2011

The Extended Kalman-Consensus Filter

Andrew Pellett

THE EXTENDED KALMAN-CONSENSUS FILTER

By

Andrew Pellett

B.S. University of Maine, 2009

A THESIS

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

(in Computer Engineering)

The Graduate School

The University of Maine

December, 2011

Advisory Committee:

Donald M. Hummels, Castle Professor of Electrical and Computer Engineering, Advisor

Yifeng Zhu, Associate Professor of Electrical and Computer Engineering

Richard Eason, Associate Professor of Electrical and Computer Engineering

LIBRARY RIGHTS STATEMENT

In presenting this thesis in partial fulfillment of the requirements for an advanced degree at The University of Maine, I agree that the Library shall make it freely available for inspection. I further agree that permission for "fair use" copying of this thesis for scholarly purposes may be granted by the Librarian. It is understood that any copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Signature: _____ Date: _____

Andrew Pellett

THE EXTENDED KALMAN-CONSENSUS FILTER

By Andrew Pellett

Thesis Advisor: Dr. Donald M. Hummels

An Abstract of the Thesis Presented
in Partial Fulfillment of the Requirements for the
Degree of Master of Science
(in Computer Engineering)
December, 2011

An algorithm called the extended Kalman-Consensus filter is developed as an extension of the Kalman-Consensus filter to the non-linear case. The extended Kalman-Consensus filter is a technique for estimating the state of a non-linear process disturbed by noise using multiple observations from a distributed set of sense nodes. All sense nodes attempt to estimate the same state by determining how their observations affect that state, and by communicating with neighbor nodes. The algorithm is designed to be more accurate through measurement diversity, scalable to a large number of nodes, and robust against loss of nodes during operation.

Simulations are used to compare the performance of the algorithm to the standard extended Kalman filter, the central extended Kalman filter, and the distributed extended Kalman filter. The extended Kalman-Consensus filter performs more accurate estimation than the standard and distributed extended Kalman filters, and is more scalable than the central extended Kalman filter, with a similar degree of estimation accuracy.

ACCEPTANCE STATEMENT

On behalf of the Graduate Committee for Andrew Pellett, I affirm that this manuscript is the final and accepted thesis. Signatures of all committee members are on file with the Graduate School at the University of Maine, 42 Stodder Hall, Orono, Maine.

Submitted for graduation in December, 2011.

Signature: _____ Date: _____

Dr. Donald M. Hummels
Professor, Electrical and Computer Engineering

ACKNOWLEDGMENTS

The author would like to thank his committee, and especially committee chair Dr. Don Hummels for providing vital help, extra teaching, and general support beyond expectations.

The author would also like to thank his parents for continued support through college, as well as his friends for keeping him from getting too wrapped up in work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Background

Determining the location and orientation of an object is a fundamental task in many applications, from vehicle navigation, to rocket guidance, to limb tracking. There is a limited set of physical phenomena that can be measured to help determine location and orientation using dead reckoning, including inertial measurements (e.g. acceleration, angular velocity) and magnetic measurements. Dead reckoning measurements are often combined with some sort of absolutely referenced measurement, such as a radionavigation measurement (e.g. GPS, LORAN), to avoid accumulation of errors from numerical approximation of integration.

In reality, all measurements have inherent random error, but by identifying the statistical tendencies of the error, a more accurate measurement can be obtained over time. The Kalman filter is a common approach to this problem, and has been shown to be the optimal estimator for a linear process disturbed by Gaussian noise.

The use of networks of sensors to match or surpass the performance of a single sensor has become a popular area of research, and allows for less expensive, more robust sensing systems.

This work draws from methods for tracking location and orientation, Kalman filtering, and sensor networks to examine the performance of a distributed sensing system for tracking the movements of an object in three dimensional space.

A similar task of distributed tracking of a target with linear kinematics has been heavily explored in [1], [2], and [3]. The original proposed distributed Kalman filtering algorithm involved pushing sensor and covariance data through consensus filters, then performing a standard Kalman filter update on the consensus values. The use of consensus filters allows an average value of each quantity to propogate through the network.

After considering various modifications of the consensus filters, this path culminated in a type of distributed Kalman filter called the Kalman-Consensus filter which integrates a consensus filter on the state estimate with the Kalman filter. The Kalman-Consensus filter builds on the roots of consensus filtering, explored in [4], which aims to allow nodes of a graph to reach a consensus on the value of some quantity by exchanging messages between nodes.

1.2 Purpose

This research develops an algorithm for the application of a non-linear distributed Kalman filtering technique to a set of sparsely-connected sense nodes with the goal of tracking the location and orientation of an object moving in three dimensional space.

This research branches from [2], but considers a similar, yet different, tracking task, where a set of sensors on an object are used to track the object, versus a set of sensors moving independently attempting to track a target.

1.3 Outline

Chapter 2 builds the notation used to describe the concepts underlying the work to follow. Chapter 3 proposes an extended version of the Kalman-Consensus filter in [2] as a technique for estimating the state of a non-linear process via distributed sensing. The details of the simulation developed for testing is discussed in Chapter 4. Chapter 5 analyzes the characteristics of different ways of tracking the location and orientation of an object using a distributed network of sense nodes. Chapter 6 discusses the performance of the techniques from Chapter 5, and draws a conclusion on the extended Kalman-Consensus filter.

Chapter 2

BACKGROUND CONCEPTS

This chapter sets up the basic concepts and notation underlying the rest of the work. It will be helpful to distinguish among scalar, vector, matrix, and quaternion quantities, so the notation for each is given in Table 2.1.

s	Scalar	$\boldsymbol{v}$	Vector
$\underset{\sim}{\boldsymbol{M}}$	Matrix	$\mathfrak{q}$	Quaternion

Table 2.1. Notation for quantity types

2.1 Notation

This section is a quick introduction to the various notation to be used. More careful descriptions of these quantities are given in the following sections.

Newton's dot notation is used for describing linear and angular kinematics, as shown in Table 2.2.

$\boldsymbol{p}$	Position	$\boldsymbol{\theta}$	Angular position
$\dot{\boldsymbol{p}}$	Velocity	$\dot{\boldsymbol{\theta}}$	Angular velocity
$\ddot{\boldsymbol{p}}$	Acceleration	$\ddot{\boldsymbol{\theta}}$	Angular acceleration

Table 2.2. Newton's dot notation for describing linear and angular kinematics

The distinction between frames of reference and their associated coordinate systems is important in some calculations, so a notation is established in Table 2.3. Coordinate system specification will be represented as a superscript.

x^B	Quantity in *body* coordinate system
x^W	Quantity in *world* coordinate system
$\mathfrak{q}^{B\to W}$	Coordinate transformation quaternion

Table 2.3. Notation for specifying coordinate systems

A number of less conventional mathematical operators are used, and their notation is described in 2.4.

$\star$	Quaternion multiply	$\bullet$	Quaternion conjugation
$\times$	Cross product		

Table 2.4. Mathematical operator definitions

2.2 Kinematics

The kinematics of an object moving in three dimensional space are considered in the development and simulation of this research. Weak assumptions are made about the nature of the movement, so the work is tunable to different applications, such as tracking of cars, aircraft, and limbs, all of which have subtle but important tendencies. A car won't climb in altitude as quickly as an aircraft; neither a car or aircraft will change directions nearly as quickly as a limb.

2.2.1 Kinematic Equations

Free movement in three dimensional space can be expressed using conventional linear kinematics in combination with angular (rotational) kinematics.

Some useful relationships between linear and angular position, velocity, and acceleration are described in Equation 2.2.1. The subscript k refers to the current discrete time step, which has duration dt. The duration of time step dt is assumed short (relative to the dynamics of the system), so the acceleration over dt is assumed to be constant.

$$\begin{aligned} \boldsymbol{p}_k &= \boldsymbol{p}_{k-1} + \dot{\boldsymbol{p}}_{k-1}\, dt + \frac{1}{2}\, \ddot{\boldsymbol{p}}_{k-1}\, dt^2 \\ \dot{\boldsymbol{p}}_k &= \dot{\boldsymbol{p}}_{k-1} + \ddot{\boldsymbol{p}}_{k-1}\, dt \\ \ddot{\boldsymbol{\theta}}_{k-1} &= \frac{\dot{\boldsymbol{\theta}}_k - \dot{\boldsymbol{\theta}}_{k-1}}{dt} \end{aligned} \tag{2.2.1}$$

2.3 Coordinate Systems

It is necessary to differentiate between quantities in the *body* frame of reference versus the *world* frame of reference. Ultimately, the quantities in the *world* frame are most useful; they describe the motion of the object from an external point of view that can be referenced to other objects. For example, the *world* frame position of an aircraft can be used in guiding a precision approach to a runway.

Not all measurements are made in the *world* frame, though. For instance, inertial measurements taken on the body of the object being tracked are in a *body* frame that is separate from the external *world* frame. This is also true of rotational measurements from gyroscopes. While the *world* frame remains fixed, the *body* frame is constantly changing orientation relative to the *world* frame, and this difference in orientation must be kept track of in order to describe *body* frame quantities in the *world* frame, and vice versa.

As a result, it is important to keep track of the frame of reference associated with a quantity, as well as some way to translate between quantities in the *world* frame and the *body* frame. Quaternions are helpful for the latter task.

2.4 Quaternions

There are a few ways to describe the orientation of an object in three dimensional space (e.g. Euler angles, axis-angle, rotation matrix), but unit quaternions have a few key advantages over other representations (note: only unit quaternions are considered in this work, so the "unit" distinction will be dropped).

Quaternions are a four parameter description of rotations in three dimensional space. The extra parameter makes the quaternion more robust. Quaternions are immune to the phenomenon of gimbal lock (a singularity that occurs when two rotation axes become coplanar, so there is no distinction between them, and a degree-of-freedom is

lost). Numerical precision is less of an issue for quaternions — a small error in quaternion components is a small error in represented rotation — versus rotation matrices, which must be kept orthogonal. Quaternions are also computationally efficient.

Gyroscopes sense a rotation rate along a given axis, so their readings translate directly to an axis-angle form. The reading can be integrated to determine an angle of rotation about some *body* frame vector.

Given an axis unit vector $\boldsymbol{u}$, and a rotation θ about $\boldsymbol{u}$, obtaining a quaternion representation of the rotation is straightforward, and described in Equation 2.4.1.

$$\begin{aligned} \boldsymbol{u} &= (u_x, u_y, u_z) \\ \mathfrak{q} &= \left(\cos(\frac{\theta}{2}), \boldsymbol{u} sin(\frac{\theta}{2})\right) \\ &= \left(\cos(\frac{\theta}{2}), u_x sin(\frac{\theta}{2}), u_y sin(\frac{\theta}{2}), u_z sin(\frac{\theta}{2})\right) \end{aligned} \tag{2.4.1}$$

For example, consider a rotation of θ about the x axis. The resulting quaternion is shown in Equation 2.4.2.

$$\begin{aligned} \boldsymbol{u} &= (1, 0, 0) \\ \mathfrak{q} &= \left(\cos(\frac{\theta}{2}),\ \sin(\frac{\theta}{2}),\ 0,\ 0\right) \end{aligned} \tag{2.4.2}$$

The product of two quaternions is a quaternion representating the combination of each original quaternion's rotation. The operation of quaternion multiplication will be denoted by $\star$, and is defined in Equation 2.4.3.

$$\mathfrak{q}_a = (a_1,\ a_2,\ a_3,\ a_4) \qquad\qquad \mathfrak{q}_b = (b_1,\ b_2,\ b_3,\ b_4)$$

$$\begin{aligned} r_1 &= a_1b_1 - a_2b_2 - a_3b_3 - a_4b_4 \\ r_2 &= a_1b_2 - a_2b_1 - a_3b_4 - a_4b_3 \\ r_3 &= a_1b_3 - a_2b_4 - a_3b_1 - a_4b_2 \\ r_4 &= a_1b_4 - a_2b_3 - a_3b_2 - a_4b_1 \end{aligned}$$

$$\begin{aligned} \mathfrak{q}_{result} &= \mathfrak{q}_a \star \mathfrak{q}_b \\ &= (r_1, r_2, r_3, r_4) \end{aligned} \tag{2.4.3}$$

The properties of quaternion multiplication allow for more complex quaternions to be built up incrementally from simple quaternions, as shown in Equation 2.4.4.

$$\begin{aligned} \mathfrak{q}_x &= \left(\cos(\frac{\theta_x}{2}),\ \sin(\frac{\theta_x}{2}),\ 0,\ 0\right) \\ \mathfrak{q}_y &= \left(\cos(\frac{\theta_y}{2}),\ 0,\ \sin(\frac{\theta_y}{2}),\ 0\right) \\ \mathfrak{q}_z &= \left(\cos(\frac{\theta_z}{2}),\ 0,\ 0,\ \sin(\frac{\theta_z}{2})\right) \end{aligned} \tag{2.4.4}$$

$$\mathfrak{q}_{new} = \mathfrak{q}_{old} \star \mathfrak{q}_x \star q_y \star q_z \tag{2.4.5}$$

Since the rotation quaternions being used are unit quaternions, there is no increase in magnitude over time, so the quaternion is a stable representation of overall orientation that can be updated indefinitely, as described by Equation 2.4.5.

The act of rotating a vector by a quaternion is called *conjugation* by a quaternion. Conjugation, denoted by $\bullet$ and described in Equation 2.4.6, is composed of a series of quaternion multiplies. Conjugation requires the use of the quaternion inverse, defined in Equation 2.4.7. Since quaternions and vectors are different types of quantities, some padding is necessary to allow the vector quantity to be used in a quaternion multiply.

$$\begin{aligned} \boldsymbol{v}_{new} &= \mathfrak{q} \bullet \boldsymbol{v} \\ \mathfrak{q}_v &= (0, v_1, v_2, v_3) \\ \mathfrak{q}_{vnew} &= \mathfrak{q} \star \mathfrak{q}_v \star \mathfrak{q}^{-1} \\ \boldsymbol{v}_{new} &= (q_{vnew,2},\ q_{vnew,3},\ q_{vnew,4}) \end{aligned} \tag{2.4.6}$$

$$\mathfrak{q}^{-1} = (q_1,\ -q_2,\ -q_3,\ -q_4) \tag{2.4.7}$$

2.5 Projection

In describing the motion of a rigid body, the quantities defined in Table 2.2 are the primary concerns. The angular kinematics are assumed constant across the body, as is the nature of a rigid body, but the linear kinematics require more careful consideration.

A rigid physical connection between sets of sense nodes is assumed, and is used to describe linear movements at one point on the body given measurements at another point. The vector $\boldsymbol{r}$ is the vector pointing from the sensing point to the tracking point. The convention for referring to different physical positions on the object is described in Table 2.5.

x_t	Quantity at tracking point
x_s	Quantity at sensing point
$\boldsymbol{r}$	Vector between tracking and sensing points

Table 2.5. Notation for quantities at different points on an object

The equations for this relationship begin with the simple case of expressing the *world* position of a tracking point in terms of the position of a sensing point, where the

sensing point *world* position $\boldsymbol{p}_s^W$ and a *body* vector between the tracking and sensing points $\boldsymbol{r}$ are known. Equation 2.5.1 describes this relationship, which is simple vector addition with the added complexity of a rotating frame of reference. This relationship is described graphically in Figure 2.1.

Observing that the change in r over time is governed by ω, the time derivative of r can be written as Equation 2.5.2, which allows the simple position equation to be differentiated to express the same relation for velocity and acceleration in Equations 2.5.3 and 2.5.4, respectively.

$$\boldsymbol{p}_t^W = \boldsymbol{p}_s^W + \mathfrak{q}^{B\to W}\boldsymbol{r}^B = \boldsymbol{p}_s^W + \boldsymbol{r}^W \tag{2.5.1}$$

$$\dot{\boldsymbol{r}} = \boldsymbol{\omega} \times \boldsymbol{r} \tag{2.5.2}$$

$$\dot{\boldsymbol{p}}_t^W = \dot{\boldsymbol{p}}_s^W + \dot{\boldsymbol{r}}^W = \dot{\boldsymbol{p}}_s^W + \boldsymbol{\omega} \times \boldsymbol{r}^W \tag{2.5.3}$$

$$\ddot{\boldsymbol{p}}_t^W = \mathfrak{q}^{B\to W} \bullet \Big(\underbrace{\ddot{\boldsymbol{p}}_s^B}_{\text{term 1}} + \underbrace{\ddot{\boldsymbol{\theta}}^B \times \left(\boldsymbol{p}_t^B - \boldsymbol{p}_s^B\right)}_{\text{term 2 (tangential)}} + \underbrace{\dot{\boldsymbol{\theta}}^B \times \left(\dot{\boldsymbol{\theta}}^B \times \left(\boldsymbol{p}_t^B - \boldsymbol{p}_s^B\right)\right)}_{\text{term 3 (radial)}} \Big) \tag{2.5.4}$$

2.6 The Kalman Filter

The Kalman filter is a method for optimal estimation (least square error) of the state of a linear process disturbed by Gaussian noise, as described by Equation 2.6.1. The state is denoted $\boldsymbol{x}$, while $\underset{\sim}{\boldsymbol{A}}$ is the process matrix, which describes how the state evolves from one time step to the next. The Gaussian noise $\boldsymbol{w}$ is shaped by the process noise matrix $\underset{\sim}{\boldsymbol{B}}$ to represent the noise that disturbs the process evolution.

Evolution of the state of the process is inferred from observations, $\boldsymbol{z}$, which are linearly related to the unknown state, and also disturbed by noise, as shown in Equation 2.6.2. The observation matrix $\underset{\sim}{\boldsymbol{H}}$ relates the state of the system to a corresponding observation, once again disturbed by Gaussian noise, $\boldsymbol{v}$.

Each successive state estimate is based on the previous best estimate of the system state, which allows efficient real-time calculation. Discrete time indexes are

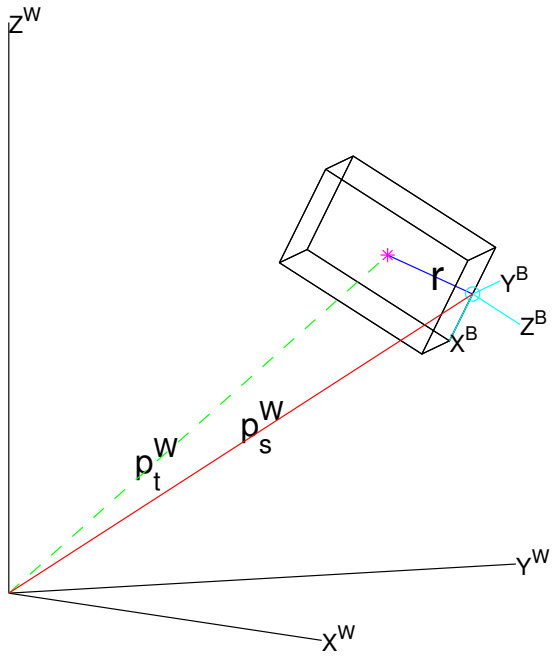

Figure 2.1. World versus body coordinate systems

dropped in this Kalman filter notation (to reduce clutter), but the superscript $+$ (e.g. $\boldsymbol{x}^+$) is used to differentiate a quantity at the next time step from a quantity at the current time step.

$$\boldsymbol{x}^+ = \underset{\sim}{\boldsymbol{A}}\boldsymbol{x} + \underset{\sim}{\boldsymbol{B}}\boldsymbol{w} \tag{2.6.1}$$

$$\boldsymbol{z} = \underset{\sim}{\boldsymbol{H}}\boldsymbol{x} + \boldsymbol{v} \tag{2.6.2}$$

In implementation, the Kalman filter consists of two stages, where estimates of the state are formed, and a covariance matrix representing the confidence in the state estimate is maintained. The predict stage uses the previous state estimate and a process matrix that describes how the linear process evolves in time to form a prediction of the current state. This predicted state is then modified by incorporating the difference between actual observations and an educated guess at what the observations should be. This is called the update stage, and produces the updated state estimate, which then feeds back into the predict stage to advance to the next time step. A block diagram overview of the Kalman filter is shown in Figure 2.2. Equations for calculating the Kalman filter are given in Equation 2.6.3.

Definitions of the set of variables involved in the Kalman filter are given in Table 2.6. Dimensions of each variable are given as subscripts. n represents the number of elements in the state vector. m represents the number of observations being made. l represents the dimension of the process noise.

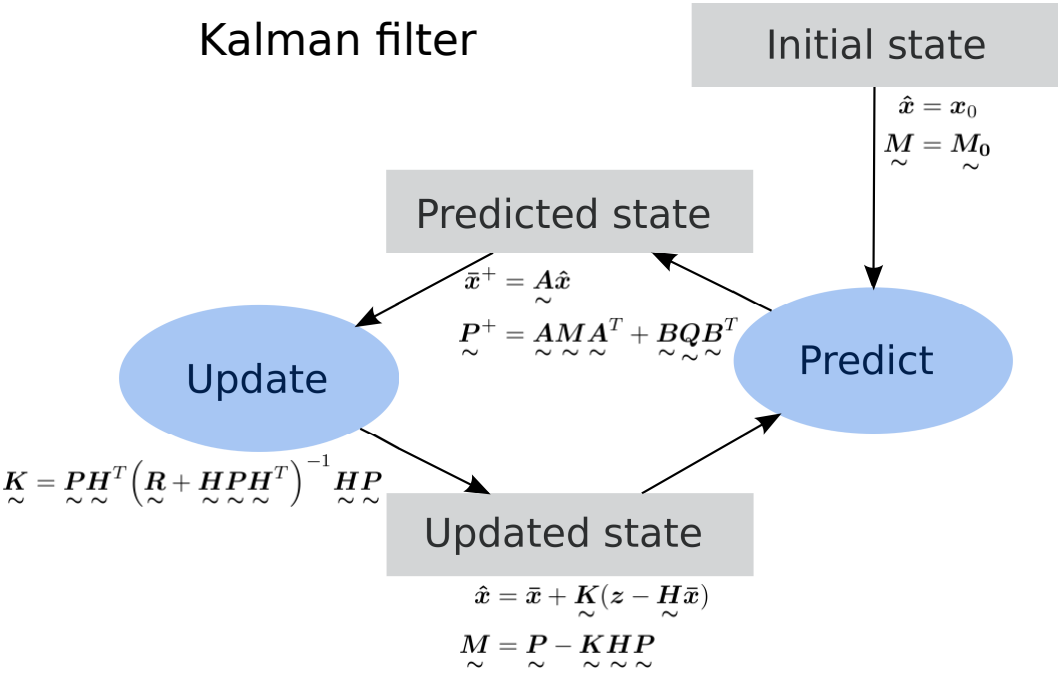

Figure 2.2. Kalman filter block diagram

$\boldsymbol{z}_{m\times 1}$	Observations	$\underset{\sim}{\boldsymbol{H}}_{m\times n}$	Observation matrix
$\boldsymbol{x}_{n\times 1}$	Actual state	$\underset{\sim}{\boldsymbol{K}}_{n\times m}$	Kalman gain
$\bar{\boldsymbol{x}}_{n\times 1}$	Predicted state	$\underset{\sim}{\boldsymbol{P}}_{n\times n}$	Predicted state covariance
$\hat{\boldsymbol{x}}_{n\times 1}$	Updated state	$\underset{\sim}{\boldsymbol{M}}_{n\times n}$	Updated state covariance
$\boldsymbol{w}_{l\times 1}$	Process noise	$\underset{\sim}{\boldsymbol{Q}}_{l\times l}$	Process noise covariance
$\boldsymbol{v}_{m\times 1}$	Observation noise	$\underset{\sim}{\boldsymbol{R}}_{m\times m}$	Observation noise matrix
$\underset{\sim}{\boldsymbol{A}}_{n\times n}$	Process matrix	$\underset{\sim}{\boldsymbol{B}}_{n\times l}$	Process noise matrix

Table 2.6. Kalman variable definitions

$$
\begin{aligned}
&\text{Update stage} \begin{cases} \hat{\boldsymbol{x}} = \bar{\boldsymbol{x}} + \underset{\sim}{\boldsymbol{K}} \left(\boldsymbol{z} - \underset{\sim}{\boldsymbol{H}} \bar{\boldsymbol{x}} \right) \\ \underset{\sim}{\boldsymbol{M}} = \underset{\sim}{\boldsymbol{P}} - \underset{\sim}{\boldsymbol{K}} \underset{\sim}{\boldsymbol{H}} \underset{\sim}{\boldsymbol{P}} \end{cases} \\
&\text{Kalman gain} \left\{ \underset{\sim}{\boldsymbol{K}} = \underset{\sim}{\boldsymbol{P}} \underset{\sim}{\boldsymbol{H}}^T \left(\underset{\sim}{\boldsymbol{R}} + \underset{\sim}{\boldsymbol{H}} \underset{\sim}{\boldsymbol{P}} \underset{\sim}{\boldsymbol{H}}^T \right)^{-1} \underset{\sim}{\boldsymbol{H}} \underset{\sim}{\boldsymbol{P}} \right. \\
&\text{Predict stage} \begin{cases} \bar{\boldsymbol{x}}^+ = \underset{\sim}{\boldsymbol{A}} \hat{\boldsymbol{x}} \\ \underset{\sim}{\boldsymbol{P}}^+ = \underset{\sim}{\boldsymbol{A}} \underset{\sim}{\boldsymbol{M}} \underset{\sim}{\boldsymbol{A}}^T + \underset{\sim}{\boldsymbol{B}} \underset{\sim}{\boldsymbol{Q}} \underset{\sim}{\boldsymbol{B}}^T \end{cases}
\end{aligned}
\tag{2.6.3}
$$

2.6.1 Information Form

A common alteration to the Kalman filter involves using the inverse of the covariance matrix and a modified version of the state vector to describe the process. This allows multiple observations to be combined in one update stage, rather than using multiple update stages as the traditional Kalman filter would require. Because this work focuses on distributed measurements, it will be useful to take advantage of information form later on. The information form of the Kalman filter equations is shown in Equations 2.6.4 through 2.6.8.

$$\hat{\boldsymbol{x}} = \bar{\boldsymbol{x}} + \underset{\sim}{\boldsymbol{M}} \left(\boldsymbol{y} - \underset{\sim}{\boldsymbol{S}} \bar{\boldsymbol{x}} \right) \tag{2.6.4}$$

$$\underset{\sim}{\boldsymbol{M}} = (\underset{\sim}{\boldsymbol{P}}^{-1} + \underset{\sim}{\boldsymbol{S}})^{-1} \tag{2.6.5}$$

$$\underset{\sim}{\boldsymbol{S}} = \underset{\sim}{\boldsymbol{M}}^T \underset{\sim}{\boldsymbol{R}}^{-1} \underset{\sim}{\boldsymbol{H}} \tag{2.6.6}$$

$$\bar{\boldsymbol{x}}^+ = \underset{\sim}{\boldsymbol{A}} \hat{\boldsymbol{x}} \tag{2.6.7}$$

$$\underset{\sim}{\boldsymbol{P}}^+ = \underset{\sim}{\boldsymbol{A}} \underset{\sim}{\boldsymbol{M}} \underset{\sim}{\boldsymbol{A}}^T + \underset{\sim}{\boldsymbol{B}} \underset{\sim}{\boldsymbol{Q}} \underset{\sim}{\boldsymbol{B}}^T \tag{2.6.8}$$

2.7 The Extended Kalman Filter

The Kalman filter can be extended to handle non-linear processes, as well as non-linear observations; this is called the extended Kalman filter. This research deals only with non-linearities in the observations, so Equation 2.7.2 replaces Equation 2.6.2, but Equation 2.7.1 is not used in the extended Kalman filter considered here.

$$\boldsymbol{x}^{+} = \boldsymbol{a}(\boldsymbol{x}) + \boldsymbol{b}(\boldsymbol{x}, \boldsymbol{w}) \tag{2.7.1}$$

$$\boldsymbol{z} = \boldsymbol{h}(\boldsymbol{x}) + \boldsymbol{v} \tag{2.7.2}$$

The non-linear nature of the extended Kalman filter is realized by replacing terms involving static matrices with functions that return similar, but dynamic, matrices. When evaluated at a point in the state space, the functions (e.g. $\boldsymbol{a}(\boldsymbol{x})$) produce the corresponding term in the Kalman equations, which is valid only at that point in the state space. This is illustrated in Equation 2.7.3.

$$\begin{aligned} \boldsymbol{a}(\boldsymbol{x})|_{\boldsymbol{x}=\boldsymbol{x_k}} &= \underset{\sim}{\boldsymbol{A}}\boldsymbol{x}_k \\ \boldsymbol{b}(\boldsymbol{x}, \boldsymbol{w})|_{\boldsymbol{x}=\boldsymbol{x_k}} &= \underset{\sim}{\boldsymbol{B}}\boldsymbol{w} \\ \boldsymbol{h}(\boldsymbol{x})|_{\boldsymbol{x}=\boldsymbol{x_k}} &= \underset{\sim}{\boldsymbol{H}}\boldsymbol{x}_k \end{aligned} \tag{2.7.3}$$

There are other non-linear Kalman filters (e.g. unscented Kalman filter), but they are not addressed in this research.

The general idea behind the extended Kalman filter for a linear process with non-linear observations is to choose a point in the state space to linearize around in order to make an estimate of the next observation. The linearization involves calculating the first order Taylor series expansion around the current state estimate, which requires maintaining a Jacobian matrix of the observation matrix, as in Equation 2.7.5.

$$\boldsymbol{h}(\boldsymbol{x}) \approx \boldsymbol{h}(\bar{\boldsymbol{x}}) + \underbrace{\frac{d\boldsymbol{h}(\bar{\boldsymbol{x}})}{d\bar{\boldsymbol{x}}}}_{\text{Jacobian}}(\boldsymbol{x} - \bar{\boldsymbol{x}}) \tag{2.7.4}$$

$$\underset{\sim}{\check{\boldsymbol{H}}} = \frac{d\boldsymbol{h}(\bar{\boldsymbol{x}})}{d\bar{\boldsymbol{x}}} \tag{2.7.5}$$

The Jacobian matrix $\underset{\sim}{\check{\boldsymbol{H}}}$ describes how the observation changes for small changes in the state, so the product of the Jacobian and the state difference gives the change in observation between the observations at $\boldsymbol{x}$ and $\bar{\boldsymbol{x}}$.

Because the estimate of the next observation involves linearizing a non-linear function, there is some amount of error introduced, so the extended Kalman filter can no longer be shown to be an optimal estimation technique. If too much error is introduced by the linearization, the estimate of the next observation will diverge from the true next observation, and thereby the filter's state estimate will diverge from the actual state. This makes the extended Kalman filter a bit delicate, but still effective if carefully designed.

The equations for calculating the extended Kalman filter on a linear process with non-linear observations are given in Equation 2.7.6.

$$
\begin{aligned}
&\text{Update stage} \begin{cases} \hat{\boldsymbol{x}} = \bar{\boldsymbol{x}} + \underset{\sim}{\boldsymbol{K}} \left(\boldsymbol{z} - \boldsymbol{h}(\bar{\boldsymbol{x}}) \right) \\ \underset{\sim}{\boldsymbol{M}} = \underset{\sim}{\boldsymbol{P}} - \underset{\sim}{\boldsymbol{K}} \underset{\sim}{\check{\boldsymbol{H}}} \underset{\sim}{\boldsymbol{P}} \end{cases} \\
&\text{Kalman gain} \left\{ \underset{\sim}{\boldsymbol{K}} = \underset{\sim}{\boldsymbol{P}} \underset{\sim}{\check{\boldsymbol{H}}}^T \left(\underset{\sim}{\boldsymbol{R}} + \underset{\sim}{\check{\boldsymbol{H}}} \underset{\sim}{\boldsymbol{P}} \underset{\sim}{\check{\boldsymbol{H}}}^T \right)^{-1} \underset{\sim}{\check{\boldsymbol{H}}} \underset{\sim}{\boldsymbol{P}} \right. \\
&\text{Predict stage} \begin{cases} \bar{\boldsymbol{x}}^+ = \underset{\sim}{\boldsymbol{A}} \hat{\boldsymbol{x}} \\ \underset{\sim}{\boldsymbol{P}}^+ = \underset{\sim}{\boldsymbol{A}} \underset{\sim}{\boldsymbol{M}} \underset{\sim}{\boldsymbol{A}}^T + \underset{\sim}{\boldsymbol{B}} \underset{\sim}{\boldsymbol{Q}} \underset{\sim}{\boldsymbol{B}}^T \end{cases}
\end{aligned}
\tag{2.7.6}
$$

Chapter 3

THE EXTENDED KALMAN-CONSENSUS FILTER

The distributed Kalman-Consensus filter proposed in [2], [3] applies to a linear process with linear observations. In this chapter, a distributed extended Kalman-Consensus filter is proposed to handle a linear process with non-linear observations.

3.1 The Kalman-Consensus Filter

The Kalman-Consensus filter fuses the information form of the Kalman filter with the concept of a consensus filter, creating a distributed filter that converges to a common state estimate. This is desirable in a distributed Kalman filtering scheme, because it eliminates the need for a single master node, making the system more robust against losing nodes. It also makes the system more flexible, in that it is possible that the network of nodes could be dynamic.

The consensus filter, which has roots in graph theory and linear algebra, provides an established method for causing multiple estimates of some quantity to converge to the same estimate. The amount of time it takes for consensus to be reached can vary widely, and is heavily affected by the architecture of the communication network between nodes.

The Kalman-Consensus filter was proposed in [2], [3], and lead to the algorithm described in Equations 3.1.2 through 3.1.7. The subscripts refer to a specific node. The set of neighbor nodes that communicate to (one-way) a node i are denoted by $\boldsymbol{N}_i$, while $\boldsymbol{J}_i$ is simply the set $\boldsymbol{N}_i$ with node i included.

$$\boldsymbol{J}_i = \boldsymbol{N}_i \cup \{i\} \tag{3.1.1}$$

Each node communicates the two quantities shown in 3.1.2, as well as the node's current state estimate $\bar{\boldsymbol{x}}_i$, to all the nodes it has an outgoing communication link with.

$$\boldsymbol{u}_i = \underset{\sim}{\boldsymbol{H}_i}^T \underset{\sim}{\boldsymbol{R}_i}^{-1} \boldsymbol{z}_i \qquad\qquad \underset{\sim}{\boldsymbol{U}_i} = \underset{\sim}{\boldsymbol{H}_i}^T \underset{\sim}{\boldsymbol{R}_i}^{-1} \underset{\sim}{\boldsymbol{H}_i} \tag{3.1.2}$$

As each node receives messages from other nodes, the receiving node sums up the two quantities in the message, as shown in 3.1.3. This term will take the place of the sensed observations term in the Kalman update stage.

$$\boldsymbol{y}_i = \sum_{j \in \boldsymbol{J}_i} \boldsymbol{u}_j \qquad\qquad \underset{\sim}{\boldsymbol{S}_i} = \sum_{j \in \boldsymbol{J}_i} \underset{\sim}{\boldsymbol{U}_j} \tag{3.1.3}$$

Equation 3.1.4 is the new version of the Kalman update stage, using the summed, weighted sensed observations and predicted observations. The corresponding update covariance and predict stage are given in Equations 3.1.5 through 3.1.7.

$$\hat{\boldsymbol{x}}_i = \bar{\boldsymbol{x}}_i + \underset{\sim}{\boldsymbol{M}_i}(\boldsymbol{y}_i - \underset{\sim}{\boldsymbol{S}_i}\bar{\boldsymbol{x}}_i) + \gamma \underset{\sim}{\boldsymbol{P}_i} \sum_{j \in \boldsymbol{N}_i} (\bar{\boldsymbol{x}}_j - \bar{\boldsymbol{x}}_i) \tag{3.1.4}$$

$$\underset{\sim}{\boldsymbol{M}_i} = (\underset{\sim}{\boldsymbol{P}_i}^{-1} + \underset{\sim}{\boldsymbol{S}_i})^{-1} \tag{3.1.5}$$

$$\bar{\boldsymbol{x}}_i^+ = \underset{\sim}{\boldsymbol{A}}\hat{\boldsymbol{x}}_i \tag{3.1.6}$$

$$\underset{\sim}{\boldsymbol{P}_i}^+ = \underset{\sim}{\boldsymbol{A}}\underset{\sim}{\boldsymbol{M}_i}\underset{\sim}{\boldsymbol{A}}^T + \underset{\sim}{\boldsymbol{B}}\underset{\sim}{\boldsymbol{Q}}\underset{\sim}{\boldsymbol{B}}^T \tag{3.1.7}$$

The constant ϵ, from Equation 3.1.8, acts as a scale factor on the consensus gain γ in Equation 3.1.4, and is used to determine how much of an effect the consensus term has on the update stage state estimate. In [2] and [3], there is some discussion of how to set ϵ, but the general advice is that ϵ should be proportional to the discrete time step size.

$$\gamma = \frac{\epsilon}{\|\underset{\sim}{\boldsymbol{P_i}}\| + 1} \qquad \|\underset{\sim}{\boldsymbol{X}}\| = tr(\underset{\sim}{\boldsymbol{X}}^T \underset{\sim}{\boldsymbol{X}})^{\frac{1}{2}} \tag{3.1.8}$$

As discussed in [3], this Kalman-Consensus filter is not an optimal estimator even for a process that evolves in a linear fashion with linear observations, because the optimal Kalman gain is not used. Calculating the optimal Kalman gain seems to require all-to-all communication, which would impair the robustness of the system.

3.2 Derivation for the Non-Linear Case

In the linear case, the Kalman-Consensus filter involves exchanging weighted versions of the observations and measurement covariance ($\boldsymbol{u}_i$, $\underset{\sim}{\boldsymbol{U}_i}$ respectively) between nodes. The non-linear case isn't as straightforward, because the quantities being exchanged are weighted versions of *the difference* between the sensed observation and the predicted observation, expressed in Equation 3.2.2, where subscript s denotes the sending node, and subscript c denotes the receiving and calculating node.

$$\underset{\sim}{\boldsymbol{U}_i} = \underset{\sim}{\boldsymbol{H}_i}^T \underset{\sim}{\boldsymbol{R}_i}^{-1} \underset{\sim}{\boldsymbol{H}_i} \qquad \rightarrow \qquad \underset{\sim}{\boldsymbol{U}_s} = \underset{\sim}{\check{\boldsymbol{H}}_s}^T \underset{\sim}{\boldsymbol{R}_s}^{-1} \underset{\sim}{\check{\boldsymbol{H}}_s} \tag{3.2.1}$$

$$\boldsymbol{u}_i = \underset{\sim}{\boldsymbol{H}_i}^T \underset{\sim}{\boldsymbol{R}_i}^{-1} \boldsymbol{z}_i \qquad \rightarrow \qquad \boldsymbol{u}_s = \underset{\sim}{\check{\boldsymbol{H}}_s}^T \underset{\sim}{\boldsymbol{R}_s}^{-1} (\boldsymbol{z}_s - \boldsymbol{h}_s(\bar{\boldsymbol{x}}_c)) \tag{3.2.2}$$

Each node calculates its own observation estimate $\boldsymbol{h}_i(\boldsymbol{x})$ and Jacobian $\underset{\sim}{\check{\boldsymbol{H}}_i}$ using its own state estimate $\bar{\boldsymbol{x}}_i$.

When a node sums weighted observations (i.e. Equation 3.1.3), it needs to sum using the difference between the sensed observation and predicted observation *at the calculating node's state*. This leads to a new step in the algorithm for the non-linear case. First, consider the weighted observation being sent out by each node. Equation 3.2.3 is the same as in the linear case, and uses only each node's own information. This same message is sent to each other node the sending node is communicating with (i.e. a broadcast).

$$\boldsymbol{u}_s = \underset{\sim}{\check{\boldsymbol{H}}_{\boldsymbol{s}}}^T \underset{\sim}{\boldsymbol{R}_{\boldsymbol{s}}}^{-1}(\boldsymbol{z}_s - \boldsymbol{h}_s(\bar{\boldsymbol{x}}_s)) \tag{3.2.3}$$

In order to find the difference between the sensed observation and predicted observation *at the calculating node's state*, it is necessary to look back at the fundamental linearization of the extended Kalman filter, Equation 3.2.4. This suggests a way to calculate the sending node's estimate of what should be observed at the calculating node's state, given in Equation 3.2.5.

$$\boldsymbol{h}_s(\boldsymbol{x}_s) \approx \boldsymbol{h}_s(\bar{\boldsymbol{x}}_s) + \underset{\sim}{\check{\boldsymbol{H}}_{\boldsymbol{s}}}(\boldsymbol{x}_s - \bar{\boldsymbol{x}}_s) \tag{3.2.4}$$

$$\boldsymbol{h}_s(\bar{\boldsymbol{x}}_c) = \boldsymbol{h}_s(\bar{\boldsymbol{x}}_s) + \underset{\sim}{\check{\boldsymbol{H}}_{\boldsymbol{s}}}(\bar{\boldsymbol{x}}_c - \bar{\boldsymbol{x}}_s) \tag{3.2.5}$$

Using this result, the desired difference can be calculated as Equation 3.2.6. Because the information form is being used, this difference needs to be weighted, as in Equation 3.2.7. Distributing the weighting terms leads to Equation 3.2.9. The two underlined terms are the non-linear equivalent of the two quantities that each node sends to its neighbors in the linear Kalman-Consensus filter. Additionally, the state of each node must be sent in order to calculate the state difference.

$$\boldsymbol{z}_s - \boldsymbol{h}_s(\bar{\boldsymbol{x}}_c) = \boldsymbol{z}_s - \boldsymbol{h}_s(\bar{\boldsymbol{x}}_s) - \underset{\sim}{\check{\boldsymbol{H}}_{\boldsymbol{s}}}(\bar{\boldsymbol{x}}_c - \bar{\boldsymbol{x}}_s) \tag{3.2.6}$$

$$\boldsymbol{d}_s = \underset{\sim}{\check{\boldsymbol{H}}_{\boldsymbol{s}}}^T \underset{\sim}{\boldsymbol{R}_{\boldsymbol{s}}}^{-1}(\boldsymbol{z}_s - \boldsymbol{h}_s(\bar{\boldsymbol{x}}_c)) \tag{3.2.7}$$

$$= \underset{\sim}{\check{\boldsymbol{H}}_{\boldsymbol{s}}}^T \underset{\sim}{\boldsymbol{R}_{\boldsymbol{s}}}^{-1}(\boldsymbol{z}_s - \boldsymbol{h}_s(\bar{\boldsymbol{x}}_s) - \underset{\sim}{\check{\boldsymbol{H}}_{\boldsymbol{s}}}(\bar{\boldsymbol{x}}_c - \bar{\boldsymbol{x}}_s)) \tag{3.2.8}$$

$$= \underbrace{\underset{\sim}{\check{\boldsymbol{H}}_{\boldsymbol{s}}}^T \underset{\sim}{\boldsymbol{R}_{\boldsymbol{s}}}^{-1}(\boldsymbol{z}_s - \boldsymbol{h}_s(\bar{\boldsymbol{x}}_s))}_{\boldsymbol{u}_s} - \underbrace{\underset{\sim}{\check{\boldsymbol{H}}_{\boldsymbol{s}}}^T \underset{\sim}{\boldsymbol{R}_{\boldsymbol{s}}}^{-1} \underset{\sim}{\check{\boldsymbol{H}}_{\boldsymbol{s}}}(\bar{\boldsymbol{x}}_c - \bar{\boldsymbol{x}}_s)}_{\underset{\sim}{\boldsymbol{U}_s}} \tag{3.2.9}$$

Finally, the receiving and calculating node must sum the $\boldsymbol{d}_s$ terms for each node that it receives a message from, as in Equation 3.2.10. The neighborhood of nodes sending messages to the calculating node is denoted $\boldsymbol{J}_c$ (includes the calculating node).

$$\begin{aligned} \boldsymbol{g}_c &= \sum_{s \in \boldsymbol{J}_c} \boldsymbol{d}_s \\ \boldsymbol{g}_c &= \sum_{s \in \boldsymbol{J}_c} \boldsymbol{u}_s - \underset{\sim}{\boldsymbol{U}_s}(\bar{\boldsymbol{x}}_c - \bar{\boldsymbol{x}}_s) \end{aligned} \tag{3.2.10}$$

Equation 3.2.10 is a bit different from its linear analog in Equation 3.1.3, and this induces a change in the non-linear update equation, Equation 3.2.11. Because the difference $\boldsymbol{y}_i - \underset{\sim}{\boldsymbol{S}_i}\bar{\boldsymbol{x}}_i$ is calculated at an earlier step, Equation 3.2.10, only $\boldsymbol{g}_c$ appears in the update equation.

$$\hat{\boldsymbol{x}}_c = \bar{\boldsymbol{x}}_c + \underset{\sim}{\boldsymbol{M}_c}\boldsymbol{g}_c + \gamma \underset{\sim}{\boldsymbol{P}_c} \sum_{s \in \boldsymbol{N}_c} (\bar{\boldsymbol{x}}_s - \bar{\boldsymbol{x}}_c) \tag{3.2.11}$$

$$\underset{\sim}{\boldsymbol{M}_c} = (\underset{\sim}{\boldsymbol{P}_c}^{-1} + \underset{\sim}{\boldsymbol{S}_c})^{-1} \tag{3.2.12}$$

$$\bar{\boldsymbol{x}}_c = \underset{\sim}{\boldsymbol{A}}\hat{\boldsymbol{x}}_c \tag{3.2.13}$$

$$\underset{\sim}{\boldsymbol{P}_c} = \underset{\sim}{\boldsymbol{A}}\underset{\sim}{\boldsymbol{M}_c}\underset{\sim}{\boldsymbol{A}}^T + \underset{\sim}{\boldsymbol{B}}\underset{\sim}{\boldsymbol{Q}}\underset{\sim}{\boldsymbol{B}}^T \tag{3.2.14}$$

The consensus gain scale factor ϵ is still subject to the same constraints discussed in Section 3.1 and Equations 3.1.8. Choosing ϵ too large amplifies the consensus term, overpowering the effect of the Kalman filter, and provides a consensus filter tracking a nonsense value. Choosing ϵ too small will cause a very large convergence time for the consensus filter, if it converges at all.

3.3 The Extended Kalman-Consensus Filter

The derivation leads to the following algorithm to be run by every node in the network:

1. Take available local measurements, $\boldsymbol{z}_i$
2. Calculate $\boldsymbol{u}_i$ (Equation 3.2.2) and $\underset{\sim}{\boldsymbol{U}}_i$ (Equation 3.2.1)
3. Transmit $\boldsymbol{u}_i$, $\underset{\sim}{\boldsymbol{U}}_i$, and $\bar{\boldsymbol{x}}_i$ to neighbor nodes
4. Receive $\boldsymbol{u}_i$, $\underset{\sim}{\boldsymbol{U}}_i$, and $\bar{\boldsymbol{x}}_i$ from all neighbor nodes
5. Calculate $\boldsymbol{g}_c$ (Equation 3.2.10)
6. Calculate update and predict stages via Equations 3.2.11 through 3.2.14

The state, x, being tracked is the state of the tracking point on the object. Each node uses the projection techniques discussed in Section 2.5 to estimate the state of the tracking point given measurements at its own location. The result is that each node is estimating the state of the same point, which makes it possible to perform a consensus operation on the state vector.

Chapter 4

SIMULATION

This chapter describes the simulation developed for testing the proposed extended Kalman-Consensus filter. The architecture of the program is discussed, followed by a description of the test case developed for comparing the proposed algorithm to other estimation techniques.

Taking advantage of measurement diversity (multiple measurements of the same quantity) should increase the accuracy of the overall state estimate.

4.1 Sensor Networks

Consider a set of sense nodes distributed at various well-known locations across an object being tracked. Each sense node runs its own extended Kalman filter to track the state of the tracking point on the object. There are a few different ways for the sense nodes to work together to come up with a more accurate estimate. One way involves taking in to account the node's own observations as well as those of any nodes it communicates with (the distributed extended Kalman filter). The proposed extended Kalman-Consensus filter expands on this by also sharing state estimates between neighbor nodes, with the intention that the state estimates of all nodes will converge to a single, more accurate state estimate.

While actual sensors (e.g. accelerometers, gyroscopes) aren't mentioned, the measurements considered are limited to those that are physically possible. To illustrate the effectiveness of the extended Kalman-Consensus filter, arrangements of accelerometers, gyroscopes, and absolute position sensors (e.g. GPS) are examined.

4.1.1 Capabilities

Depending on the hardware configuration, a given sense node is usually only capable of measuring a small subset of the information about the state of the process being tracked. Three possible node types are outlined in Table 4.1.

Node type	Axes	Sensor type
0	x, y, z	acceleration
	x, y, z	angular velocity
1	x, y, z	position
2	x, y	acceleration

Table 4.1. Example sense node capabilities

Each sense node contributes what it can to a state estimate, but that contribution will be different for different sense node types. As a result, the dimensions of the vectors and matrices behind the filtering will change depending on what is being observed, but the dimensions of the state vector will remain the same as long as the dimensions follow the pattern established in Table 2.6. This becomes important when nodes share measurements between one another.

4.1.2 Communication

Communication between sense nodes is necessary to form an overall estimate of object state that accounts for the various distributed measurements. Intuitively, the best performance will be achieved when the communication graph is complete (all nodes can talk to all nodes), but this architecture suffers from a lack of real-world robustness, and communication complexity that increases as $\boldsymbol{O}(n^2)$ with the number of nodes n.

A more feasible architecture is a relatively sparse communication graph, where nodes communicate with their local neighbors. Making the communication graph dynamic would make the system more robust.

In the simulations, communication between nodes occurs in both directions, but there is no reason that the communication can't be one-way. For example, a large

network could have smart nodes (sensors and filtering calculations) that take care of their local neighborhood of dumb sensors (sensors only, no calculations), and communicate with other smart nodes in to execute the distributed filtering algorithm. This scenario is not explored any further in this work.

4.2 Simulation Architecture

In order to test the proposed algorithm, a simulation environment was developed in MATLAB, using object-oriented design principles. A diagram of the simulation architecture is shown in Figure 4.1. The arrows represent information being passed to the pointed-at class. The red lines indicate information requested by the judge, blue lines correspond to information requested by the sensors, and green lines are the inter-node communications.

Descriptions of the important classes follow:

Node Each node contains a filter (or filters) and a set of sensors, defined by a sensor mask indicating which quantities the sensor is capable of sensing. This makes it easy to investigate the effects of different sensor combinations. The node also knows its location and orientation relative to the tracking point, which is necessary for filter calculations.

Filter The filter class implements the various Kalman filter types discussed in this work. After a node has interrogated its sensors, it passes that information on so the filter can make its next calculation.

World The world class is responsible for receiving queries from the sensors, and replying with the values that the sensor should be reading to follow some pre-determined path, taking in to account the location of the sensor on the object.

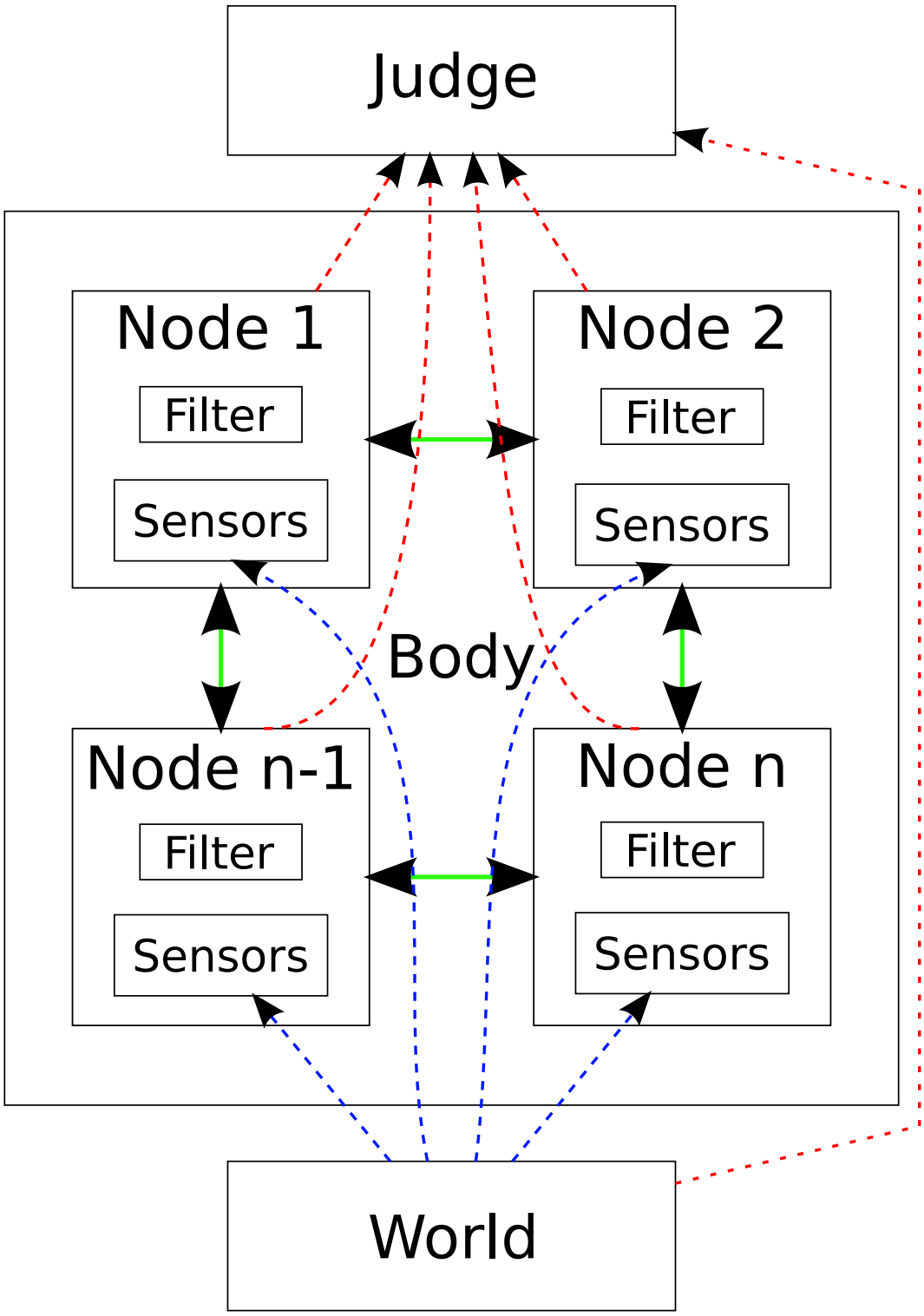

Figure 4.1. Simulation architecture

Judge The judge checks in with all the nodes and the world every iteration, and keeps track of the true and estimated states. The judge then plots this data for later analysis.

4.3 Kalman Filter Design

The design parameters of the Kalman filter were chosen to fit a general case of three dimensional motion, and were not specialized to any specific type of motion. The matrices are formed from the general kinematic equations given in 2.2.1.

The process matrix is an 18-by-18 element matrix that describes how the state of the process evolves from one time step to the next. The complete set of rows in the process matrix are given in Equations 4.3.1 through 4.3.6.

$$\underset{\sim}{\boldsymbol{A}}_{1:3} = \begin{bmatrix} 1 & 0 & 0 & dt & 0 & 0 & .5dt^2 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & dt & 0 & 0 & .5dt^2 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & 0 & dt & 0 & 0 & .5dt^2 & 0 & \cdots & 0 \end{bmatrix} \tag{4.3.1}$$

$$\underset{\sim}{\boldsymbol{A}}_{4:6} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & dt & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & dt & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & dt & 0 & \cdots & 0 \end{bmatrix} \tag{4.3.2}$$

$$\underset{\sim}{\boldsymbol{A}}_{7:9} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & a_1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & a_1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & a_1 & 0 & \cdots & 0 \end{bmatrix} \tag{4.3.3}$$

$$\underset{\sim}{\boldsymbol{A}}_{10:12} = \begin{bmatrix} 0 & \cdots & 0 & 1 & 0 & 0 & dt & 0 & 0 & .5dt^2 & 0 & 0 \\ 0 & \cdots & 0 & 0 & 1 & 0 & 0 & dt & 0 & 0 & .5dt^2 & 0 \\ 0 & \cdots & 0 & 0 & 0 & 1 & 0 & 0 & dt & 0 & 0 & .5dt^2 \end{bmatrix} \tag{4.3.4}$$

$$\underset{\sim}{\boldsymbol{A}}_{13:15} = \begin{bmatrix} 0 & \cdots & 0 & 0 & 0 & 0 & 1 & 0 & 0 & dt & 0 & 0 \\ 0 & \cdots & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & dt & 0 \\ 0 & \cdots & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & dt \end{bmatrix} \tag{4.3.5}$$

$$\underset{\sim}{\boldsymbol{A}}_{16:18} = \begin{bmatrix} 0 & \cdots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & a_2 & 0 & 0 \\ 0 & \cdots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & a_2 & 0 \\ 0 & \cdots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & a_2 \end{bmatrix} \tag{4.3.6}$$

The time step dt was chosen to be 1 for the simulations. The variables a_1 and a_2 determine how previous accelerations are weighted when considering the next acceleration. a_1 and a_2 are set so that an acceleration's impact has faded to 10% of it's original magnitude 4 time steps after it is introduced; the math is expressed in Equation 4.3.7.

$$a^4 = .1 \tag{4.3.7}$$

$$a_2 = a_1 = .5623 \tag{4.3.8}$$

Each row in the process matrix corresponds to an element in the state vector, and each column represents the contribution of that element in the state vector to the resulting state vector. For example, in Equation 4.3.1, the element at row 1, column 9 indicates that one component of the next state vector's position comes from the current state vector's acceleration multiplied by $.5dt^2$; this is expressed in Equation 4.3.9.

$$x_1^+ = x_1 + (dt)x_4 + (.5dt^2)x_7 \tag{4.3.9}$$

The process noise matrix was set to the value shown in Equation 4.3.10. The actual process noise matrix used in each predict step will vary in dimensions, based on the dimensions of the observation, which is determined by the node type. When preparing to do the predict calculation, the appropriate rows (i.e. those that correspond to the measurements being observed) from the matrix in Equation 4.3.10 are combined to form the actual $\underset{\sim}{\boldsymbol{B}}$ matrix used in the calculation of $\underset{\sim}{\boldsymbol{P}}$.

$$\begin{aligned}\underset{\sim}{\boldsymbol{B}} = diag(&b_1, b_1, b_1, 0, 0, 0, b_1, b_1, b_1, ... \\ &0, 0, 0, 0, 0, 0, b_2, b_2, b_2)\end{aligned} \tag{4.3.10}$$

The variables b_1 and b_2 combined with a_1 and a_2 shape a filter that controls the effect of the process noise on the evolving state. b_1 and b_2 are chosen to set the variance of the process noise to 10^{-4}, given the a_1 and a_2 chosen above, and white process noise. This is shown in Equation 4.3.11.

$$\begin{aligned} \sigma^2 &= \frac{b^2\sigma_n^2}{1-a^2} \\ \sigma^2 &= 10^{-4} \\ \sigma_n^2 &= 1 \\ b_2 &= b_1 = .0083 \end{aligned} \tag{4.3.11}$$

The observation noise associated with the sensors was picked as in Equation 4.3.12. In an actual application, these values would be determined by the specs of the actual sensors being used.

$$\begin{aligned} \underset{\sim}{\boldsymbol{R}} = diag(&10^{-8}, 10^{-8}, 10^{-8}, 10^{-4}, 10^{-4}, 10^{-4}, 10^{-4}, 10^{-4}, 10^{-4}, ... \\ &10^{-4}, 10^{-4}, 10^{-4}, 10^{-4}, 10^{-4}, 10^{-4}, 10^{-4}, 10^{-4}, 10^{-4}) \end{aligned} \tag{4.3.12}$$

4.4 Test Case

A test case was developed to compare the performance of the various filtering algorithms discussed in this work.

4.4.1 Sense Node Layout

Two different sense node formations will be considered in the following examples. In the formation figures, the boxes represent sense nodes, the lines are communication links, and the circle is the tracking point. Each sense node is labelled with an identifying number, position on the object ((x, y, z) relative to the tracking point), and sensing type.

The formation in Figure 4.2 is in the form of a pentagram, with each sense node communicating with it's adjacent sense nodes around the perimeter of the pentagram. Each node is separated by at most 3 hops, and information propagates in both directions around the perimeter of the pentagram.

The second formation, in Figure 4.3 is a simple line, with each sense node communicating with only adjacent sense nodes. In this case, nodes 1 and 5 are separated by 4 hops, and there is only one propagation path for messages.

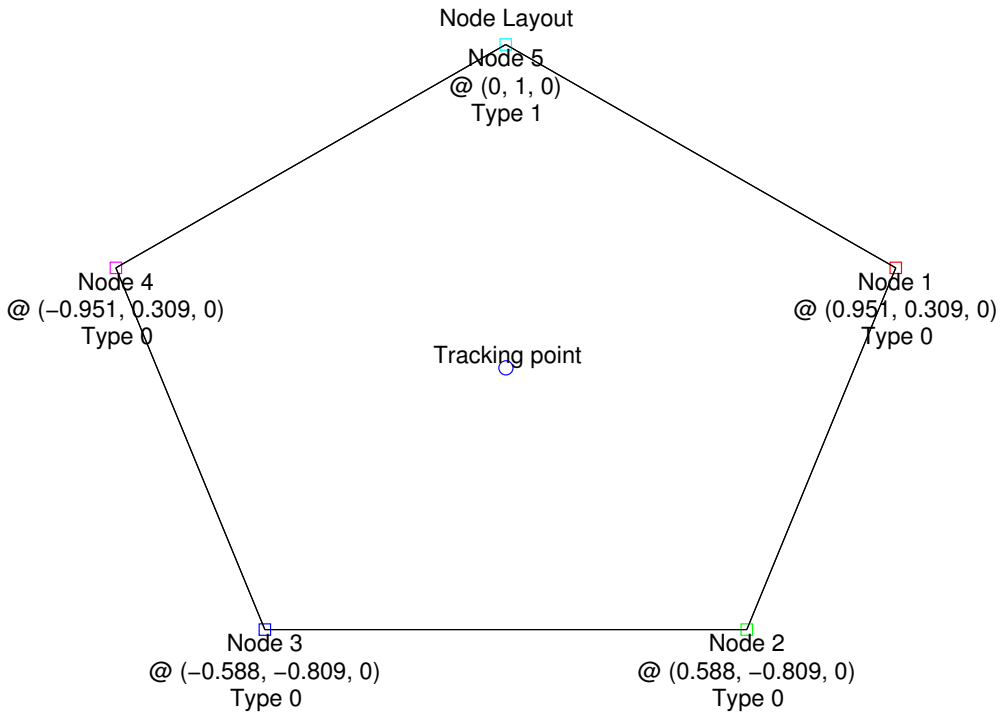

Figure 4.2. Pentagram sense node formation

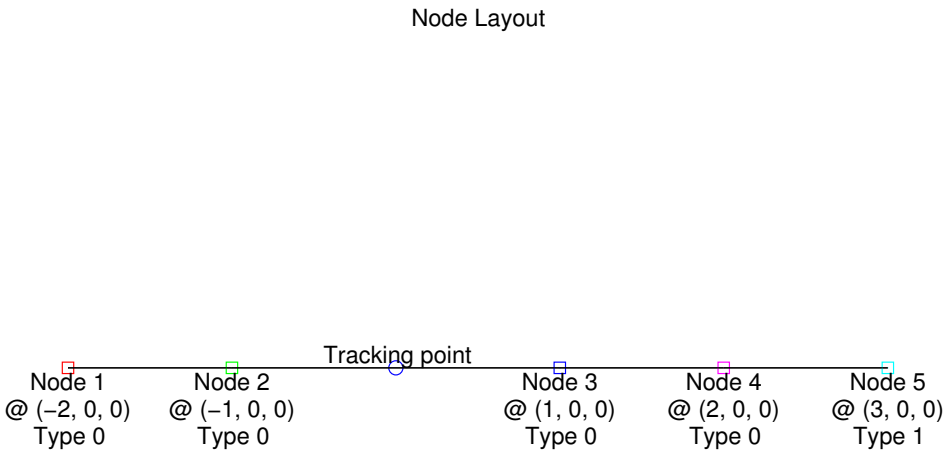

Figure 4.3. Line sense node formation

4.4.2 Object Motion

In order to evaluate the performance of the proposed extended Kalman-Consensus filter and compare it to other methods, it became necessary to generate a well-defined, predictable behavior for the object's movement.

It's difficult to think about defining a path (i.e. set of positions) by picking the correct accelerations at each time step, and even more difficult to pick the corresponding rotation rates to complete the puzzle. So the problem was approached from the opposite direction. An equation, Equation 4.4.1, was chosen for the position of the object $\boldsymbol{p}$ as a function of time and a constant rotational rate ω, Equation 4.4.2.

$$\boldsymbol{p} \begin{cases} p_x = .05 * 2 * \pi * t \\ p_y = 1 - \cos(p_x * t) \\ p_z = 0 \end{cases} \tag{4.4.1}$$

$$\omega = \frac{.05 * 2 * \pi}{4} \tag{4.4.2}$$

The resulting path is shown in Figure 4.4. The plot shows the position of the tracking point on the object (as referenced in the legend), and a vector (terminated by an asterisk) that indicates the orientation of the object.

It is important that the chosen equations are twice differentiable, so the exact acceleration values can be easily generated.

Every iteration of the simulation, each node's sensors are interrogated, and the world uses the above formulas to determine what each node should be sensing, taking in to consideration the node's position as well as the orientation of the object.

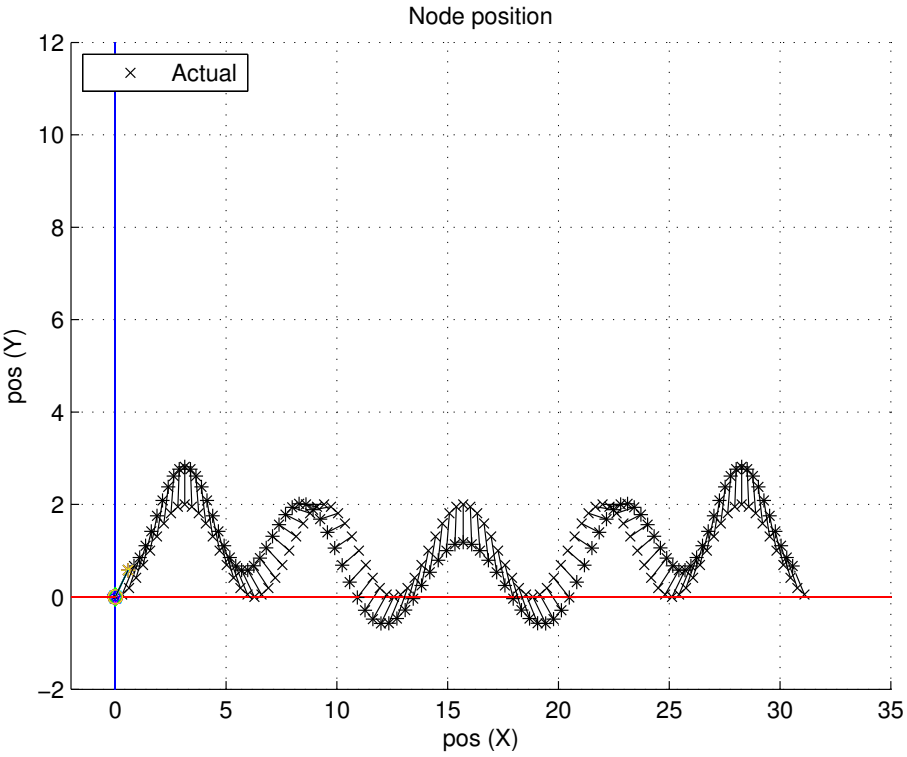

Figure 4.4. Object motion

Chapter 5

ANALYSIS

The performance of the various Kalman filters investigated are compared in this chapter.

In each case, all filters are initialized with the correct starting state.

In the following position plots, each node's estimate of the position of the tracking point on the object is shown by the corresponding symbol in the plot's legend. There is a vector (terminated by an asterisk) attached to each symbol which indicates the estimate of the orientation of the object.

The position error plots show the difference between the estimated and actual positions of the tracking point for each node. The x, y, and z components are displayed separately.

5.1 Isolated Extended Kalman Filters

The simplest case, where there is no communication between sense nodes, provides a lower bound for performance of a distributed method for tracking an object. This produces a separate estimate of the object state for each sense node.

For each sensor formation (e.g. pentagram, line), each sensor's estimate of the object's position and orientation are plotted, and accompanied by a separate plot of the error in the position estimate (i.e. difference between estimate and actual position).

Figures 5.1 and 5.2 demonstrate how the pure inertial nodes (type 0) capture the shape of the motion well, but build up error constantly due to the error in integration, which is unavoidable. This behavior can't be fixed by simply allowing inertial nodes to communicate; there needs to be another reference to pull the state estimate back towards the actual state. A type 1 sensor (e.g. GPS) is perfect for this.

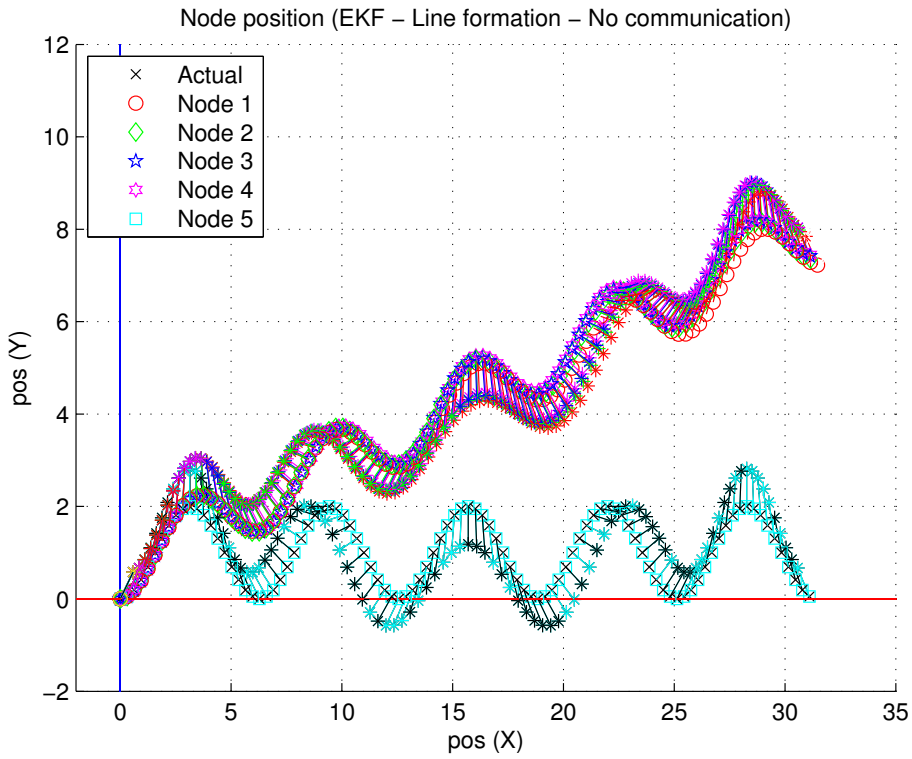

Figure 5.1. Line EKF plot

The type 1 sensor does a remarkably good job at tracking the object's position by itself, but this is due to the update rate in the simulation. In reality, GPS updates are not available at a rate that is fast enough for scenarios like precision approaches, so integration with inertial sensors is common, due to their much faster rate of measurement. As more sensors are fused in a distributed fashion, the overall measurement system can do a progressively better job.

Even though there is no communication between nodes, there is a slight difference in the behavior of the different sensor formations. This is due to the location of each sensor on the object; different sensor locations induce slight differences in the initial error. This error grows in the same way for each sensor, but the initial difference persists.

The error plots in Figures 5.3 and 5.4 are also virtually identical. This positively suggests that the location of the sensors doesn't affect the position estimate, other than the initial error in the inertial nodes discussed previously.

For most sense node types, an isolated extended Kalman filter will quickly diverge from the actual state of the object, if it is able to track at all. In general, different node types will encounter their own unique problems in this case:

Node type 0

Errors from integrating various quantities will build up, causing estimated state to diverge from actual state.

Node type 1

Real-world position updates are relatively slow, and orientation isn't tracked, so complete three dimensional tracking isn't feasible.

Node type 2

Measurements are only taken in two dimensions, so there is no way to track position in three dimensional space. Also, orientation isn't tracked.

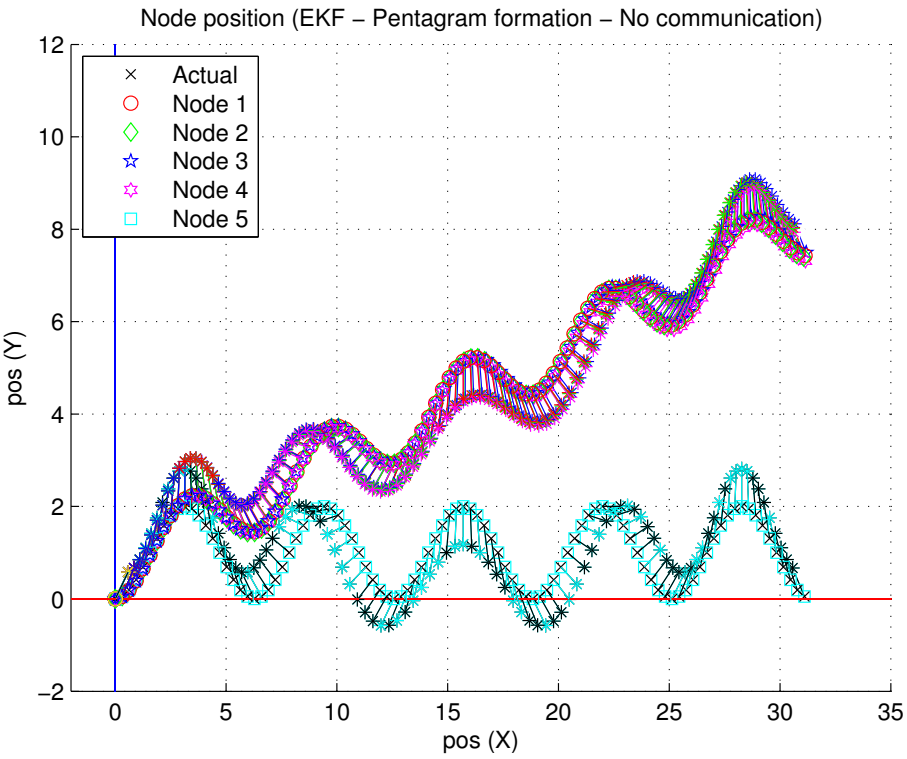

Figure 5.2. Pentagram EKF plot

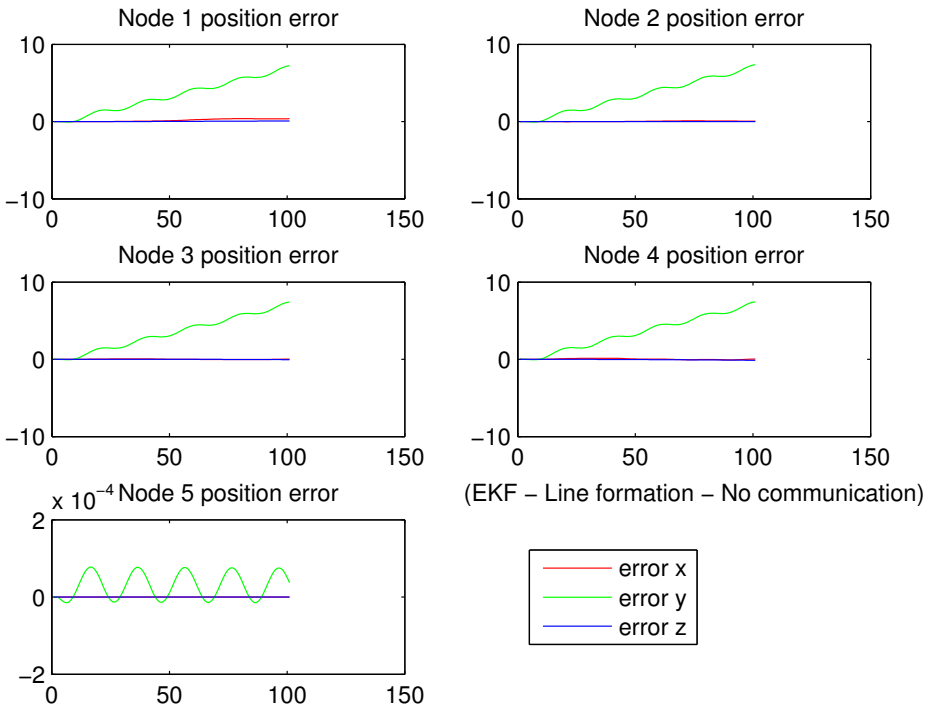

Figure 5.3. Line EKF error plot

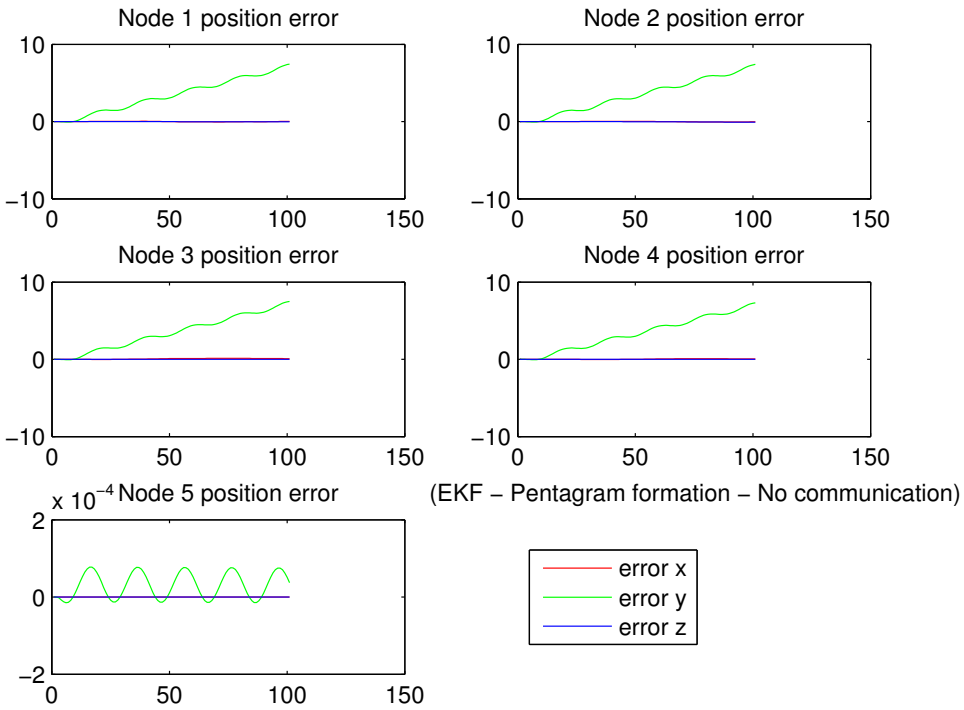

Figure 5.4. Pentagram EKF error plot

Node type 0+1

This node type will have the best performance, but that performance can be improved upon by incorporating measurement diversity from multiple nodes.

5.2 Central Extended Kalman Filter

In the case of a complete communication graph, each node has access to all observations made throughout the network, and will use those observations to calculate an estimate of the object's state. This is called a central extended Kalman filter, and should provide an upper bound for performance of a distributed method for tracking an object.

Plots of the position estimate for each node, as well as the corresponding error plots are presented in the following figures. Each node calculates a state estimate using the same observations, so the node position points are plotted very near one another.

The position estimates plotted in Figure 5.5 (line formation), and Figure 5.6 (pentagram formation) are almost indistinguishable, which suggests that the sensor formation has little effect on the overall system's state estimate. This is encouraging, as it suggests that the sensor formation is flexible provided the right types of sensors are mixed in. Still, more work could be done to investigate the limitations that will likely be imposed by using less capable sensors in more ambitious configurations (e.g. single-axis sensors communicating solely with other single-axis sensors).

There is a slight difference in the quality of the estimates of the different formations; this shows up in the following error plots. At this magnitude of error, the shape of the error curves is significantly affected by the random error introduced by taking measurements.

The pentagram formation error in Figure 5.8 has similar, small magnitude to that of the line formation in Figure 5.7. This suggests that the error is mostly attributable to the noise introduced in the observations, and not by the sensor formation. As can be

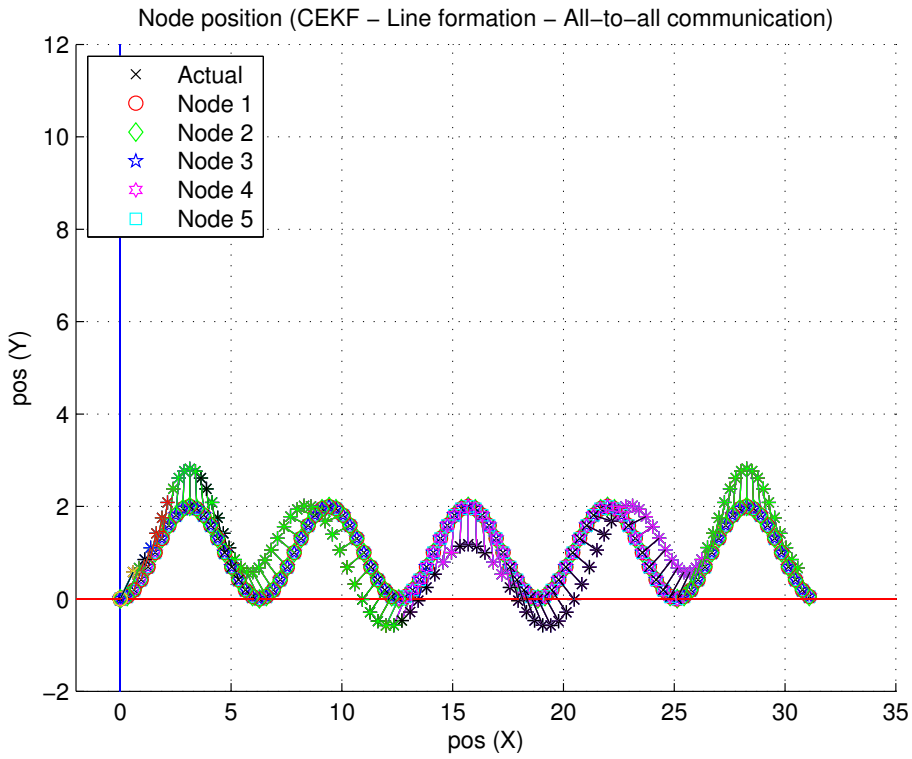

Figure 5.5. Line CEKF plot

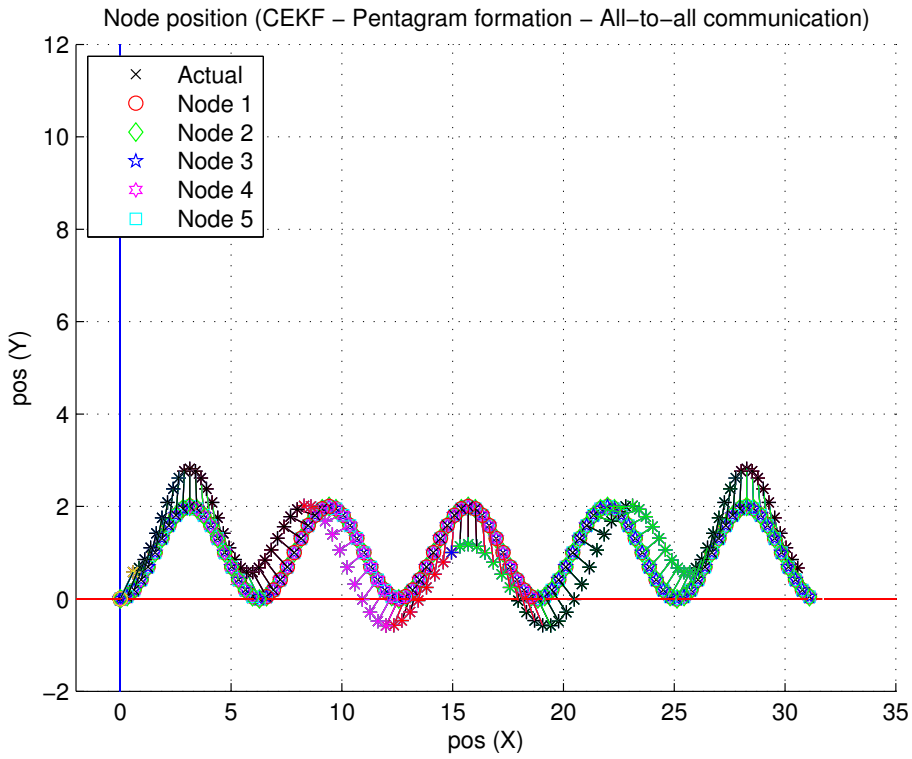

Figure 5.6. Pentagram CEKF plot

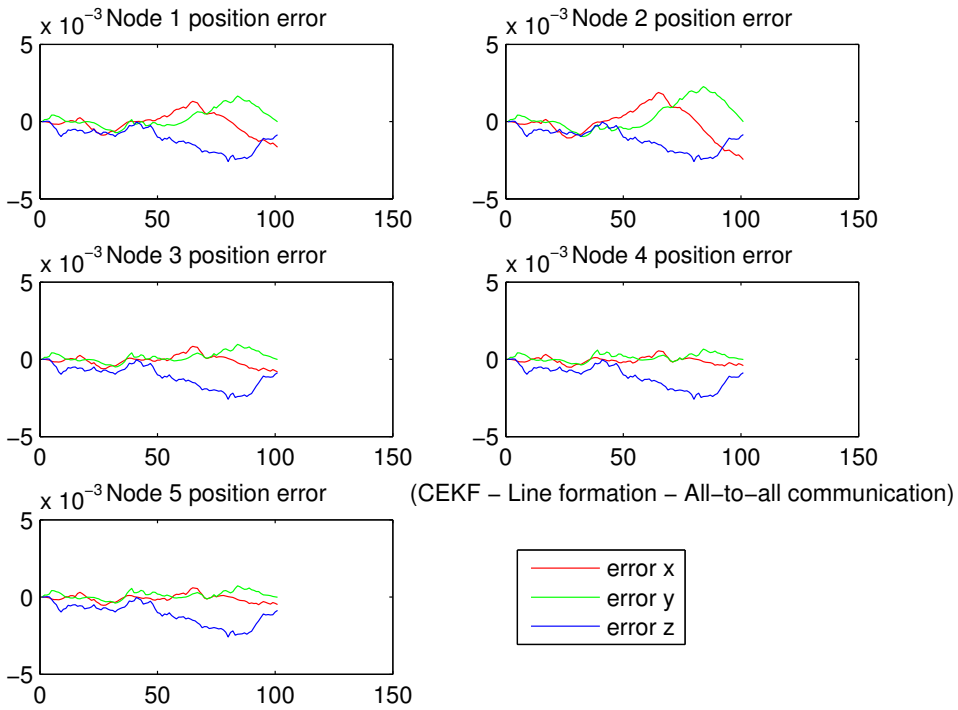

Figure 5.7. Line CEKF error plot

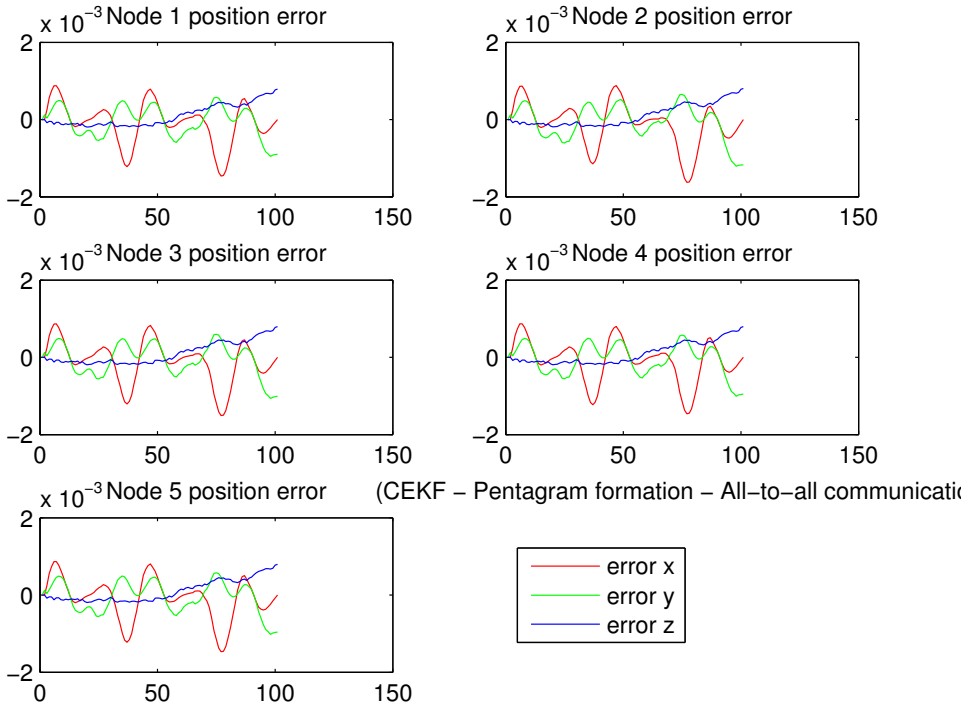

Figure 5.8. Pentagram CEKF error plot

expected of a central extended Kalman filter – where each node observations from all nodes in it's calculations – the error characteristics of each node are extremely similar to one another, within a given formation.

5.3 Distributed Extended Kalman Filter

Consider an incomplete communication graph where each node only communicates with a few of its neighbors. In this case, the distributed extended Kalman filter can be applied to allow the sharing of observations between neighboring nodes. This scenario allows for more efficient scaling with an increasing number of sense nodes, as opposed to the all-to-all communication of the central extended Kalman filter.

An issue with the distributed extended Kalman filter becomes apparent in the line formation position plot, Figure 5.9. Not long in to the simulation, the position sensor (node 5) becomes corrupted, estimating the state very poorly. The behavior is a result of node 5 not making any angular measurements. The only angular information node 5 has is from its initialized state, and the angular rate measurements from node 4's messages. Any slight error from node 4's measurements causes an error in the estimated orientation of node 5, and before long, the estimated orientation is too poor to provide a good linearization when calculating $\boldsymbol{h}(\boldsymbol{x})$. This effect feeds itself, until the state estimate becomes completely corrupted.

The issue doesn't manifest in the pentagram formation, Figure 5.10, presumably due to the position sensor receiving angular information from more than one source. It is unclear how fragile this issue is, but it likely depends on the actual application. In order to be reliable, the distributed extended Kalman filter would need careful sensor layout to prevent the corruption of more fragile sensor types; this also makes the filter less robust against losing sensor nodes.

The error of each node in the line formation, Figure 5.11, demonstrates how the limited communication causes issues in non-favorable formations. The nodes farther

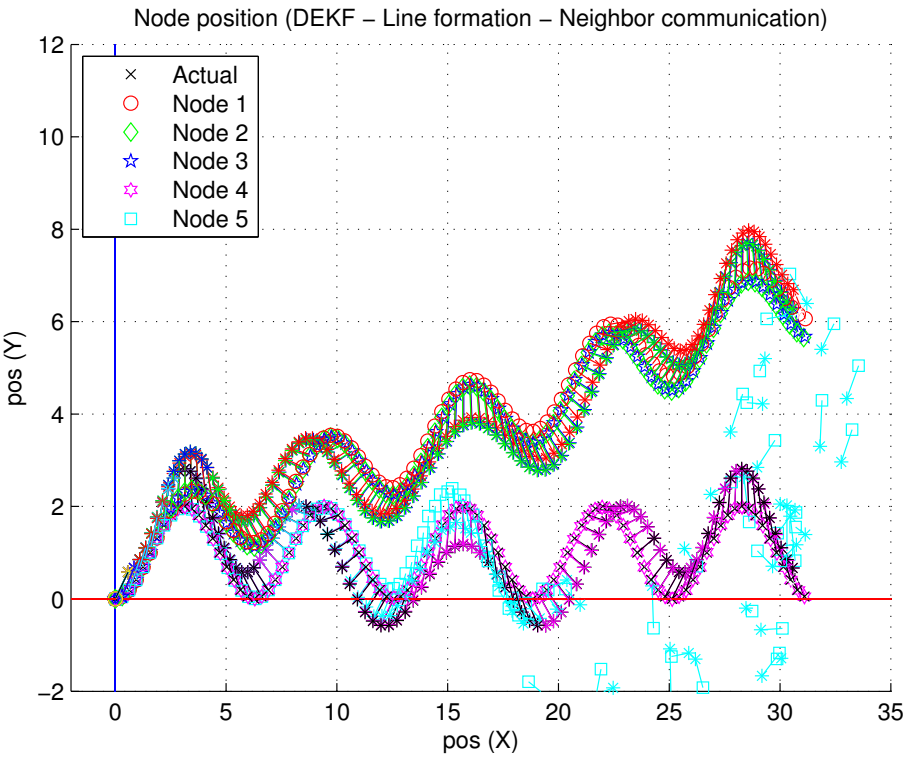

Figure 5.9. Line DEKF plot

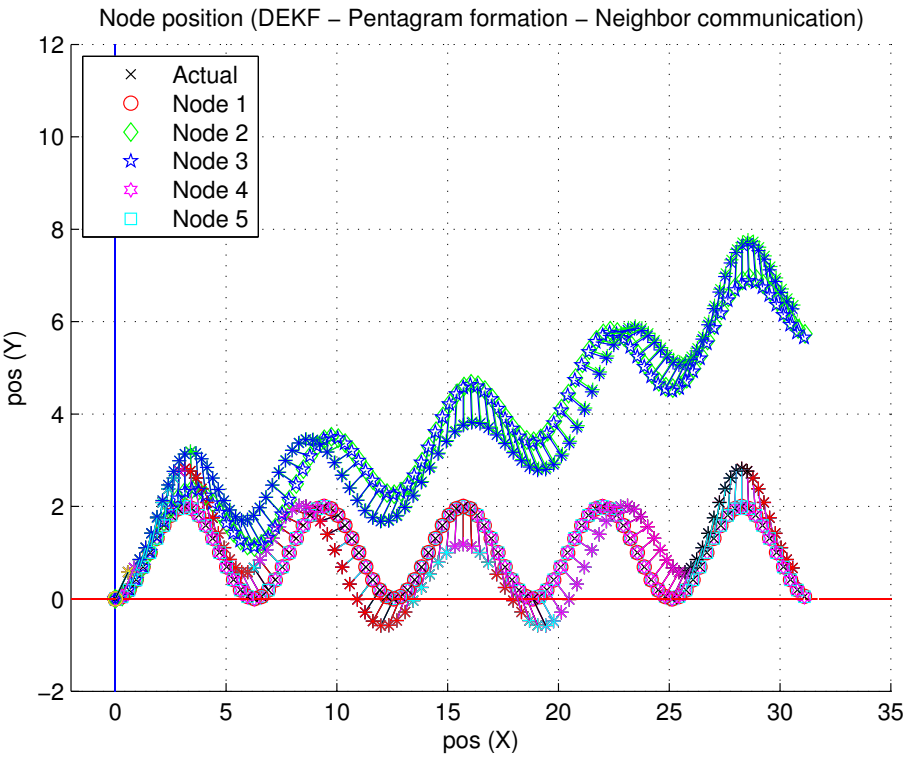

Figure 5.10. Pentagram DEKF plot

away from the position sensor, don't benefit at all from the position measurements, making them only slightly more effective than isolated extended Kalman filters. Node 4 begins by tracking a similar error curve to that of node 5, but doesn't become corrupted when node 5 does. Node 4 keeps track of its own orientation, which is influenced by angular measurements from node 4 and node 3, so it is still able to linearize and make use of the position observations from node 5 (node 5 is still making good observations, despite having a bad orientation estimate).

The pentagram formation displays less error, Figure 5.12, but the same behavior; the nodes farther away from the position sensor don't receive any benefit from the position sensor. Positive behavior is shown in the nodes adjacent to the position sensor; nodes 1 and 4 take on an error characteristic very similar to that of the position sensor.

The distributed extended Kalman filter shows improvement in scalibility, and promise in sharing observations between nodes, but comes up short as a flexible and robust solution for distributed non-linear estimation.

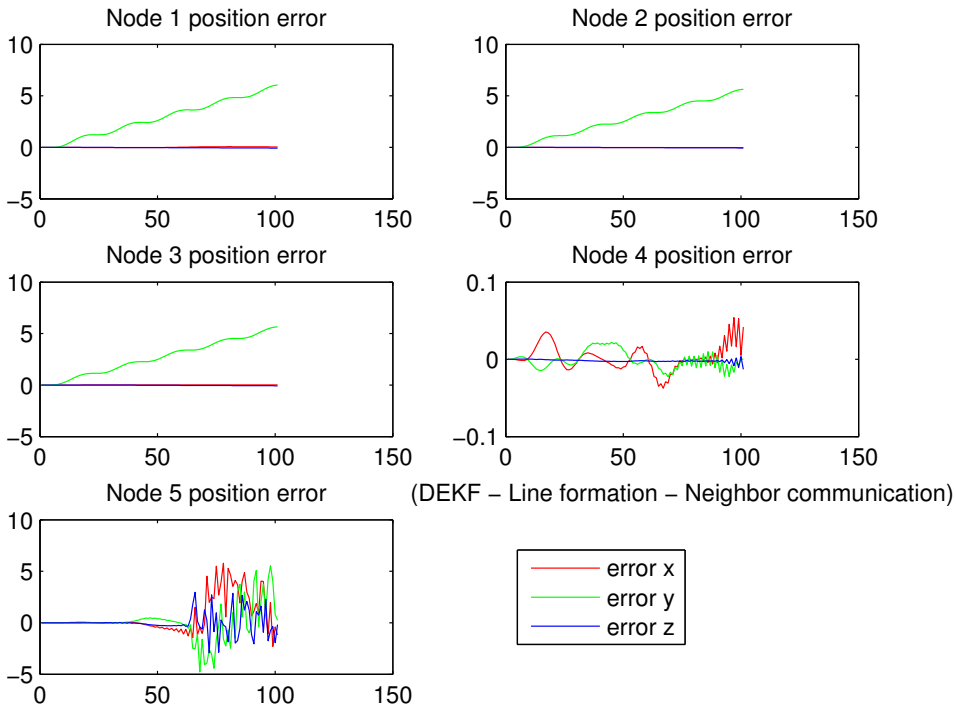

Figure 5.11. Line DEKF error plot

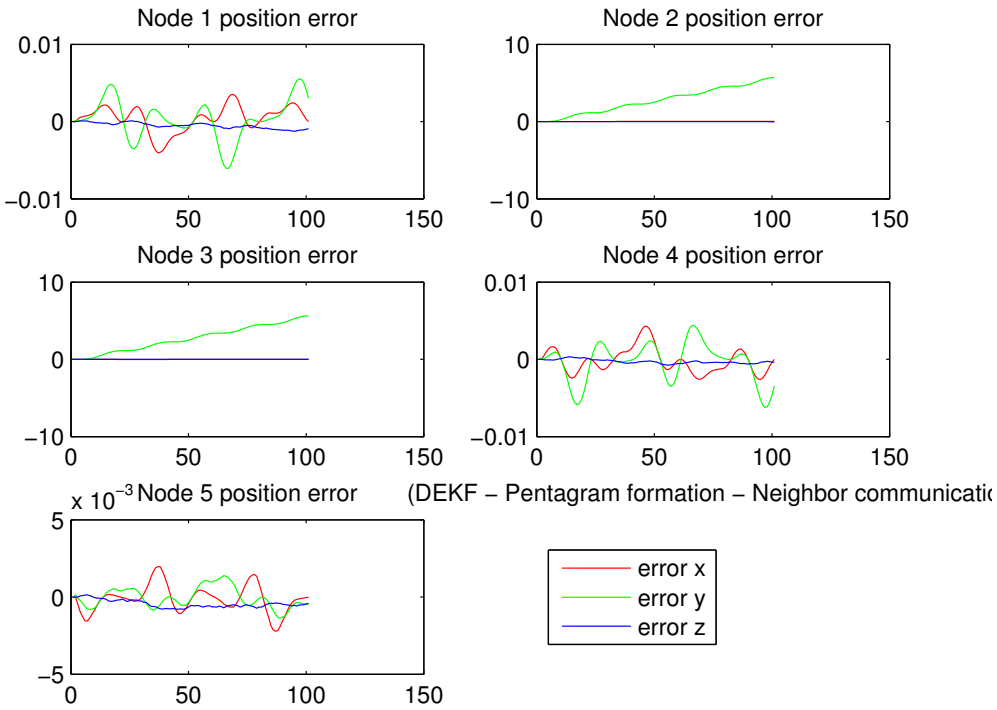

Figure 5.12. Pentagram DEKF error plot

5.4 Extended Kalman-Consensus Filter

The proposed extended Kalman-Consensus filter builds on the scalability and local communication of the distributed extended Kalman filter by adding a consensus on the state estimate. In the following simulations, the value of the consensus gain scale factor ϵ was chosen to be:

$$\epsilon = .8$$

This value was chosen according to the guidelines in Sections 3.1 and 3.2, and tweaked to create a reasonable convergence time in the test case. The behavior of the line formation in Figure 5.13 is encouraging; each sensor's estimates begin following their usual track, but are quickly pulled together by the consensus.

The pentagram formation performs in a similar fashion in Figure 5.14, but exhibits a quicker convergence time. This suggests that some sensor formations will perform more favorably than others. Specifically, the closed loop of the pentagram formation allows quicker propagation of the state between the nodes, as opposed to the open-ended line formation.

The error plot of the line formation in Figure 5.15 reveals more details which further support the characterization of the proposed filter. The nodes farther away from the position sensor take longer to converge to the better state estimate. In each of the nodes from node 1, to node 2, to node 3, the shape of the error curve becomes more like that of the least-error curve of node 5, suggesting that the nodes are converging properly to an estimate with as little error as possible.

The pentagram formation error plot in Figure 5.16 exhibits the same characteristics. The nodes farthest away from the least-error node converge just a bit more slowly than the nearer nodes, eventually agreeing on a state estimate with low error.

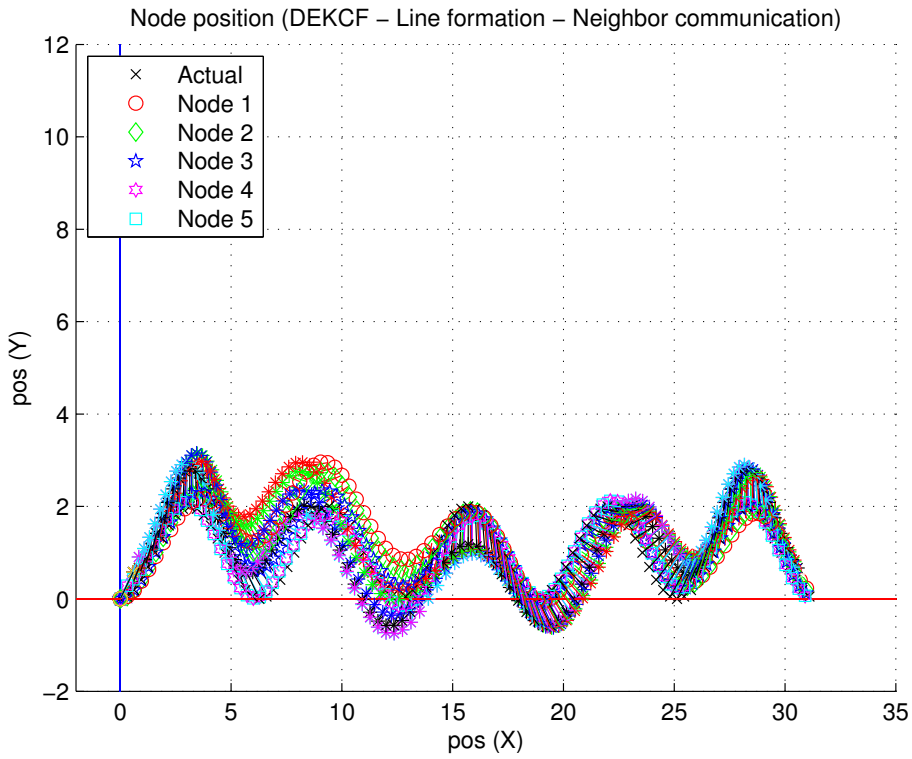

Figure 5.13. Line EKCF plot

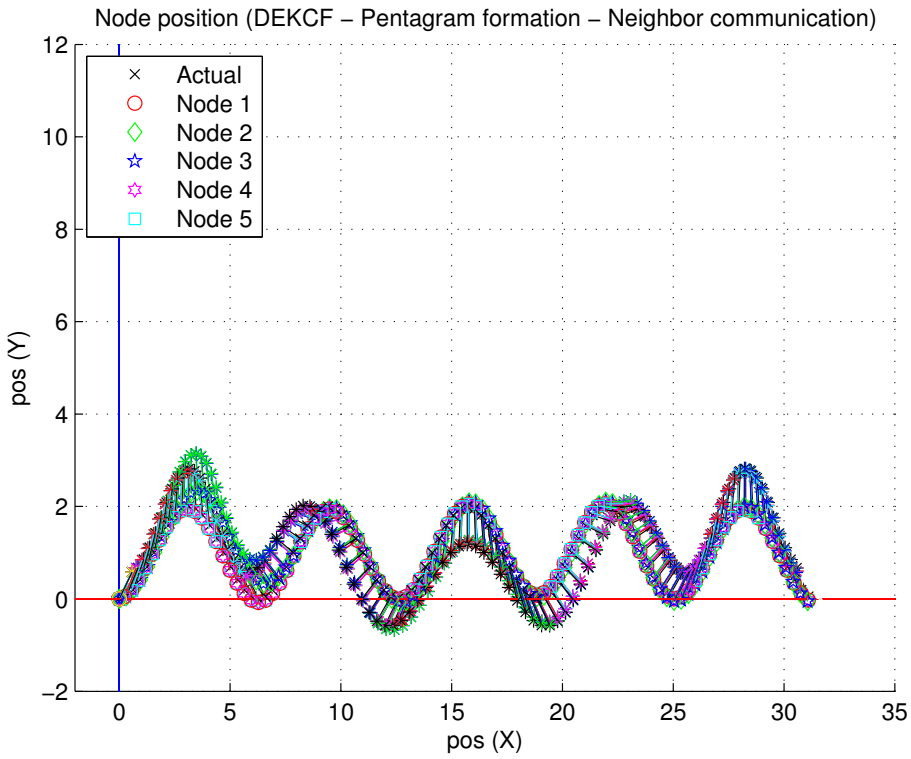

Figure 5.14. Pentagram EKCF plot

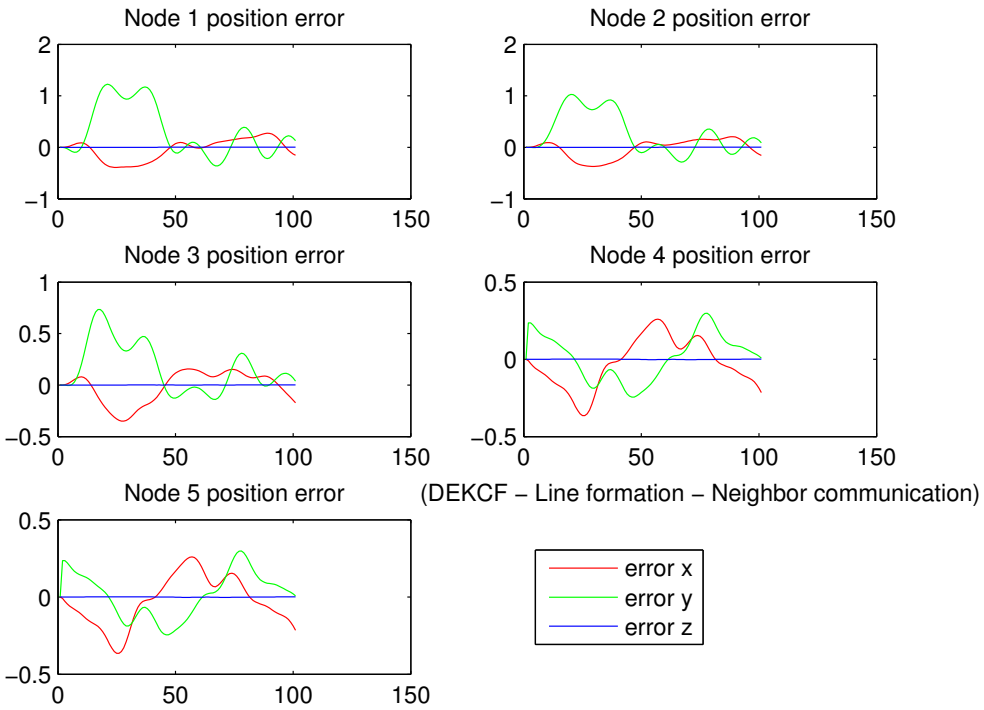

Figure 5.15. Line EKCF error plot

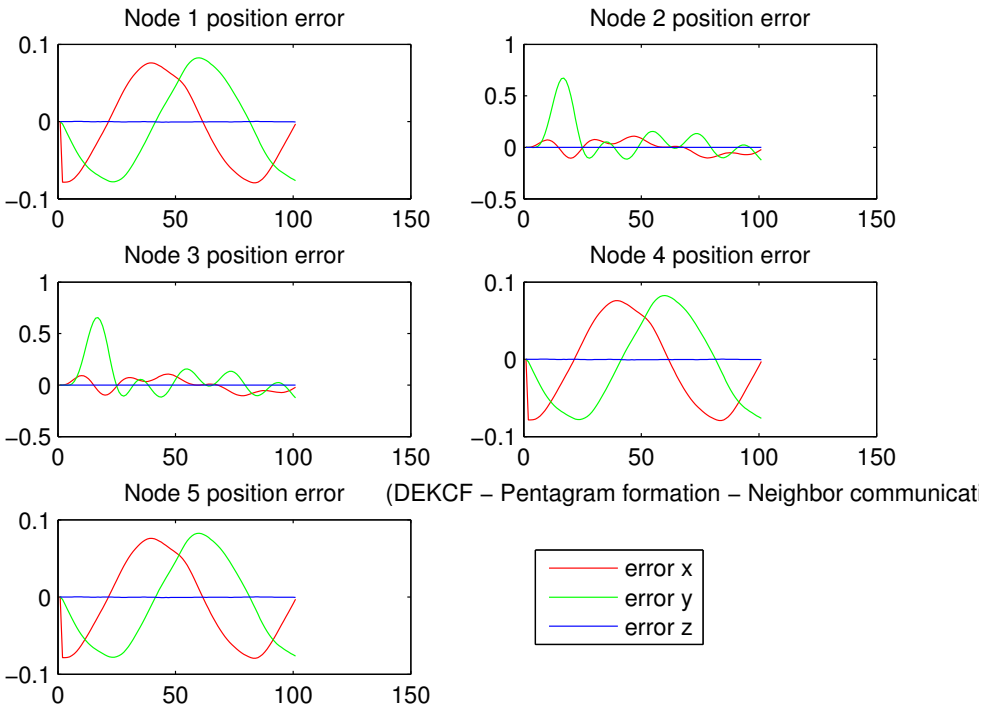

Figure 5.16. Pentagram EKCF error plot

5.4.1 Long Term Behavior

The previous plots have made a strong case for the behavior of the extended Kalman-Consensus filter in terms of the ability to induce convergence between the nodes. In order to compare the error magnitude performance, a longer simulation was run to allow the error to settle. The line formation was put through the same simulation for five times as long, and the resulting position plot is shown in Figure 5.17.

After all node's estimates have converged, they stay locked together with no signs of divergence. The error plot in Figure 5.18 tells the same story, but also shows the error converging to within an envelope of approximately the same magnitude as that of the central extended Kalman filter.

The simulations suggest that the proposed extended Kalman-Consensus filter is capable of achieving estimation accuracy comparable to that of the central extended Kalman filter, while maintaining greater scalability.

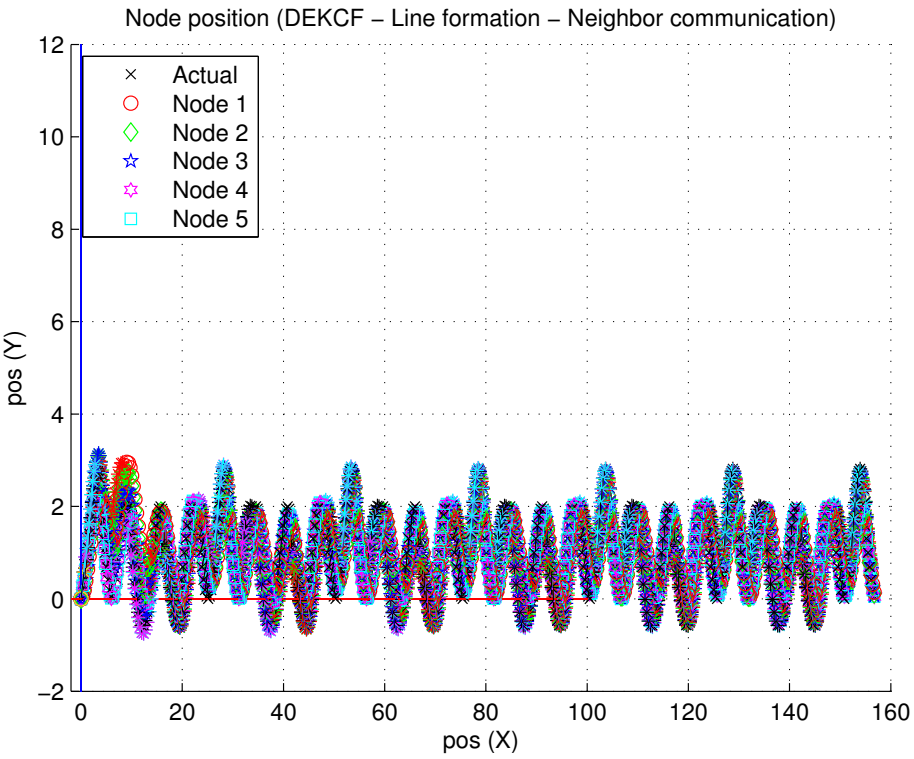

Figure 5.17. Long-term line EKCF plot

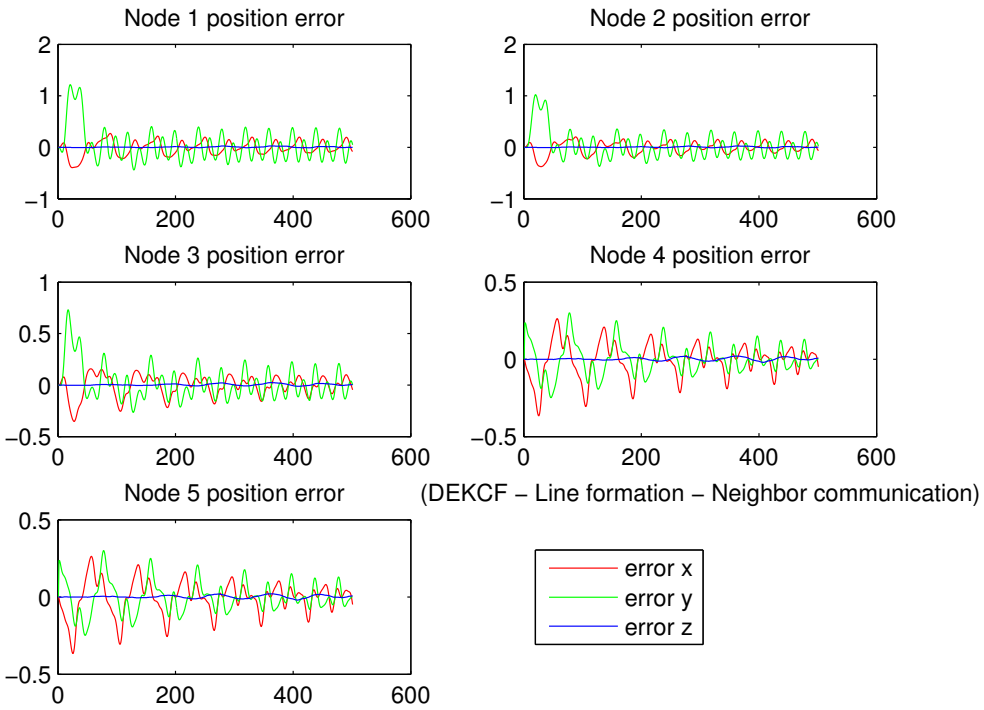

Figure 5.18. Long-term line EKCF error plot

5.4.2 Consensus Gain Behavior

A smaller consensus gain scale factor will cause slower convergence of the state estimates of the nodes to one another. The previous choice of ϵ caused aggresive convergence; now a more relaxed ϵ will be examined:

$$\epsilon = .4$$

This behavior is illustrated in the position plot in Figure 5.19 and the corresponding position error plot, Figure 5.20.

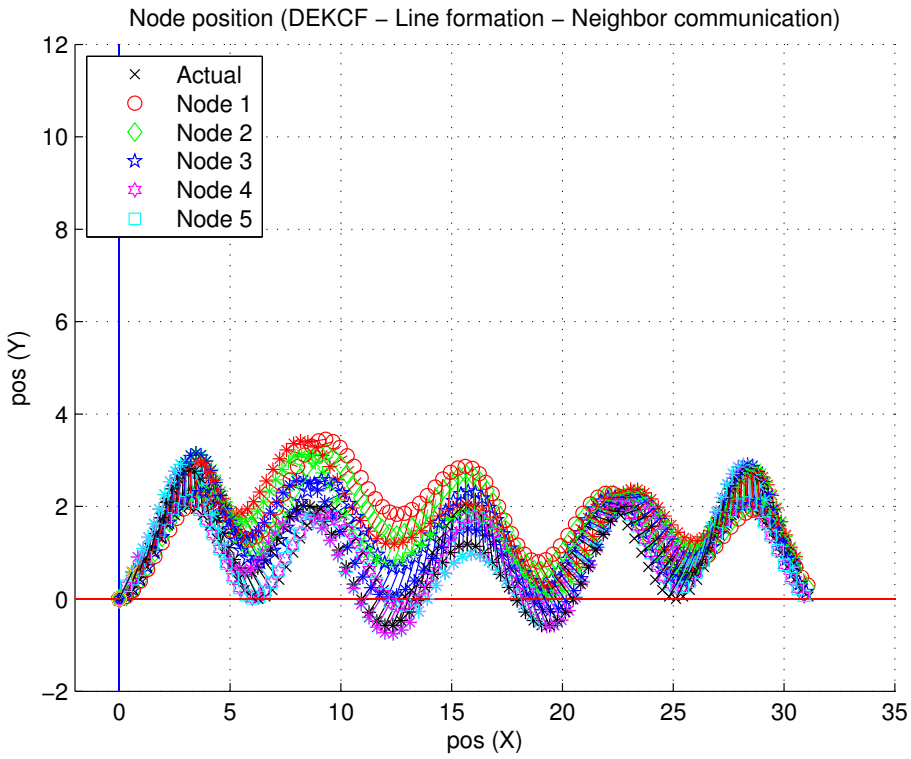

Figure 5.19. Slow consensus line EKCF plot

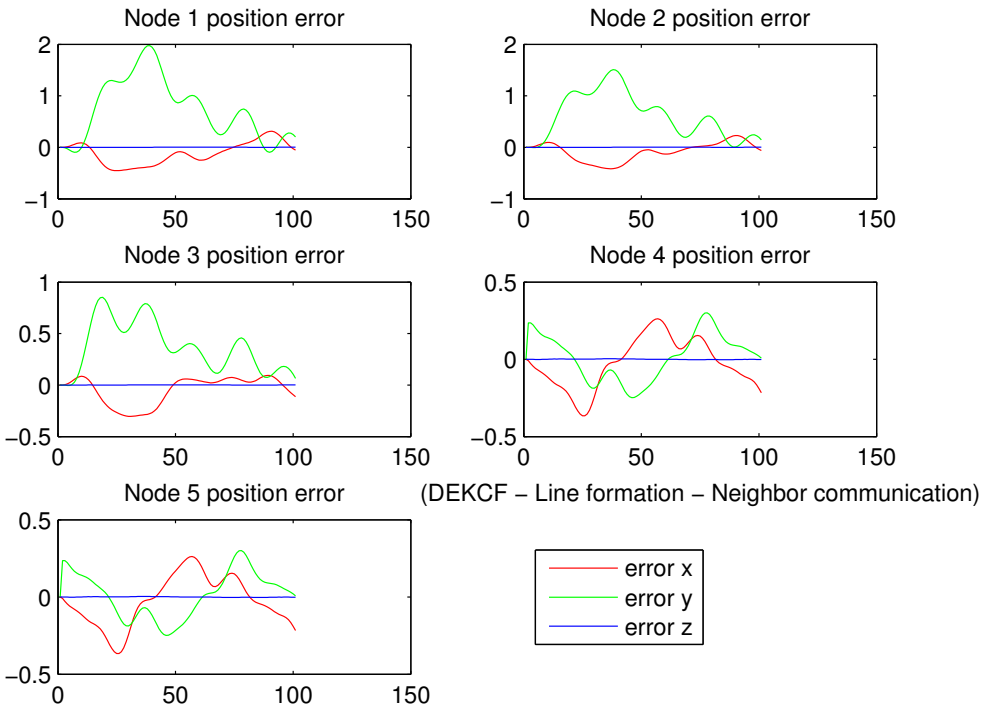

Figure 5.20. Slow consensus line EKCF error plot

Chapter 6

CONCLUSION

6.1 Challenges

6.1.1 Choosing a Linearization Point

Choice of a state estimate to linearize around is probably the most challenging difference to overcome between a linear Kalman filter and a non-linear Kalman filter. As established in Section 2.7, a poor choice of linearization point can easily cause the filter to diverge from the correct state. This problem is amplified, because in the distributed case there are now n choices of linearization points, one for each sense node, and each one is potentially unique. All n filters are attempting to estimate the same actual state, so it would be helpful if some consensus could be reached for a linearization point.

The proposed extended Kalman-Consensus filter is effective in preventing the choice of linearization point from becoming an issue, because as long as the algorithm converges on a state estimate quickly enough (i.e. as long as the filter is well tuned), each node is using nearly identical linearization points.

6.1.2 Ensuring Consensus

In addition to the tuning parameters that accompany any Kalman filter implementation, the extended Kalman-Consensus filter introduces the need to tune the consensus gain scale factor ϵ. While a more in depth discussion of selecting ϵ and convergence is presented in [3], no complete solution is found. Therefore, the choice of ϵ is not straightforward, and will likely require tweaking via a guess and check method to ensure convergence of the filter.

6.1.3 Timing

The timing of communication between the nodes will likely limit the number of nodes that can communicate with each other in a given neighborhood. The latency of the communication network and any associated overhead will place a limit on how quickly nodes can exchange messages, and will be a major consideration when applying the extended Kalman-Consensus filter to a new task.

6.1.4 Initial Conditions

As is the case with any extended Kalman filter, the use of linearization means that good initial conditions can be vital to ensuring the filter is able to converge. This is another application-specific problem, but the problem is widely known, so there is prior work.

6.2 Performance of the extended Kalman-Consensus filter

The position error plots in Chapter 5 demonstrated that the extended Kalman-Consensus filter is able to converge on a state estimate as anticipated, provided the filter is properly tuned. The propogation of the state estimate between the nodes also behaves as anticipated, with the state estimates of any pair of nodes converging towards the more accurate estimate, and doing so more quickly when the nodes are separated by fewer communication hops.

The proposed extended Kalman-Consensus filter scales better with an increasing number of nodes than a central extended Kalman filter, and provides similar, if not better, state estimation accuracy. The simulations have shown that the extended Kalman-Consensus filter is a promising algorithm for estimating the state of a non-linear process via distributed sensing.

6.3 Future Work

A more mathematically rigorous investigation of the error characteristics of the extended Kalman-Consensus filter would allow a quantifiable description of the performance of the filter.

More work needs to be done to determine how the formation of sensors affects the performance of the overall system, and how well the system performs with a larger number of sense nodes. This work laid a general framework for the extended Kalman-Consensus filter, but a real-world application of the filter to a specific task would truly test its usefulness. Sense node hardware could be designed, and the software could be tweaked to apply the nodes to any of a number of tasks.

The sensor types and formations used in the preceding simulations were not chosen in an attempt to push the limits of this type of algorithm. More work could be done to explore those limits by testing networks of less capable (e.g. single-axis) sensors arranged in various less-than-favorable formations.

REFERENCES

[1] R. Olfati-Saber, "Distributed kalman filter with embedded consensus filters," in *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC '05. 44th IEEE Conference on*, pp. 8179 – 8184, dec. 2005.

[2] R. Olfati-Saber, "Distributed kalman filtering for sensor networks," in *Decision and Control, 2007 46th IEEE Conference on*, pp. 5492 –5498, dec. 2007.

[3] R. Olfati-Saber, "Kalman-consensus filter : Optimality, stability, and performance," in *Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on*, pp. 7036 –7042, dec. 2009.

[4] D. P. Spanos, R. Olfati-Saber, and R. M. Murray, "Distributed sensor fusion using dynamic consensus," *http://users.cms.caltech.edu/ murray/preprints/sm05-ifac.pdf*, 2005.

BIOGRAPHY OF THE AUTHOR

Andrew Pellett was born in Sitka, Alaska on August 13, 1987. After growing up in Juneau, Alaska, he moved to Winthrop, Maine, where he attended Winthrop High School, graduating in 2005.

Andrew enrolled at the University of Maine in 2005, and graduated in 2009 with his Bachelor of Science degree with a double major in Electrical Engineering and Computer Engineering. He then continued at the University of Maine, pursuing a Master of Science degree in Computer Engineering.

While at the University of Maine, Andrew worked as a Teaching Assistant for ECE 486 Digital Signal Processing, and as a Research Assistant on the off semesters. His research interests include signal processing, hardware design, and difficult problems in general.

Andrew is a member of IEEE, and his extracurricular interests include hiking, travel, flying, and staying active.

He is a candidate for the Master of Science degree in Computer Engineering from the University of Maine in December 2011.