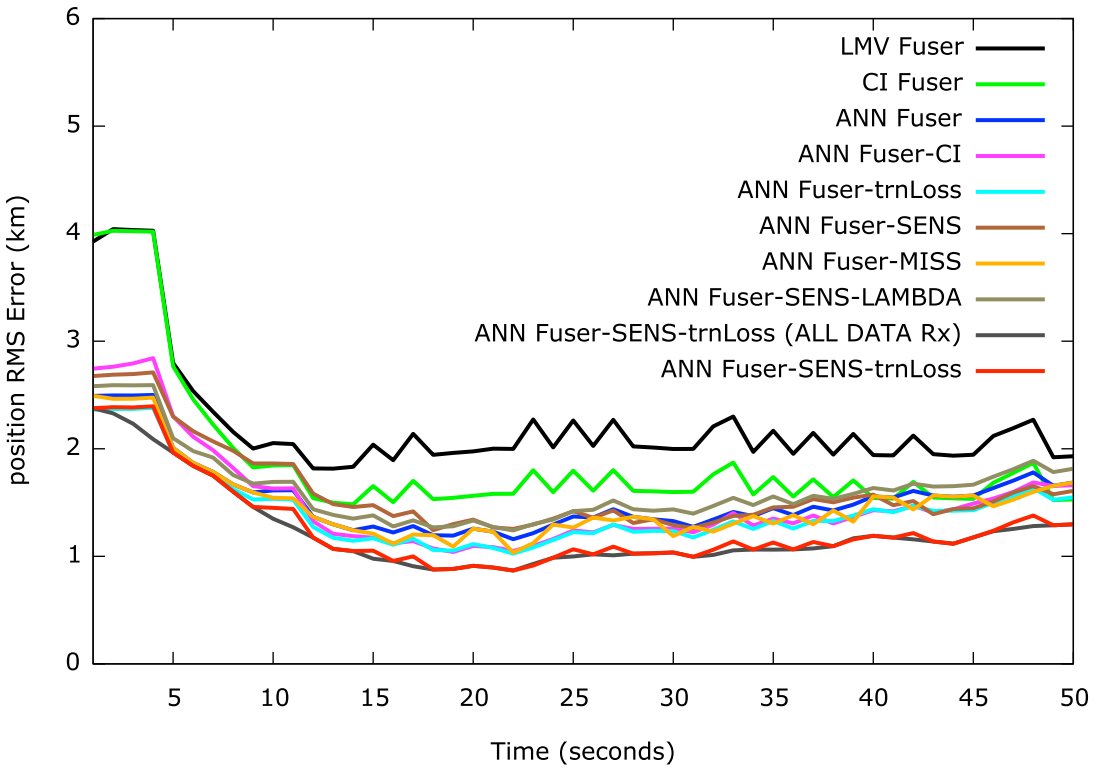


Figure 5.1: Position RMS error (km) for the linear and nonlinear fusers with packet losses for the ballistic target.

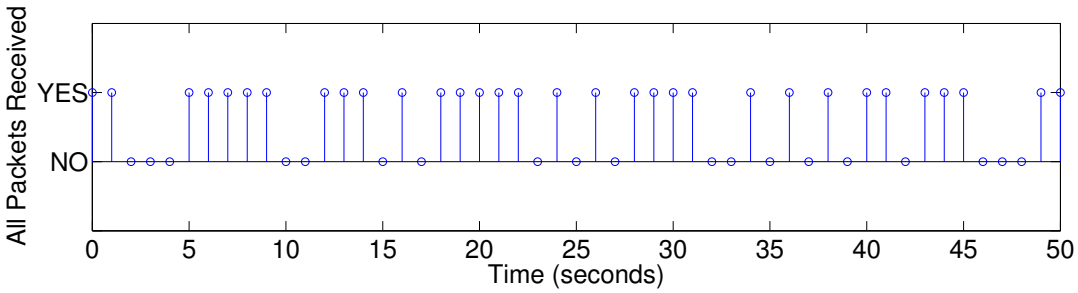


Figure 5.2: Time instants of packet losses for Figure 5.1.

Chapter 6

Discussion, Conclusions, and Future Work

Overall, this thesis is intended to be an initial investigation into the use of machine learning techniques for multisensor fusion in target tracking. Several existing machine learning approaches were investigated and extended for this particular application, and the real-world effects of utilizing long-haul networks for these approaches were also explored. Of all of the fusers that were investigated here, the ANN fuser was shown to have the best performance, and simulation results showed that further improvements could be made by modifying the conventional training procedures and the ANN architecture in order to enhance its performance and practicality.

6.1 Thesis Summary and Discussion

Chapter 2 provided the background material for implementing the target tracking system simulations. In Chapter 3, prior work in multisensor fusion was presented, where various fusers were run in a full system simulation of two different targets: a maneuvering and a ballistic target. Two popular linear fusers, the LMV and the CI fusers, were compared with four nonlinear learning-based fusers, and it was found that of all of the fusers explored

herein, the ANN fuser yielded the best results (in terms of the position RMS error) for both targets. Therefore, it was selected for further investigation into improving certain aspects of the fuser, namely its *performance* and its *practicality*.

In Chapter 4, a new method for training the ANN fuser was introduced utilizing the error covariance estimates, which heuristically appeared to provide better generalization to unseen data as fewer training data were required to obtain results better than that of the linear fusers (which were used throughout as the baseline comparison). The proposed method was compared with other existing regularization schemes and was found to outperform these existing methods. However, the training trajectories used still needed to be within fairly close vicinity to the testing trajectory. Therefore, an alternate data normalization scheme was also suggested for use with the ANN fuser, thus broadening the training space, allowing for the training and testing data to be farther apart in space yet still yield good results.

Several additional studies were conducted for enhancing the ANN fuser performance:

1. (**Multiple state fusers**). The use of multiple fusers to cover the state space was investigated, but it was found that (especially with the proposed alternate normalization scheme), it would be more prudent to utilize all of the training data to train a single fuser than to split up the training data to train multiple fusers. However, it was also found that multiple fusers would be beneficial for the case where there are multiple differing target motions (e.g., maneuver versus traveling in a straight line).

2. (**Utilizing more dynamic information**). Also under investigation in Chapter 4 was the utilization of more dynamic information as inputs into the fusers instead of simply the current time step, as was previously used by default. Several experiments were carried out for each target where the input features were the current state estimates in addition to either a function of the previous state estimates, or the previous state estimates themselves. It was found that using previous state estimates (projected to the current time step) actually assisted in further reducing the error for both targets, but the error increased the further back in time we went.

3. (**Varying the ANN architecture**). Lastly, the ANN architecture was briefly studied in varying the number of hidden nodes and layers of the feedforward network. The 'optimal' number of hidden nodes were different for each target (i.e., greater for the ballistic target), likely due to the additional coordinate that the ballistic target has over the maneuvering target in its position. It was also found that increasing the number of hidden layers did not seem to help reduce the error over a single hidden layer, but it is well known that deeper neural networks are notoriously difficult to train due to the increased number of parameters. It is possible that other pre-training schemes other than the one provided may yield better results.

In Chapter 5, we investigated the effects of packet losses on the ANN fuser. Since it is likely that we may have an assumed motion motion for the target, we can utilize that motion model to predict/impute the missing estimates. It was found that the ANN fuser already has reduced error for lost packets relative to the increase in error of the linear fusers. A new scheme was presented for further reducing the overall error for missing packets by first minimizing the sensitivity of the neural network and then minimizing the error. Simulation results showed improved overall performance.

6.1.1 Remarks on RMS Error

Throughout this thesis, the fusers have solely been judged on their relative RMS error to one another in the two different types of target simulations, but we would like to comment here on what may be considered an acceptable level of absolute RMS error. The absolute RMS error of the maneuvering target is actually on the same order as that in current target tracking literature (e.g., Yuan et al. (2011) [19] showed RMS errors in position ranging from about 25m$-$50m for a maneuvering target), but for the ballistic target, the absolute RMS errors appear to be quite large (ranging from 1km$-$4km).

It is difficult to compare the ballistic target errors with current work as much of the publicly published work studying ballistic targets is concerned with the boost or reentry

phases instead of the coast phase. Yeddanapudi et al. (1995) [17], however, do report RMS position errors for a coast ballistic target ranging from 0.1km−0.25km for sensors with angle measurement errors of 25μrad. (Note that the work by Yeddanapudi et al. in [17] is what is currently used to propagate the ballistic target trajectory in our simulations). If we use this as a point of comparison, our sensors (due to the distance of the sensor locations and the state-dependent measurement noise) have sensor angle measurement errors on the order of 350−600μrad. If the overall RMS position errors are proportional to the sensor angle measurement errors, then our level of sensor errors would extrapolate to RMS position errors of around 1.4km−6km, which is on the same order of the error seen in our simulation results.

However, it still remains that these levels of absolute RMS position errors for a ballistic target are likely unacceptable in a real-world scenario. It is worth noting then, that there are calculations used in our simulations that contribute to the high absolute RMS errors in position that may be better in a real-world scenario. For example, the generated sensor measurement errors in the simulations may actually be quite a bit higher than in a real-world scenario due to the long distance of the sensors from the actual target. These measurement errors were generated based on the distance between the sensor and the target using the Cobra Dane radar specifications published in 1976 as the baseline (which had an angle measurement error of approximately 278μrad for a target located 1852km away from the sensor [22]). It may very well instead be the case that if a radar was needed to actually track a ballistic target with the intent to take action, that a closer radar would be used, thus decreasing the actual sensor measurement error. Furthermore, the radar specifications utilized in these simulations were published several decades ago and have likely improved since then. Improvement in these two factors (i.e., shorter distance to the target and better radar specifications) would reduce the sensor measurement error and thus also reduce the overall RMS position error beyond the error levels seen in these simulations.

6.2 Contributions

Overall, the main contributions of this thesis fall under four main topics:

- **Machine Learning in Multisensor Fusion for Target Tracking.** The application and investigation of machine learning techniques in multisensor fusion for target tracking.
- **Improved, More Robust ANN Fusers for Target Tracking.** New ANN training procedures were introduced to help enhance the robustness of the ANN fuser against packet losses and limited amounts of available training data.
- **Training Considerations.** Evaluated the impact of various heuristics used in training learning-based fusers.
- **Full System-Level Simulations.** Demonstrated fuser performance using full system-level simulations of two different types of nonlinear targets.

Based on the work in this thesis, several recommendations can be made for applying learning-based fusers to multisensor fusion in target tracking:

1. The ANN fuser was found to give the best performance. When training the ANN, it is recommended to apply the error regularization scheme proposed herein (utilizing the error covariance estimates).
2. The proposed normalization scheme should also be used as it vastly widens the usable training space.
3. It is better to utilize all of the training data to train a single fuser than to split up the training data to train multiple fusers. However, if there are differing target motions, a separate fuser should be used for each type of target motion (e.g., maneuver versus traveling in a straight line).

4. Using more dynamic information (e.g., the previous state estimates projected to the current time step) helps further reduce the error, but the error may increase the further back in time the state estimates are.

5. The optimal number of hidden nodes may vary depending on the type of target.

6. Utilizing the proposed sensitivity approach from Chapter 5 can help further reduce the error and improve the ANNs robustness against packet losses.

6.3 Future Work

While this investigation into learning-based fusers for multisensor fusion in target tracking was certainly not exhaustive, and experiments were run only on simulated data, this thesis intends to provide some suggestions as to what parameters or aspects of the ANN may be explored to help improve fuser performance. Further investigation may be necessary to again reduce the amount of training data required before these nonlinear, machine learning-based fusers can actually be utilized in the field as the cost requirements for field tests may be very high. Several additional topics (beyond the reduction of training data) that may be of future interest may include: a deeper investigation into multi-layered neural networks to devise better pre-training schemes for this application; better utilization of dynamic information; the use of real-world data; and testing different sensitivity metrics to further improve the robustness of the fuser against packet losses. In addition, it may be possible to analytically show that the proposed error regularization leads to better tracking. One would first need to show that using regularized neural networks leads to smoother and more accurate trajectories. Next, since the learning algorithm is not guaranteed to reach a global minimum, one could use a VC-type bound to show the approximation.

Another topic of interest for future work is outlier detection and mitigation (e.g., in the case where the received data is faulty perhaps due to a malfunctioning sensor). The aberrant data can either be rejected or utilized in an algorithm that is designed to be robust

to outliers. There are numerous methods for detecting outliers [75], and data that may be egregiously incorrect (e.g., all zeros) may be detected and discarded. For the outliers that are retained, future work in this area would include testing the proposed sensitivity approach on data containing faulty entries, and perhaps expanding the training algorithm to utilize a different error function, such as that introduced by Liano in [76] for ANNs to be robust against outliers.

Bibliography

[1] N. S. V. Rao, K. Brigham, V. K. Bhagavatula, Q. Liu, and X. Wang, "Effects of computing and communications on state fusion over long-haul networks," in *15th International Conference on Information Fusion*, 2012.

[2] Y. Bar-Shalom, T. Kirubarajan, and X.-R. Li, *Estimation with Applications to Tracking and Navigation.* New York, NY, USA: John Wiley & Sons, Inc., 2002.

[3] Z. Zhao, T. Rong Li, and V. Jilkov, "Best linear unbiased filtering with nonlinear measurements for target tracking," *Aerospace and Electronic Systems, IEEE Transactions on*, vol. 40, no. 4, pp. 1324 – 1336, Oct. 2004.

[4] F. A. A. El-Salam and S. E. A. El-Bar, "Computation of the different errors in the ballistic missiles range," *ISRN Applied Mathematics*, 2011.

[5] B. Bouchon-Meunier, Ed., *Aggregation and Fusion of Imperfect Information.* Physica-Verlag, 1998.

[6] P. K. Varshney, *Distributed Detection and Data Fusion.* Springer-Verlag, 1997.

[7] Y. Bar-Shalom, P. K. Willett, and X. Tian, *Tracking and Data Fusion: A Handbook of Algorithms.* YBS Publishers, 2011.

[8] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks*, vol. 38, no. 4, pp. 393 – 422, 2002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128601003024

[9] "Earth systems research laboratory, national oceanic and atmospheric administration," //www.esrl.noaa.gov/gmd/ccgg/index.html.

[10] "Network telescope research," 2011, www.caida.org/research/security/telescope/.

[11] "Global network of new-generation telescopes will track astrophysical events as they happen," Jan 2011.

[12] Y. Xiao, H. Chen, and F. H. Li, Eds., *Handbook on Sensor Networks.* World Scientific, 2010.

[13] F. Zhao and L. Guibas, *Wireless Sensor Networks.* Elsevier, 2004.

[14] X. Rong Li and V. Jilkov, "Survey of maneuvering target tracking. part i. dynamic models," *Aerospace and Electronic Systems, IEEE Transactions on*, vol. 39, no. 4, pp. 1333 – 1364, Oct. 2003.

[15] H. Wang, T. Kirubarajan, and Y. Bar-Shalom, "Precision large scale air traffic surveillance using an imm estimator with assignment," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 35, no. 1, pp. 255–266, Jan. 1999.

[16] X. Li and V. Jilkov, "Survey of maneuvering target tracking. part ii: Motion models of ballistic and space targets," *Aerospace and Electronic Systems, IEEE Transactions on*, vol. 46, no. 1, pp. 96 –119, Jan. 2010.

[17] M. Yeddanapudi, Y. Bar-Shalom, K. R. Pattipati, and S. Deb, "Ballistic missile track initiation from satellite observations," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 31, no. 3, pp. 1054–1071, 1995.

[18] Z. Zhao, X. Li, V. Jilkov, and Y. Zhu, "Optimal linear unbiased filtering with polar measurements for target tracking," in *Information Fusion, 2002. Proceedings of the Fifth International Conference on*, vol. 2, 2002, pp. 1527 – 1534.

[19] T. Yuan, Y. Bar-Shalom, and X. Tian, "Heterogeneous track-to-track fusion," *Journal of Advances in Information Fusion*, vol. 6, no. 2, pp. 131–149, Dec. 2011.

[20] T. Kerr, "Streamlining measurement iteration for ekf target tracking," *Aerospace and Electronic Systems, IEEE Transactions on*, vol. 27, no. 2, pp. 408 –421, Mar 1991.

[21] G. R. Curry, *Radar System Performance Modeling.* Artech House Publishers, 2004.

[22] E. Filer and J. Hartt, "Cobra dane wideband pulse compression system," in *Proceedings of IEEE EASCON*, 1976, pp. 26–29.

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

[24] S.-L. Sun and Z.-L. Deng, "Multi-sensor optimal information fusion kalman filter," *Automatica*, vol. 40, no. 6, pp. 1017 – 1023, 2004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0005109804000287

[25] S. J. Julier and J. K. Uhlmann, *General Decentralized Data Fusion with Covariance Intersection*, ser. Handbook of Multisensor Data Fusion. Boca Raton, FL: CRC Press, 2001.

[26] Y. Wang and X. Li, "Distributed estimation fusion with unavailable cross-correlation," *Aerospace and Electronic Systems, IEEE Transactions on*, vol. 48, no. 1, pp. 259 –278, Jan. 2012.

[27] X. R. Li and P. Zhang, "Optimal linear estimation fusion - part iii: Cross-correlation of local estimation errors," in *Proc. 2001 Int. Conference Information Fusion*, 2001.

[28] C.-Y. Chong, S. Mori, W. Barker, and K.-C. Chang, "Architectures and algorithms for track association and fusion," *Aerospace and Electronic Systems Magazine, IEEE*, vol. 15, no. 1, pp. 5 –13, Jan 2000.

[29] K. Kim, "Development of track to track fusion algorithms," in *American Control Conference, 1994*, vol. 1, june-1 july 1994, pp. 1037 – 1041 vol.1.

[30] Y. Bar-Shalom, "On the track-to-track correlation problem," *Automatic Control, IEEE Transactions on*, vol. 26, no. 2, pp. 571 – 572, apr 1981.

[31] N. S. V. Rao, "A generic sensor fusion problem: Classification and function estimation," in *Multiple Classifier Systems: 5th International Workshop, MCS 2004. Proceedings, volume 3077 of Lecture.* Springer-Verlag GmbH, 2004, pp. 16–30.

[32] ——, "Nadaraya-watson estimator for sensor fusion problems," in *Proceedings of the 1997 IEEE International Conference on Robotics and Automation*, 1997.

[33] ——, "Measurement-based statistical fusion methods for distributed sensor networks," in *Distributed Sensor Networks*, S. S. Iyengar and R. R. Brooks, Eds. Chapman and Hall/CRC Publishers, 2005.

[34] J. J. Braun, "Sensor data fusion with support vector machine techniques," in *Proceedings of Sensor Fusion: Architectures, Algorithms, and Applications VI*, Orlando, FL, 2002, pp. 98–109.

[35] B. Waske and J. Benediktsson, "Fusion of support vector machines for classification of multisensor data," *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 45, no. 12, pp. 3858 –3866, Dec. 2007.

[36] A. Starzacher and B. Rinner, "Embedded realtime feature fusion based on ann, svm and nbc," in *Information Fusion, 2009. FUSION '09. 12th International Conference on*, July 2009, pp. 482 –489.

[37] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification.* Wiley-Interscience, 2000.

[38] D. Basak, S. Pal, and D. C. Patranabis, "Support vector regression," *Neural Information Processing - Letters and Reviews*, vol. 11, no. 10, pp. 203–224, 2007.

[39] S. S. Blackman and R. Popoli, *Design and Analysis of Modern Tracking Systems.* Boston: Artech House, 1999.

[40] M. K. Sundareshan and F. Amoozegar, "Neural network fusion capabilities for efficient implementation of tracking algorithms," *Opt. Eng.*, vol. 36, no. 3, pp. 692–707, 1997.

[41] J. Zhongliang, X. Hong, and Z. Xueqin, "Information fusion and tracking of maneuvering targets with artificial neural network," in *Neural Networks, 1994. IEEE World Congress on Computational Intelligence., 1994 IEEE International Conference on*, vol. 5, Jun-Jul 1994, pp. 3403 –3408.

[42] S. Gezici, H. Kobayashi, and H. Poor, "A new approach to mobile position tracking," *Proc. 5th IEEE Int. Conf. Universal Personal Communications*, pp. 204–207, Mar 2003.

[43] N. Yadaiah, L. Singh, R. Bapi, V. Rao, B. Deekshatulu, and A. Negi, "Multisensor data fusion using neural networks," in *Neural Networks, 2006. IJCNN '06. International Joint Conference on*, 2006, pp. 875 –881.

[44] F. Chowdhury, "A neural approach to data fusion," in *American Control Conference, 1995. Proceedings of the*, vol. 3, Jun 1995, pp. 1693 –1697.

[45] L.-W. Fong and C.-Y. Fan, "Multisensor fusion algorithms for maneuvering target tracking," in *E-Learning in Industrial Electronics, 2006 1ST IEEE International Conference on*, Dec. 2006, pp. 80 –84.

[46] R. Luo and M. Kay, "Data fusion and sensor integration: State-of-the-art 1990s," *Data Fusion in Robotics and Machine Intelligence*, pp. 7–135, 1992.

[47] M. W. Roth, "Survey of neural network technology for automatic target recognition," *Neural Networks, IEEE Transactions on*, vol. 1, no. 1, pp. 28–43, 1990.

[48] A. Filippidis, L. Jain, and N. Martin, "Fusion of intelligent agents for the detection of aircraft in sar images," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 22, no. 4, pp. 378–384, Apr 2000.

[49] T. W. Lewis and D. M. W. Powers, "Audio-visual speech recognition using red exclusion and neural networks," *Aust. Comput. Sci. Commun.*, vol. 24, no. 1, pp. 149–156, Jan. 2002. [Online]. Available: http://dx.doi.org/10.1145/563857.563819

[50] C. Pohl and J. L. Van Genderen, "Review article multisensor image fusion in remote sensing: concepts, methods and applications," *International journal of remote sensing*, vol. 19, no. 5, pp. 823–854, 1998.

[51] J. A. Benediktsson and I. Kanellopoulos, "Classification of multisource and hyperspectral data based on decision fusion," *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 37, no. 3, pp. 1367–1377, 1999.

[52] C. Cimander, M. Carlsson, and C.-F. Mandenius, "Sensor fusion for on-line monitoring of yoghurt fermentation," *Journal of Biotechnology*, vol. 99, no. 3, pp. 237 – 248, 2002, highlights from {ECB10} - Novel Bioactive Substances and Bioremediation Technologies. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0168165602002134

[53] L. Yiyao, Y. Venkatesh, and C. C. Ko, "A knowledge-based neural network for fusing edge maps of multi-sensor images," *Information Fusion*, vol. 2, no. 2, pp. 121–133, 2001.

[54] P. K. Atrey, M. A. Hossain, A. El Saddik, and M. S. Kankanhalli, "Multimodal fusion for multimedia analysis: a survey," *Multimedia systems*, vol. 16, no. 6, pp. 345–379, 2010.

[55] H. Pasika, S. Haykin, E. Clothiaux, and R. Stewart, "Neural networks for sensor fusion in remote sensing," in *Neural Networks, 1999. IJCNN '99. International Joint Conference on*, vol. 4, 1999, pp. 2772 –2776 vol.4.

[56] V. N. Vapnik, *The nature of statistical learning theory.* New York, NY, USA: Springer-Verlag New York, Inc., 1995.

[57] B. Scholkopf, A. J. Smola, and R. Williamson, "Shrinking the tube: A new support vector regression algorithm," *Advances in neural information processing systems*, 1999.

[58] N. S. V. Rao, "Nearest neighbor projective fuser for function estimation," in *Proceedings of International Conference on Information Fusion*, 2002.

[59] C. M. Bishop, *Neural Networks for Pattern Recognition.* New York, NY, USA: Oxford University Press, Inc., 1995.

[60] K. Priddy and P. Keller, *Artificial Neural Networks: An Introduction.* SPIE Publications, 2005.

[61] S. Geman, E. Bienenstock, and R. Doursat, "Neural networks and the bias/variance dilemma," *Neural Comput.*, vol. 4, no. 1, pp. 1–58, Jan. 1992. [Online]. Available: http://dx.doi.org/10.1162/neco.1992.4.1.1

[62] D. J. MacKay, "Bayesian interpolation," *Neural Computation*, vol. 4, pp. 415–447, 1992.

[63] J. Sietsma and R. J. Dow, "Creating artificial neural networks that generalize," *Neural Networks*, vol. 4, no. 1, pp. 67 – 79, 1991. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0893608091900332

[64] A. Webb, "Functional approximation by feed-forward networks: a least-squares approach to generalization," *Neural Networks, IEEE Transactions on*, vol. 5, no. 3, pp. 363–371, 1994.

[65] C. M. Bishop, "Training with noise is equivalent to tikhonov regularization," *Neural Computation*, vol. 7, pp. 108–116, 1994.

[66] M. Hagan and M. Menhaj, "Training feedforward networks with the marquardt algorithm," *Neural Networks, IEEE Transactions on*, vol. 5, no. 6, pp. 989 –993, Nov 1994.

[67] T. Cacoullos and V. Papathanasiou, "On upper bounds for the variance of functions of random variables," *Statistics and Probability Letters*, vol. 3, no. 4, pp. 175 – 184, 1985. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0167715285900148

[68] H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin, "Exploring strategies for training deep neural networks," *J. Mach. Learn. Res.*, vol. 10, pp. 1–40, Jun. 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1577069.1577070

[69] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, no. 7, pp. 1527–1554, Jul. 2006. [Online]. Available: http://dx.doi.org/10.1162/neco.2006.18.7.1527

[70] A. Engelbrecht and I. Cloete, "Selective learning using sensitivity analysis," in *Neural Networks Proceedings, 1998. IEEE World Congress on Computational Intelligence. The 1998 IEEE International Joint Conference on*, vol. 2, May 1998, pp. 1150–1155 vol.2.

[71] J. Zurada, A. Malinowski, and I. Cloete, "Sensitivity analysis for minimization of input data dimension for feedforward neural network," in *Circuits and Systems, 1994. ISCAS '94., 1994 IEEE International Symposium on*, vol. 6, May 1994, pp. 447–450 vol.6.

[72] P. Ponnapalli, K. Ho, and M. Thomson, "A formal selection and pruning algorithm for feedforward artificial neural network optimization," *Neural Networks, IEEE Transactions on*, vol. 10, no. 4, pp. 964–968, Jul 1999.

[73] S. Piche, "The selection of weight accuracies for madalines," *Neural Networks, IEEE Transactions on*, vol. 6, no. 2, pp. 432–445, Mar 1995.

[74] K. B. Petersen, M. S. Pedersen, J. Larsen, K. Strimmer, L. Christiansen, K. Hansen, L. He, L. Thibaut, M. Barão, S. Hattinger, V. Sima, and W. The, "The matrix cookbook," Tech. Rep., 2006.

[75] P. J. Rousseeuw and A. M. Leroy, *Robust regression and outlier detection.* John Wiley & Sons, 2005, vol. 589.

[76] K. Liano, "Robust error measure for supervised neural network learning with outliers," *Neural Networks, IEEE Transactions on*, vol. 7, no. 1, pp. 246–250, Jan 1996.