PARALLEL IMPLEMENTATIONS OF THE KALMAN FILTER FOR TRACKING APPLICATIONS

BY

EDWARD KWANG BOK LEE

A Thesis
Submitted to the School of Graduate Studies
in Partial Fulfillment of the Requirements
for the Degree
Doctor of Philosophy

McMaster University

PARALLEL IMPLEMENTATIONS OF THE KALMAN FILTER FOR TRACKING APPLICATIONS

DOCTOR OF PHILOSOPHY (1990) McMaster University
(Electrical Engineering) Hamilton, Ontario

TITLE: Parallel Implementations of the Kalman Filter for Tracking Applications

AUTHOR: Edward Kwang Bok Lee, B.A.Sc. (University of Toronto)
M.Eng. (University of Toronto)

SUPERVISOR: Dr. S. Haykin

NUMBER OF PAGES: xiii, 251

ABSTARACT

The first parallel implementations of the extended covariance Kalman filter (ECKF) and the extended square root covariance filter (ESRCF) for tracking applications are developed in this thesis. The decoupling technique and special properties in the tracking KF are explored to reduce computational requirements and to increase parallelism.

The use of the decoupling technique to the ECKF eliminates the need for a matrix inversion, and results in the time and measurement updates of m decoupled (n/m)–dimensional state esimate error covariance $P_o(k)$'s instead of 1 coupled n–dimensional covariance matrix P(k), where m denotes the tracking dimension and n denotes the number of state elements.

Similarly, the use of the decoupling technique to the ESRCF separates the time and measurement updates of 1 coupled $P^{1/2}(k)$ into those of m decoupled $P_o^{1/2}(k)$'s.

The updates of m decoupled matrices are found to require less computation than those of 1 coupled matrix, and they may be performed for each axis in parallel.

In the parallel implementation of time and measurement updates of P(k) in the ECKF, the updates of m decoupled $P_o(k)$'s are found to require approximately m times less number of processing elements and clock cycles than the updates of 1 coupled P(k). Similarly, the parallel implementation of the updates of m decoupled $P_o^{1/2}(k)$'s in the ESRCF requires approximately m time less number of

processing elements and clock cycles than that for 1 coupled $P^{1/2}(k)$.

The transformation of the Kalman gain which accounts for the decoupling of P(k) and $P^{1/2}(k)$ is found easy to implement.

The sparse nature of the measurement matrix and the sparse, band nature of the transition are explored to simplify matrix multiplications.

ACKNOWLEDGEMENTS

I would like to express my deep gratitude to Dr. Simon Haykin for his supervision and encouragement during this research. I am also grateful to Drs. W.F.S. Poehlman and N.K. Sinha for useful suggestions for improving this thesis. I would also like to thank the staff and graduate students of the Communications Research Laboratory for their discussions and encouragement.

Finally, to my parents and wife, I wish to express my sincere appreciation of their support and understanding during my study for the Ph. D. degree.

TABLE OF CONTENTS

CHAPTER

LIST OF FIGURES

page

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Filtering and the Kalman Filter Algorithm

The concept of filtering is fundamental to all forms of communications and signal processing. The signals which we receive are very often corrupted by noise. For example, in a conversation over a noisy radio channel, the communication is degraded by the high level of receiver noise. In this case, we filter out the noise and estimate what has been said, based on the received noisy voice. Filtering can be viewed as the minimization of the effects of noise contained in a received signal. The term "filter" is used to describe a device or algorithm designed to extract a signal. It is implemented in the form of physical hardware or computer software.

One type of filter is a circuit or system with frequency selective behaviour [40], [46]. An example is a filter in radio receivers that amplifies signals in one frequency band and attenuates unwanted noise in another. This type of filter is based on the postulate that the useful signals lie in one frequency band and unwanted noise lies in another.

However, the type of filter of interest in this thesis is the Kalman filter. The Kalman filter (KF) is an algorithm that extracts information of interest from a set of noisy data. It is based on the a priori statistical, linear, mathematical representation of system dynamics and measurement equations

([1],[5],[19],[23],[24],[44]). The system dynamics equation describes how the state of a system, to which filtering is applied, varies with time. The state of a system contains all the necessary information about the behaviour of the system. The measurement equation describes the relationships between measurements and the state of the stystem.

Based on this a priori representation, the KF estimates the state of a system by minimizing the mean–square value of errors in a state estimate. The KF uses recursively the old estimate and new input data in making a state estimate. Hence, the KF does not require to store the entire past input data nor all the previous estimates except for the most recent estimate.

In the Kalman filter, the state of a system is expressed in the form of a vector, referred to as a state vector. Similarly, the estimate of system's state is expressed in the form of a vector, called a state estimate vector.

The accuracy of a state estimate may be determined in the form of an ensemble–averaged outer product of the state estimate error vector. The state estimate error vector is defined as the difference between the state vector and the state estimate vector. This ensemble–averaged outer product is referred to as the state estimate error covariance matrix, or simply an error covariance matrix. It is interesting to note that the error covariance matrix is a multi–random variables' version of the variance in a single random error variable. Like the variance in a single random error variable, large elements of the state estimate error covariance matrix indicate large errors in corresponding state elements.

Since the exact state of a system is always unknown, the exact accuracy of a state estimate is impossible to find, but it can be estimated. The KF estimates the accuracy of its state estimate at each filtering instant. However, estimation of the error covariance matrix sometimes results in numerical problems

due to the effects of finite word-length arithmetic. To reduce these problems, it has been suggested that the estimated accuracy of a state estimate be expressed in terms of the square root of the error covariance matrix [25]. This is analogus to expressing the accuracy of a random variable in terms of a standard deviation rather than a variance. Expressing the accuracy in terms of a standard deviation rather than a variance increases the effective precision by a factor of two, since the square-root operation reduces the required number of bits to represent the accuracy by a factor of two. However, this increase in the effective precision is achieved by additional computational complexity [25]. Hence, the decision as to the way expressing the estimated accuracy should be based on the requirements of computational complexity and numerical stability.

In the literature, the KF with the accuracy expressed in terms of the square root of the error covariance matrix is referred to as the square root covariance filter (SRCF), whereas the KF that makes use of the error covariance matrix is referred to as the covariance Kalman filter. The term, KF, will be generally used to refer to both versions of the Kalman filter as a whole hereafter, but it will be occasionally used to refer to only the covariance KF. The use of this term should be clear in the context.

1.2 The Target Tracking Problem

The purpose of target tracking is to track the state of a target such as its position and velocity, using measurements made by suitable sensors (e.g., radars) at discrete time instants ([10],[41], [42].[43]). The target can be an aircraft, ship, or missile. Errors are inherent from various sources of measurement. One of the major sources of error is the thermal noise at the front end of the receiver of the radar. The KF may be used to filter out the inherent errors in measurements, and

to estimate the state of a target. However, the KF has to be modified to handle special features of target tracking systems.

In target tracking systems, the position and velocity of a target are expressed in Cartesian coordinates, while measurements are made in polar coordinates in terms of range and bearing angle, as shown in Figure 1.1.

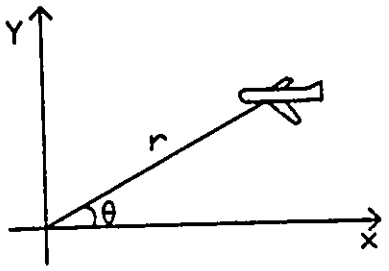


Figure 1.1 Measurement in polar coordinates

Hence, the measurement equation, describing the relationships between measurements and the state of a target, is nonlinear. Since the standard KF assumes a linear measurement equation, it has to be modified to be used in tracking applications. The KF, including the linearization of a nonlinear measurement equation and a coordinate transformation from polar to Cartesian coordinates, is called the extended KF [1],[5]. This coordinate transformation is required in filtering to relate the state in Cartesian coordinates and measurement in polar coordinates. When the covariance KF and the SRCF are modified to handle nonlinearities in tracking systems, they are referred to as the extended covariance KF and the extended SRCF, respectively.

1.3 Purpose of the Research

The computational power of a single processor is not sufficient for a large number of filtering applications using the KF, although advances in digital

integrated circuit technology have considerably improved the throughput rate of such processors. However, as VLSI technology becomes more accessible and cost effective, interest in the parallel implementation of the KF has increased. There have been a number of papers on this topic ([6],[11],[13],[21],[22], [31], [37],[48],[52]) However, with the exception of [11], these papers have focused mainly on the standard KF for general filtering applications, and not the extended KF (EKF) for tracking applications. The implementation of [11] is based on optical technology and does not discuss the implementation of the nonlinear coordinate transformation from polar to Cartesian coodinates, or the linearization of the measurement equation. Hence, there has not been any parallel implementation of the extended KF proposed for tracking applications.

The purpose of this thesis is thus to develop the first parallel architectures for the EKF for tracking applications. Parallel implementations of both the extended covariance KF and the extended SRCF are developed, so that the architecture with suitable characteristics can be selected depending on the requirements of computational complexity and numerical stability. A decoupling technique and special properties inherent in tracking systems are exploited to simplify implementations. These simplifications are discussed in detail in Chapter 4. The utilization of special properties of a particular application does not limit the applicability of the proposed architectures, because the proposed architectures are already designed for a special use.

1.4 Previous Work on the Parallel Implementations of the Kalman Filter

Architectures based on a single processor have a limited computing power. To increase the computing power, the concepts of pipelining and parallel processing are utilized, using a number of processors ([20],[30],[45]). Pipelining

and parallel processing systems are respectively depicted in Figures 1.2 and 1.3, where boxes, labelled P_i's, represent processors.

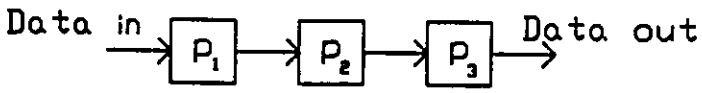


Figure 1.2 Pipeline processing system

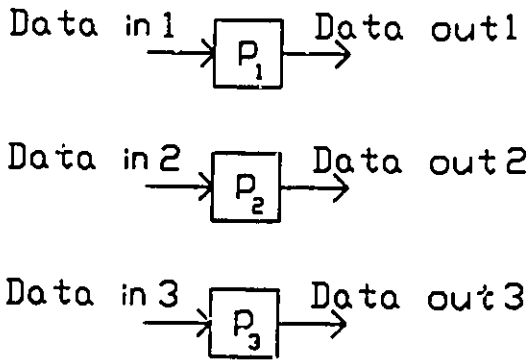


Figure 1.3 Parallel processing system

In Figure 1.2, processors are serially connected along the line. This line is referred to as a pipeline. In a pipeline system, a task is decomposed into many subtasks. Each subtask is processed at successive processors as data goes through the pipeline. When data comes out of the pipeline, a task is completed. Pipelining allows new tasks to be initiated before the current task is completed, because processors become available for new tasks once they complete their subtasks. In contrast to a pipeline system, a parallel processing system consists of a number of independent processors that perform independent tasks in parallel.

As the computing power of multiprocessor architectures increases, local

interconnection between a processor and its neighboring processors becomes more and more desirable, since it is simple to implement and reduces the amount of Input/Cutput to and from memory. Multiprocessor arhitectures are generally implemented in very large scale integration (VLSI) technology. VLSI technology favours regularity and simplicity of processors to reduce design errors, time, and cost [30],[34]. Modularity is also desired to adjust the number of processors for the required computing power.

Systolic arrays, originally proposed by H.T.Kung [26],[28], have all the features mentioned above and exploit both the concepts of pipelining and parallel processing. A systolic array consists of an array of individual processing cells that are arranged in a regular structure. Each cell is connected to its nearest neighbors. The systolic array is designed such that the regular streams of data are clocked through it in a highly rhythmic fashion, much like the pumping action of the human heart; hence, the name "systolic". An example of a systolic array designed for the multiplication of two matrices is shown in Figure 1.4.

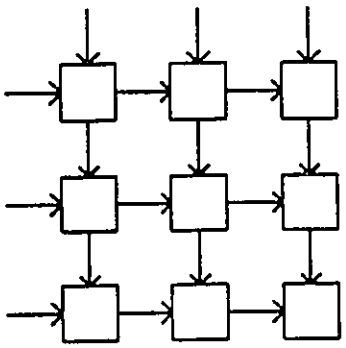


Figure 1.4 Systolic array for the multiplication of two matrices

In a sytolic array, data are reused as they travel through processing cells

in the array. The reuse of data reduces the amount of Input/Output requirements from and to memory. However, the reuse of data may cause a nonnegligible time delay because a significant amount of time is required by the data to go through many processing cells; that is particularly so when the systolic array is large. The additional delay is created in the systolic array to introduce a skew in an input data stream for the purpose of data synchronization.

When the reduction of time delay is required and the advantages of systolic arrays are not significant, a parallel processing architecture with global interconnection can be employed. This architecture, a special type of parallel processing architecture, broadcasts data to all the appropriate processing elements at once using a global interconnection, instead of passing data one at a time through processing elements in the pipeline. It is shown in Figure 1.5. The global interconnection is not preferrable to a local interconnection, but this is a price to pay to reduce the time delay.

Inspite of their desirable features, systolic arrays have drawbacks. Since all the processing cells in the systolic array are controlled by the same global clock, the incurred clock skew may be nontrivial. The clock skew should be carefully accounted for in implementation.

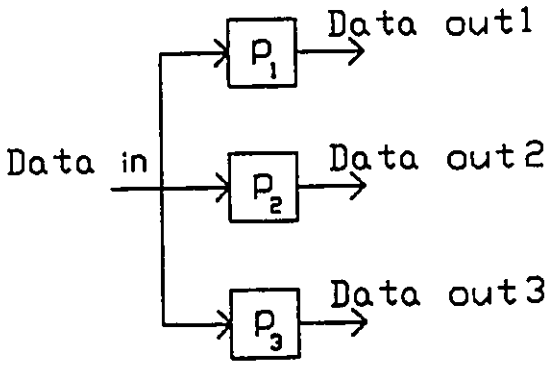


Figure 1.5 Parallel processing architecure with global interconnection

Systolic arrays have been applied to various applications such as matrix manipulation [28], relational database operations[27], and filtering applications ([18],[29],[30]). In particular, they have been used in the implementation of the Kalman filter with other types of multiprocessor architectures.

Kadela and Graham [22] studied parallel implementations of the covariance KF. Papadourkis and Taylor [37], and Yeh [52] presented a systolic imlementation of the covariance KF. Jover and Kailath [21] presented an parallel architecture for the measurement update part of the SRCF, while Sung and Hu [48], and Gaston and Irwin [14] proposed an architecture for the complete SRCF. Furthemore, Chen and Yao [6], S.Y.Kung and Hwang [31], and Gaston and Irvin [13] developed systolic architectures for the modified SRCF which estimates the accuracy of its state estimates in terms of the inverse of the square root of the error covariance matrix. However, all of these architectures implemented different versions of the standard KF, not the extended KF for tracking systems. Hence, this thesis is the first complete development of parallel architectures of the extended KF for tracking application.

1.5 Outline of the Thesis

In Chapter 2, we review the representation of typical tracking systems, and various filters for tracking applications: the α–β filter and different versions of the KF including the extended KF.

In Chapter 3, we first study design principles for parallel architectures, then review two commonly used systolic arrays to gain insights into parallel architectures, and to study the properties of systolic arrays. At the end of Chapter 3, we review some of the parallel implementations of the KF so far presented.

In Chapter 4, we first present a decoupling technique and properties of

matrices in the tracking KF. We then show how the decoupling technique and properties of matrices can be used to simplify the extended covariance KF and the extended SRCF. Next, we study the performance of the simplified EKF, in comparison to the standard EKF.

In Chapters 5 and 6, we develop parallel architectures for the extended covariance KF and the extended SRCF respectively, utilizing the decoupling technique and special properties of the certain matrices in the tracking KF.

Finally, in Chapter 7 we present conclusions.

CHAPTER 2

TRACKING FILTERS

The purpose of "tracking" is to keep track of the state of a target such as its position and velocity. Tracking filters estimate the state of a target by filtering out inherent errors in measurements. Such filters require the a priori representation of the target dynamics, and that of the relationships between measurements and the state of a target. The state of a target is represented in the form of a vector.

Different tracking filters have different characteristics in terms of filtering accuracy, computational requirements, implementational complexity, robustness, and numerical properties. The implementational complexity of a tracking filter is determined by the structure of information flow in the filter. A filter with regular structure is easier to implement than a filter without it. The numerical properties indicate how stable filters are with respect to round-off errors.

The operations of the tracking filters are perhaps best explained by describing a typical tracking scenario. Upon detection of a target, the tracking filter initializes a state vector, which consists of position and velocity estimates, and then the tracking filter predicts the state at the next filtering time instant. This prediction is made with a priori knowledge of the target motion dynamics. At the next filtering time instant, the tracking filter first makes a state estimate combining the previous predicted state estimate with the newly received

measurement, and then the tracking filter makes a state prediction for the following filtering instant. The tracking filter repeats this estimation and prediction procedure at each sampling instant. There are various ways of combining the predicted state and measured values in tracking filters.

In the following sections, a more detailed representation of tracking systems will be given, followed by a discussion of various tracking filters.

2.1 Tracking System Representation

A target tracking system may be expressed in the following two equations:

a) A target dynamics equation

$$X(k+1) = \phi(k)\, X(k) + D(k) \tag{2.1}$$

b) A measurement equation

$$Z(k) = H(k)\, X(k) + E(k) \tag{2.2}$$

In this representation, the state of a target, which contains all the information necessary to specify the condition of a target, is expressed in the form of a vector. This vector is called a state vector, denoted by X(k), and the elements of this vector are called state variables. The representation of a system in terms of state vectors is referred to as a state–space model, since an n–dimensional space can be formed in which each coordinate is defined by one of the state variables, where n is the number of state elements.

The state vectors for typical 2– and 3–dimensional tracking are as follows

a) 2–dimensional case:

$$X(k) = \begin{bmatrix} x(k) \\ \dot{x}(k) \\ \ddot{x}(k) \\ y(k) \\ \dot{y}(k) \\ \ddot{y}(k) \end{bmatrix} = \begin{bmatrix} \text{x–position} & \text{at time k} \\ \text{x–velocity} & \text{at time k} \\ \text{x–acceleration} & \text{at time k} \\ \text{y–position} & \text{at time k} \\ \text{y–velocity} & \text{at time k} \\ \text{y–acceleration} & \text{at time k} \end{bmatrix}$$

b) 3–dimensional case:

$$X(k) = \begin{bmatrix} x(k) \\ \dot{x}(k) \\ \ddot{x}(k) \\ y(k) \\ \dot{y}(k) \\ \ddot{y}(k) \\ z(k) \\ z(k) \\ \ddot{z}(k) \end{bmatrix} = \begin{bmatrix} \text{x–position} & \text{at time k} \\ \text{x–velocity} & \text{at time k} \\ \text{x–acceleration} & \text{at time k} \\ \text{y–position} & \text{at time k} \\ \text{y–velocity} & \text{at time k} \\ \text{y–acceleration} & \text{at time k} \\ \text{z–position} & \text{at time k} \\ \text{z–velocity} & \text{at time k} \\ \text{z–acceleration} & \text{at time k} \end{bmatrix}$$

Note that in this representation the acceleration of a target is also assumed to be kept track of, in addition to the position and velocity of a target.

The target dynamics equation, as the name implies, expresses the target dynamics in terms of state vectors X(k) and X(k+1). The relationships between X(k) and X(k+1) are described by a transition matrix $\phi(k)$. A typical transition matrix for 2–dimensional tracking is [42]

$$\phi(k) = \begin{bmatrix} 1 & T & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & \rho & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & T & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & \rho \end{bmatrix}$$

where T is the sampling time interval, and ρ is a correlation coefficient for acceleration.

The vector D(k) in Equation (2.1) represents the disturbance or

uncertainty in the system dynamics model. This disturbance is due to the approximation in the model of the dynamics, random motions caused by environments, and deliberate target maneuver.

The measurement equation (2.2) relates, using a measurement matrix H(k), a state vector X(k) and a measurement vector Z(k) whose elements are physically measured. The vector E(k) describes inherent errors associated with measurements. A typical tracking radar does not measure all the elements of a state vector. For example, some tracking radars measure only the position of a target.

The measurement equation will be explained with a hypothetical, simple 1-dimensional tracking example. We assume that the state vector X(k) for 1-dimensional tracking is defined as

$$X(k) = \begin{bmatrix} x(k) \\ \dot{x}(k) \\ \ddot{x}(k) \end{bmatrix} = \begin{bmatrix} \text{x-position at time k} \\ \text{x-velocity at time k} \\ \text{x-acceleration at time k} \end{bmatrix}$$

and only the position of a target is measured with errors, such that the measurement vector Z(k) and the measurement noise vector E(k) are defined as

$$Z(k) = \begin{bmatrix} z(k) \end{bmatrix}$$

$$E(k) = \begin{bmatrix} e(k) \end{bmatrix}$$

where z(k) and e(k) denote, respectively, a position measurement and a measurement error at time k. Then the measurement matrix H(k) becomes

$$H(k) = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$$

and the measurement equation is expressed as

$$Z(k) = H(k)X(k) + E(k) = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x(k) \\ \dot{x}(k) \\ \ddot{x}(k) \end{bmatrix} + \begin{bmatrix} e(k) \end{bmatrix}$$

A typical tracking radar which is of interest in this thesis measures range r and azimuth angle θ_A in 2–dimensional tracking, and range r, azimuth angle θ_A and elevation angle θ_E in 3–dimensional tracking, such that the measurement vector Z(k) is for 2–dimensional tracking

$$Z(k) = \begin{bmatrix} r(k) \\ \theta_A(k) \end{bmatrix} = \begin{bmatrix} \text{range at time k} \\ \text{azimuth angle at time k} \end{bmatrix},$$

and for 3–dimensional tracking,

$$Z(k) = \begin{bmatrix} r(k) \\ \theta_A(k) \\ \theta_E(k) \end{bmatrix} = \begin{bmatrix} \text{range at time k} \\ \text{azimuth angle at time k} \\ \text{elevation anlge at time k} \end{bmatrix}$$

Yet, the state vector X(k) is defined in terms of x(k), $\dot{x}(k)$, $\ddot{x}(k)$, y(k), $\dot{y}(k)$, and $\ddot{y}(k)$ in Cartesian coordinates, since the motion of a target is linear in Cartesian coordinates. Hence it is not possible to express the measurement equation, the relationships between X(k) and Z(k), in the form of a linear matrix H(k), but in the form of a nonlinear function h(.). The required nonlinear funcion actually performs a transformation from Cartesian to polar coordinates to relate X(k) in Catesian coordinates and Z(K) in polar coordinates. A modified measurement

equation with a nonlinear measurement function h(.) is expressed by

$$Z(k) = h(X(k)) + E(k) \tag{2.3}$$

The nonlinear function h(X(k)) for 2–dimensional tracking is defined using Figure 2.1, and that for 3–dimensional tracking is defined using Figure 2.2.

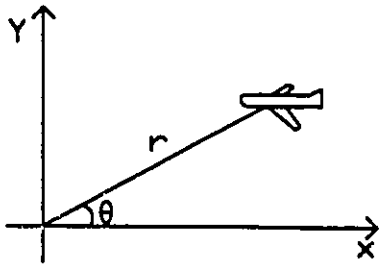


Figure 2.1 Measurement for 2–dimensional tracking

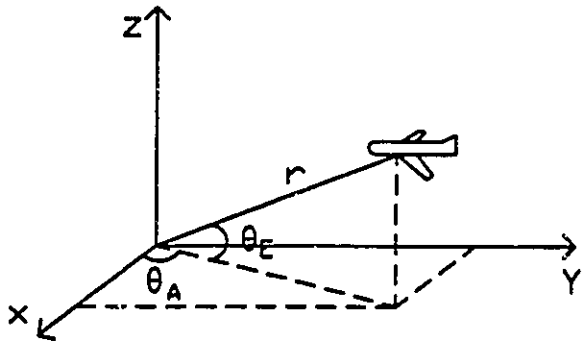


Figure 2.2 Measurement for 3–dimensional tracking

For 2–dimensional tracking, the two components of h(X(k)) are

$$r(k) = ((x(k))^2 + (y(k))^2)^{1/2} \tag{2.4}$$

$$\theta_A(k) = \tan^{-1}(y(k)/x(k)) \tag{2.5}$$

For 3–dimensional tracking, the three components of h(X(k)) are

$$r(k) = ((x(k))^2 + (y(k))^2 + (z(k))^2)^{1/2} \tag{2.6}$$

$$\theta_A(k) = \tan^{-1}(y(k)/x(k)) \tag{2.7}$$

$$\theta_E(k) = \tan^{-1}(z(k)/(x(k)^2+y(k)^2)^{1/2}). \tag{2.8}$$

2.2 Tracking Filter

The purpose of a tracking filter is to estimate a state vector in some optimal sense. A number of tracking filters have been used such as the α–β filter and the tracking Kalman filter. These filters make estimates as a weighted combination of a prediction and measurement, as described before. The type of filter to be used is determined by its complexity, achievable estimation accuracy and other numerical properties of filters. The complexity of a tracking filter is largely determined by the way the filter computes a weighting factor, sometimes called a gain matrix, to combine predictions and measurements. Some filters like the α–β filter use a fixed gain matrix, whereas the Kalman filter computes a gain matrix at every time instant.

In this section, we first briefly describe the α–β filter for tracking applications. We then describe the standard covariance KF and the standard square root covariance KF to gain insights into the functionalities of the KF in

general, followed by the extended covariance KF and the extended square root covariance filter for tracking applications.

2.2.1 The α–β filter.

The α–β filter estimates the state of a target by combining a prediction and actual measurement with a time invariant weighting factor ([3],[39]). The prediction is made at the previous filtering time instant with the a priori knowledge of the target dynamics, on the basis of the state estimate at that filtering instant. At the beginning of filtering, the state estimate is initialized with the a priori knowledge on the state of a target, since the state estimate is not available but required to make a prediction for the next filtering instant.

The time invariant weighting factor is generally determined to minimize the mean–square error of state estimates in a steady state, using the a priori knowledge of the target dynamics equation and a measurement equation.

The computational requirements of the α–β filter are very small compared to a filter that calculates a weighting factor at each filtering instant, since the α–β filter does not calculate a new weighting factor. However, the use of the same weighting factor makes the α–β filter not adaptable to a changing environment, because this weighting factor does not change with variations in the target dynamics and the measurements' accuracy.

2.2.2 The Kalman filter

The Kalman Filter was proposed by Kalman in 1960 [23]. In the Kalman Filter, the system dynamics equation and that of a measurement equation are required a priori. In other words, a transition matrix $\phi(k)$, a measurement matrix H(k), a target dynmics noise vector D(k), and a measurement noise vector E(k)

in Equations (2.1) and (2.2) have to be specified. Since D(k) and E(k) are random, the ensemble averaged correlation matrices

$$E[\ D(k)D^T(k)\ ] = Q(k)$$
$$E[\ E(k)E^T(k)\ ] = R(k)$$

are required to be respectively specified for D(k) and E(k).

Based on this a priori knowledge, the KF makes a state estimate by combining a prediction and actual measurement with a time varying weighting factor. As in the α–β filter, prediction is made at the previous filtering time instant with the a priori knowledge of the system dynamics. The weighting factor, or Kalman gain, is calculated at every filtering instant according to the accuracy of a prediction and measurement. This is to minimize the mean squared error of state estimates. The measurement accuracy, $E[\ E(k)E^T(k)\ ]$, is specified a priori, and the accuracy of a prediction is updated as the filtering progresses, which is one of the KF's functions. The accuracy is represented in the form of an error covariance matrix, as described in Chapter 1.

The KF is summarized in Equations (2.9 .. 2.13). The definition of all the symbols are included for completeness, although some of the symbols have already been defined.

Measurement update

$$K(k) = P(k|k-1)\ H^T(k)\ (\ H(k)\ P(k|k-1)\ H^T(k)\ +\ R(k)\ )^{-1} \tag{2.9}$$

$$P(k) = (\ I - K(k)\ H(k)\ )\ P(k|k-1) \tag{2.10}$$

$$\hat{X}(k) = \hat{X}(k|k-1) + K(k)\ (\ Z(k) - H(k)\ \hat{X}(k|k-1)\ ) \tag{2.11}$$

Time update

$$P(k+1|k) = \phi(k)\, P(k)\, \phi^{T}(k) + Q(k) \tag{2.12}$$

$$\hat{X}(k+1|k) = \phi(k)\, \hat{X}(k) \tag{2.13}$$

where $\hat{X}(k)$ = a state estimate vector

$\hat{X}(k|k-1)$ = predicted state vector

$K(k)$ = gain matrix

$Z(k)$ = measurement vector

$H(k)$ = measurment matrix

$P(k)$ = state estimate error covariance matrix

$P(k|k-1)$ = predicted state estimate error covariance matrix

$\phi(k)$ = transition matrix

$Q(k)$ = system dynamics noise variance matrix

$R(k)$ = measurement noise variance matrix

The KF in Equations (2.9 .. 2.13) is separated into two parts: measurement update and time update. In the measurement update part, the Kalman gain $K(k)$ is first calculated in Equation (2.9) which requires a computationally demanding matrix inversion. And then the state estimate error covariance matrix $P(k)$ is calculated in Equation (2.10), followed by the state estimate $\hat{X}(k)$ in Equation (2.11). The state estimate $\hat{X}(k)$ is made as a combination of the predicted state $\hat{X}(k|k-1)$ and measurement $Z(k)$. The term, $Z(k) - H(k)\hat{X}(k|k-1)$, indicates the difference between the measurement $Z(k)$ and the predicted measurement $H(k)\hat{X}(k|k-1)$. The predicted measurement is calculated by multiplying the predicted state $\hat{X}(k|k-1)$ by the measurement matrix $H(k)$.

In the time update part, the prediction $P(k+1|k)$ of the state estimate

error covariance matrix, and the prediction $\hat{X}(k+1|k)$ of the state estimate are made.

Note that a large number of matrix–matrix multiplications and matrix–vector multiplications, and a matrix inversion are required in each iteration. This meams that that the KF is computationally demanding. The high computational demand of the KF has limited the use of the KF in various real–time applications.

However, the KF does not have to store all the measurements Z(k)'s and state predictions $\hat{X}(k|k-1)$'s, because the state prediction $\hat{X}(k|k-1)$ contains all the information on the system up to time k–1. Hence, the state estimate $\hat{X}(k)$ is made of a combination of Z(k) and $\hat{X}(k|k-1)$, instead of all the previous Z(k)'s and X(k|k–1)'s. The estimation of state X(k) in terms of Z(k) and $\hat{X}(k|k-1)$, which is in turn defined in terms of $\hat{X}(k)$, makes the KF recursive, since the state estimate $\hat{X}(k-1)$ at the previous step is used in estimating $\hat{X}(k)$, and $\hat{X}(k)$ is in turn used in estimating $\hat{X}(k+1)$.

In summary, the KF is a computationally demanding recursive filtering algorithm, which requires that the system dynamics and measurement equations be defined a priori. The KF is more sophisticated, more accurate, and more computationally demanding than the α–β filter.

The Kalman filter, discussed in this section, is referred to as the covariance KF, since this KF is formulated in terms of the state estimate error covariance matrix P(k).

2.2.3 The Square Root Covariance Filter

The computation of the KF equations with finite word–length arithmetic may lead to numerical problems due to cumulative roundoff or truncation errors.

In particular, the state estimate error covariance matrix P(k) may become negative definite [4],[25]. It can be easily shown by examining Equation (2.10) where the error covariance matrix P(k) is computed as the difference between two nonnegative definite matrices.

To guarantee the positive semidefinite nature of the error covariance matrix P(k) and to improve the numerical properties of the KF, the KF has been reformulated in terms of the square root of P(k), where the square root of P(k) is updated as the filtering progresses, instead of P(k). The reformulated KF is called the square root covariance filter (SRCF). The numerical properties of the SRCF are much better than those of the KF. This can be illustrated by examining the condition number of P(k) and that of $P^{1/2}(k)$. The condition number K(A) of an arbitrary matrix A is defined as

$$K(A) = \sigma_1/\sigma_n$$

where σ_1^2 is the maximum eigenvalue of A^TA and σ_n^2 is the minimum eigenvalue of A^TA. It indicates how sensitive the matrix is to errors. The larger the condition number, the more sensitive the matrix A is to purturbations. The relationship between the condition number of P(k) and that of $P^{1/2}(k)$ is

$$K(\ P(k)\ ) = [\ K(\ P^{1/2}(k)\ )\ ]^2$$

The condition number of P(k) is a square of the condition of number of $P^{1/2}(k)$. Hence, the SRCF is numerically superior to the covariance KF.

However, the SRCF is found to require upto 50% more computation than the standard covariance KF in most practical applications [25]. The amount of

additional computation depends on the dimension of the error covariance matrix P(k) and measurement matrix H(k). Hence, the SRCF should be preferrable to the KF, when the guarantee of the positive semidefiniteness of P(k) and the improved numerical properties are required, and the additional computational requirements are acceptable.

The reformulated KF, the SRCF, is summarized below [25]:

Measurement update

$$\begin{bmatrix} F(k) & G(k) \\ 0 & S^T(k) \end{bmatrix} = Q_1 \begin{bmatrix} V^T(k) & 0 \\ S^T(k|k-1)H^T(k) & S^T(k|k-1) \end{bmatrix} \tag{2.14}$$

$$K(k) = G^T(k) / F^T(k) \tag{2.15}$$

$$\hat{X}(k) = \hat{X}(k|k-1) + K(k) \left(Z(k) - H(k) \hat{X}(k|k-1) \right) \tag{2.16}$$

Time update

$$\begin{bmatrix} S^T(k+1|k) \\ 0 \end{bmatrix} = Q_1 \begin{bmatrix} S^T(k) \, \phi^T(k) \\ U^T(k) \end{bmatrix} \tag{2.17}$$

$$\hat{X}(k+1|k) = \phi(k) \hat{X}(k) \tag{2.18}$$

where lower triangular matrices S(k), S(k+1|k), U(k), V(k) are defined as follows:

$$P(k) = S(k) S^T(k)$$

$$P(k+1|k) = S(k+1|k) S(k+1|k)^T$$

$$Q(k) = U(k) U^T(k)$$

$$R(k) = V(k) V^T(k)$$

Q_1 denotes an orthogonal matrix which upper-triangularizes a matrix on the right hand side.

Like the covariance KF, the SRCF is divided into two parts: measurement and time updates. As expected, the SRCF updates the square root S(k) of P(k). The equivalence between the update in terms of S(k) and that of P(k) can be easily shown by the multiplication of both sides of Equation (2.17) by the transpose of themselves, as show below:

$$\begin{bmatrix} S(k+1|k) & 0 \end{bmatrix} \begin{bmatrix} S^T(k+1|K) \\ 0 \end{bmatrix} = \begin{bmatrix} \phi(k)S(k) & U^T(k) \end{bmatrix} Q_1^T \; Q_1 \begin{bmatrix} S^T(k)\,\phi^T(k) \\ U^T(k) \end{bmatrix}$$

$$S(k+1|k)S^T(k+1|k) = \phi(k)S(k)\phi^T(k) + U(k)U^T(k)$$

$$P(k+1|k) = \phi(k)P(k)\phi^T(k) + Q(k)$$

It shows that Equation (2.17), which updates the square root S(k) of P(k) for the SRCF, is equivalent to Equation (2.12), which updates P(k) for the covariance KF. Note that the property, $Q_1^T Q_1 = I$, is used in this calculation.

2.2.4 The Extended Kalman filter.

As discussed in Section 2.1, a typical tracking system makes measurement in polar coordinates, and yet performs filtering in Cartesian coordinates, such that the measurement equation for a typical tracking application

$$Y(k) = h(X(k)) + E(k). \tag{2.3}$$

has a nonlinear measurement function h(.). The function h(.), which relates

measurments and the state of a target, is defined in Equations (2.4) and (2.5) for 2–dimensional tracking, and in (2.6 .. 2.8) for 3–dimensional tracking. However, the standard KF's presented so far, the covariance KF and the SRCF, assume a linear measurement equation

$$Y(k) = H(k)X(k) + E(k) \tag{2.2}$$

To overcome this problem, the extended KF (EKF) with the linearization of h(.) has been proposed [1],[5]. In this filter, the linearization of h(.):

$$H(k) = \frac{\partial h(x)}{\partial x} \Bigg| \; X = \hat{X}(k|k-1)$$

is performed at every filtering instant. The term $\frac{\partial h(x)}{\partial x}$ is defined as follows:

a) 2–dimensional tracking:

$$\frac{\partial h(x)}{\partial x} = \begin{bmatrix} \frac{\partial r}{\partial x} & \frac{\partial r}{\partial y} & \frac{\partial r}{\partial \dot{x}} & \frac{\partial r}{\partial \dot{y}} & \frac{\partial r}{\partial \dot{x}} & \frac{\partial r}{\partial \dot{y}} \\ \frac{\partial \theta_A}{\partial x} & \frac{\partial \theta_A}{\partial y} & \frac{\partial \theta_A}{\partial \dot{x}} & \frac{\partial \theta_A}{\partial \dot{y}} & \frac{\partial \theta_A}{\partial \dot{x}} & \frac{\partial \theta_A}{\partial \dot{y}} \end{bmatrix}$$

b) 3–dimensional tracking:

$$\frac{\partial h(x)}{\partial x} = \begin{bmatrix} \frac{\partial r}{\partial x} & \frac{\partial r}{\partial y} & \frac{\partial r}{\partial z} & \frac{\partial r}{\partial \dot{x}} & \frac{\partial r}{\partial \dot{y}} & \frac{\partial r}{\partial \dot{z}} & \frac{\partial r}{\partial \dot{x}\cdot} & \frac{\partial r}{\partial \dot{y}\cdot} & \frac{\partial r}{\partial \dot{z}\cdot} \\ \frac{\partial \theta_A}{\partial x} & \frac{\partial \theta_A}{\partial y} & \frac{\partial \theta_A}{\partial z} & \frac{\partial \theta_A}{\partial \dot{x}} & \frac{\partial \theta_A}{\partial \dot{y}} & \frac{\partial \theta_A}{\partial \dot{z}} & \frac{\partial \theta_A}{\partial \dot{x}\cdot} & \frac{\partial \theta_A}{\partial \dot{y}\cdot} & \frac{\partial \theta_A}{\partial \dot{z}\cdot} \\ \frac{\partial \theta_E}{\partial x} & \frac{\partial \theta_E}{\partial y} & \frac{\partial \theta_E}{\partial z} & \frac{\partial \theta_E}{\partial \dot{x}} & \frac{\partial \theta_E}{\partial \dot{y}} & \frac{\partial \theta_E}{\partial \dot{z}} & \frac{\partial \theta_E}{\partial \dot{x}\cdot} & \frac{\partial \theta_E}{\partial \dot{y}\cdot} & \frac{\partial \theta_E}{\partial \dot{z}\cdot} \end{bmatrix}$$

The linearization of h(.) results in H(k) as defined below:

a) 2–dimensional tracking:

$$H(K) = \begin{bmatrix} \frac{x}{r} & \frac{y}{r} & 0 & 0 & 0 & 0 \\ \frac{-y}{(x^2+y^2)} & \frac{x}{(x^2+y^2)} & 0 & 0 & 0 & 0 \end{bmatrix}$$

b) 3–dimensional tracking:

$$H(k) = \begin{bmatrix} \frac{x}{r} & \frac{y}{r} & \frac{z}{r} & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{-y}{(x^2+y^2)} & \frac{x}{(x^2+y^2)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{-xy}{r^2(x^2+y^2)^{1/2}} & \frac{-yz}{r^2(x^2+y^2)^{1/2}} & \frac{x^2+y^2}{r^2} & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Furthermore, the coordinate transformation of $X(k|k-1)$ is required in the extended KF to compare the predicted state estimates in Cartesian coordinates and

measurements in polar coordinates.

The extended Kalman filter is summarized in Equations (2.19 .. 2.24). The same symbols are used here as in Equations (2.9 .. 2.13) for the standard covariance KF.

Measurement update

$$H(k) = \left. \frac{\partial\, h(x)}{\partial\, x} \right|_{x=\hat{X}(k|k-1)} \tag{2.19}$$

$$K(k) = P(k|k-1)\, H^T(k)\, (\, H(k)\, P(k|k-1)\, H^T(k) \;+\; R(k,\,)\,)^{-1} \tag{2.20}$$

$$\hat{X}(k) = \hat{X}(k|k-1) + K(k)\, (\, Z(k) - h(\, \hat{X}(k|k-1)\,)\,) \tag{2.21}$$

$$P(k) = (\, I - K(k)\, H(k)\,)\, P(k|k-1) \tag{2.22}$$

Time update

$$P(k+1|k) = \phi(k)\, P(k)\, \phi^T(k) + Q(k) \tag{2.23}$$

$$\hat{X}(k+1|k) = \phi(k)\, \hat{X}(k) \tag{2.24}$$

Note that as described above, the EKF has a linearization of h(.) in Equation (2.19) in addition to the equations common with the covariance KF. The EKF also has a coordinate transformation of $\hat{X}(k|k-1)$, $h(\hat{X}(k|k-1)$, in Equation (2.21).

Simlilarly, the SRCF can be extended to handle a nonlinear measurement equation using a linearized measurement equation and a coordinate transformation. The resulting SRCF, called the extended SRCF, is summarized in Equations (2.25 .. 2.30). The symbols used for the standard SRCF in Equations (2.14 .. 2.18) are used here.

Measurement update

$$H(k) = \left. \frac{\partial\, h(x)}{\partial\, x} \right|_{x=\hat{X}(k|k-1)} \tag{2.25}$$

$$\begin{bmatrix} F(k) & G(k) \\ 0 & S^T(k) \end{bmatrix} = Q_1 \begin{bmatrix} V^T(k) & 0 \\ S^T(k|k-1)H^T(k) & S^T(k|k-1) \end{bmatrix} \tag{2.26}$$

$$K(k) = G^T(k) \,/\, F^T(k) \tag{2.27}$$

$$\hat{X}(k) = \hat{X}(k|k-1) + K(k)\,(\,Z(k) - h(\hat{X}(k|k-1))\,) \tag{2.28}$$

Time update

$$\begin{bmatrix} S^T(k+1|K) \\ 0 \end{bmatrix} = Q_1 \begin{bmatrix} S^T(k)\,\phi^T(k) \\ U^T(k) \end{bmatrix} \tag{2.29}$$

$$\hat{X}(k+1|k) = \phi(k)\,\hat{X}(k) \tag{2.30}$$

2.3 Summary

In this chpater, we have first studied the representation of a tracking system in the form of a state–space model, which is required a priori by the α–β and Kalman filters. We have found that in typical tracking applications the measurement equation, which relates a measurement vector Z(K) and a state vector X(k), has to be expressed in the form of a nonlinear function h(.), not in the form of a linear matrix H(K). The reason for this is that the state vector X(k) is defined in the Cartesian coordinates and the measurements are made in the polar coordinates.

We have shown that the KF is more accurate and computationally more demanding than the α–β filter, for the KF updates the error covariance matrix

P(k) and calculates the gain matrix K(k) at every filtering time instant.

We have presented the reformulation of the Kalman filter to improve numerical characteristics. However, the reformulated KF, the square root covariance filter, requires more computation than the already computationally demanding covariance KF. The selection of a filter to be used should be based on the required computation and numerical stability.

We have modified the standard covariance KF and SRCF to handle a nonlinear measurement equation. The modified covariance KF and SRCF, referred to as the extended covariance KF and extended SRCF, have the linearization of the nonlinear measurement equation and the coordinate transformation of a state vector X(k) from Cartesian to polar coordinates.

CHAPTER 3

PARALLEL IMPLEMENTATIONS OF THE KALMAN FILTER

Significant advancements have been made in digital integrated circuit technology. Nevertheless, the computing power of a system based on a single processor is too limited to be used for a large number of real-time signal processing applications. The forecasted physical limits in the progress of fabrication technology make parallel processor systems more attractive over single processor system to achieve the necessary throughput increase for real-time applications. A special purpose parallel architecture is preferrable to a general purpose parallel architecture because the latter has a significant system overhead and is not optimized for a particular application.

Parallel architectures should be designed with different emphasises from an architecture based on a single processor. In Section 3.1, the design principles for a parallel architecture are investigated. In Section 3.2, to gain insights into and appreciation of parallel processing, we describe two systolic arrays for the multiplication of two matrices and for the QR decomposition. These systolic arrays are commonly used in parallel implementations of various filters, including the KF for tracking applications. In Section 3.3, we then review parallel implementations of the Kalman filter so far presented in the literature.

3.1 Parallel Architecture Design Principles

Parallel architecture should be developed, bearing in mind the characteristics of VLSI technology, since parallel architectures are generally implemented in VLSI technology [29]. The overall parallel architecture should be regular and modular to reduce design errors, time, and cost. To achieve this, a building block structure, in terms of regular processing cells, is desirable.

One of the goals of any parallel design is to maximize computing power. This can be accomplished by pipelining and parallel processing, using an array of processing cells. In a pipeline system where processors are serially connected, as shown in Figure 1.1, a task is broken into many subtasks. Each subtask is processed at successive processors, as data pass through the processors. As a result, processors are available for new tasks, once data pass through them. Hence, the computing power can be increased by initiating new tasks on available processors, while some tasks are being processed in the pipeline. In contrast to pipelining, parallel processing architectures increase computing power by processing independent operations simultaneously, using an array of independent processors, as shown in Figure 1.2.

As the computing power of an architecture increases, the data input and output (I/O) rate among processing elements and that between processing elements and memory become more critical. The increased computing power cannot be utilized, if the data I/O rate is not of comparable speed. Hence, the computing power and I/O rate have to be balanced.

Local interconnection among processing elements reduces the amount of I/O to and from memory by allowing processing elements to receive data from their neighbours and to pass it to their neighbours. The local interconnection is also very desirable, for the connecting wires occupy significant part of the silicon

area, and the time delay over wires becomes comparable to the logic gate delay.

3.2 Systolic Arrays

A systolic array is a special type of multi-processor architecture which was proposed by H.T. Kung [26],[27]. It consists of an array of processing cells that are arranged in a regular structure. Each cell is connected to its neighbouring cells. This regularity and local interconnection are favoured by parallel architectures.

Furthermore, a systolic array exploits both pipelining and parallel processing to increase computing power. This architecture requires that input data is fed into the systolic array in a skewed manner. It will be described later in detail.

However, one of the disadvantages of a systolic array is that it requires a global system clock to synchronize the activities of all the processing elements in a systolic array. The provision of the global system clock signal to all the processing elements requires a global interconnection, in contrast to a desirable local interconnection. Furthermore, the propagation delay of the global clock signal over the global interconnection results in a significant variation in the arrival of the global clock to processing cells. Hence, this variation has to be accounted for very carefully. To overcome this variation, wavefront arrays were developed by S.Y. Kung [29],[30]. These arrays use a locally data-driven control.

Processing cells in a systolic array are not always completely utilized, because some processing cells sit idle waiting for data, while other processing cells are actively processing. This occurs at the beginning and end of processing. The idle time can be minimized by initiating new tasks as soon as processing cells in the systolic array are available, even before the current task is completely

finished.

A systolic array may have a long time delay between the loading of input and the unloading of output, because data may have to travel through a large number of processing cells, and because input data is fed into the systolic array in a skewed manner. The term, latency, is defined as the time between the loading of input and the unloading of output, and it is used to express the amount of delay. When a short latency is required and the advantages of systolic arrays are not significant, a parallel system with global interconnection, shown in Figure 1.3, may be preferrable.

We describe systolic arrays for a multiplication of two matrices and the QR decomposition in the following sections to illustrate the characteristics of parallel processing, and in particular systolic array processing. A matrix–matrix multiplication and the QR decomposition are very commonly used in parallel implementaions of signal processing algorithms, and are also used in the parallel implementations of the KF presented in this thesis.

3.2.1 Systolic Array for Matrix–Matrix Multiplication

In this section, we show a systolic array for the multiplication of two matrices, C = A B [29]:

$$\begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \\ c_{41} & c_{42} & c_{43} & c_{44} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix}$$

Without losing generality, matrices A and B are assumed to be 4–by–4. Figure

3.1 (a) depicts a 2–dimensional systolic array designed to perform the above matrix multiplication. This systolic array consists of only one type of processing cell whose function is defined in Figure 3.1 (b).

To understand how this architecture works, we examine how the element c_{11} of matrix C is calculated at the processing cell p_{11} in Figure 3.1 (a). This cell receives the first row of matrix A from the top and the first column of matrix B from the left. The following operations are performed at each cycle:

a) Receive one element of the first row of A and the corresponding element in the first column of B.
b) Multiply the elements received in step (a).
c) Accumulate the product of multiplication in step (b).
d) Pass the two elements of A and B, received in step (a), to the neighbouring cells.

After the above operations are repeated 4 times, c_{11} will be produced at cell p_{11}.

Similarly, the other cells in the systolic array receive the appropriate row of A and column of B, and produce a vector–vector product corresponding to an element of C. Note that input data is reused in calculating other elements in C as it goes through processing cells. For example, the first row of A is used at the cell p_{11} to produce c_{11} and is reused to generate c_{12} at cell p_{12}. However, the first row of A arrives at cell p_{12} one cycle later than it does at cell p_{11}. This one cycle delay requires that the second column of matrix B arrives at cell p_{12} one cycle later than the first column of matrix B arrives at cell p_{11}. This is achieved by introducing a zero element in the input data stream of the second

column of B. Similarly, the other rows of A and columns of B enter the systolic array in a skewed manner for synchronization.

When the matrices A and B are n–by–n, the multiplication of these two matrices on a single processor requires n^3 cycles. Here the matrices are assumed to be n–by–n for generality, instead of 4–by–4, and both multiplication and addition are assumed to take one cycle to complete. On the two–dimensional n–by–n systolic array, this matrix–matrix multiplication requires 3n–2 cycles from the time matrices enter the systolic array to the time the multiplication is completed. It indicates that the systolic array decreases the total computation by two orders of magnitudes:

$$n^3/(3n-2) = o(n^2).$$

It also indicates that processing elements are utilized about 33%, since if all the n^2 cells in the n–by–n systolic array are always entirely utilized, then the matrix multiplication would require only n cycles instead of 3n–2 cycles. The reason for the 33% utilization is that in some situations processing elements sit idle waiting for data, while other processing elements are actively multiplying and accumulating. However, the 33% utilization can be improved by initiating new multiplications as soon as processors in the systolic array are available, even before the matrix multiplication is completed.

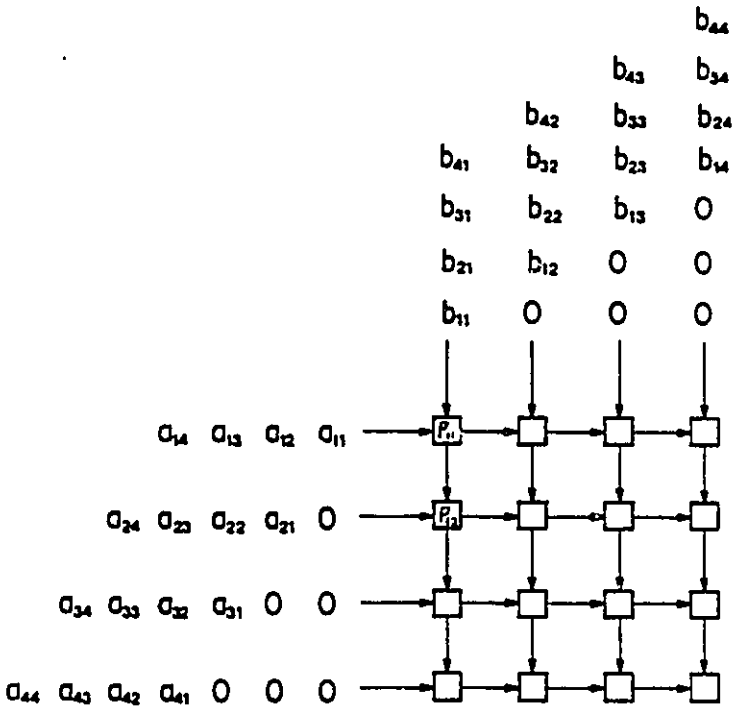
The total required time for the matrix multiplication, 3n–2 clock cycles, instead of n clock cycles, indicates that the latency of a systolic array is not negligible. This is due to the skew introduced in the input data stream, and the fact that the input matrices have to travel through n–1 cells clock to reach the processor at the bottom right corner of the systolic array. They each account for

n–1 clock cycle delay. The remaining n clock cycles (n=(3n–2)–(n–1)–(n–1)) is required to perform n multiplications for each element of matrix C.

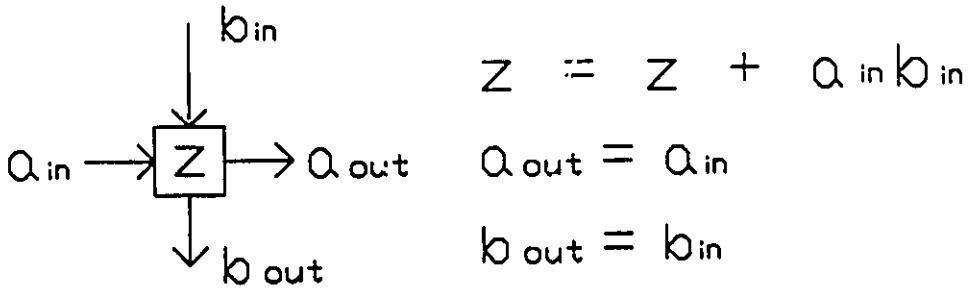
The systolic architecture presented in this section displays desirable characteristics of parallel architectures. Processors and data flow are simple and regular, and a local interconnection is used for inter–processor communication. Furthermore, a high degree of pipelining and parallelism are exploited. These advantages are achieved by utilizing the fundamental nature of data flow in a multiplication of two matrices, C = A B: the element of matrix C is the inner product of the row vector of A and the column vector of B.

A multiplication of two matrices cannot be implemented on a general–purpose pipelining or parallel processing system as efficiently as on the systolic array presented in this section, because the former is not optimized for this special use. Since the latter is designed for a particular use, it is limited in its use. Nevertheless, it is found that other types of matrix operations such as matrix–vector and vector–vector multiplications can be easily implemented on this systolic array [28].

In summary, a systolic array is a very efficient special–purpose architecture which has various desirable properties of parallel architectures.



(a)



(b)

Figure 3.1 Systolic array for matrix–matrix multiplication
a) structure of an array
b) function of a cell

3.2.2 Systolic Array for the QR decomposition.

A matrix A can be expressed as the product of a matrix Q with orthonormal columns and an invertible upper triangular matix R:

$$A = Q\ R$$

This decomposition, known as the QR decomposition ([17],[47]), arises in many applications such as eigenvalue decomposition ([17],[47]), least square methods in filtering [18], and square root Kalman filtering ([4],[25]). Furthermore, in the extended square root covariance filter for tracking applications described in Chapter 2, Equations (2.26) and (2.29) require an orthogonal upper–triangularization. This computationally demanding triangularization may be performed by using various algorithms such as Gram–Schmidt orthogonalization, Householder transformation, and Givens rotation ([16],[17],[47]).

The QR decomposition by Givens rotation is preferred in parallel design, because of the simple, regular nature of the Givens rotation. Figure 3.2 (a) shows a systolic architecture designed to perform the QR decomposition using Givens rotation. It was proposed by Gentleman and H.T. Kung [15].

This triangular systolic array consists of two types of processing cells: internal cells represented by squares and boundary cells represented by circles. The arithmetic functions of thses cells are defined in Figures 3.2 (b) and 3.2 (c). In this architecture, the matrix to be orthogonally triangularized, A, enters the systolic array from the top, in an order from the top row of matrix A to the bottom of matrix A.

After receiving the first row of matrix A, the first row of the systolic array annihilates the first element of every bottom (newly arrived) row of matrix

A, and outputs results to the second row of the systolic array. Similarly, the second row of the systolic array annihilates the second element of the bottom row of A which it receives from the first row of the systolic array, and outputs results with zeroes in the first and second entries to the third row of the systolic array. The rest of the systolic array performs similar operations to annihilate appropriate entries. The final product of these operations is an orthogonally triangularized matrix.

Like the systolic array for a multiplication of two matrices in the previous section, the systolic array for the QR decompostion presented here is very efficient and shows the desirable properties of a parallel architecture such as modularity, regularity, local interconnection, and a high degree of pipelining and parallelism. These features are achieved by utilizing the fundamental nature of data flow in the QR decomposition. Any general-purpose pipelining or parallel system cannot be as efficient for the QR decomposition as the systolic array presented in this section, because the former is not optimized for the QR decomposition.

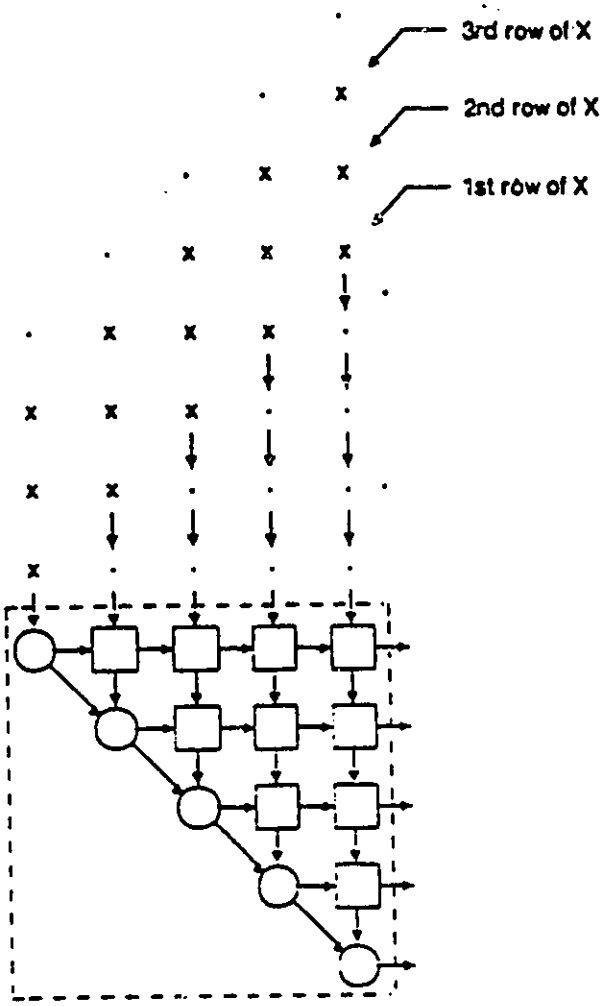
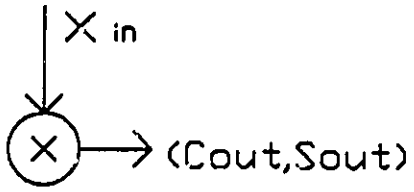


Figure 3.2 Systolic array for the QR decomposition
a) structure of an array
b) function of a boundary cell
c) function of an international cell



if $x = 0$ then

$c = 1$

$s = 0$

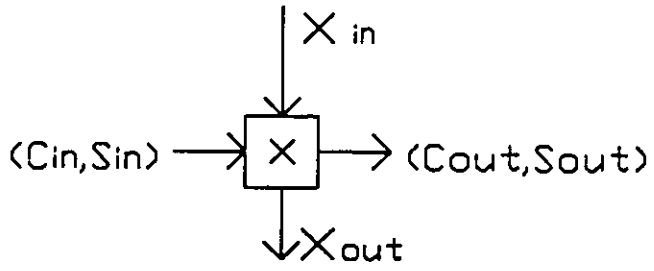
else

$$c_{out} = x / \sqrt{x^2 + x_{in}^2}$$

$$s_{out} = x_{in} / \sqrt{x^2 + x_{in}^2}$$

$$x = \sqrt{x^2 + x_{in}^2}$$

(b)



$$X_{out} = -S_{in} X + C_{in} X_{in}$$

$$X = C_{in} X + S_{in} X_{in}$$

(c)

3.3 Systolic Implementation of the Kalman Filter

There have been a number of parallel implementations of the Kalman filter proposed in the literature. Papadourakis and Taylor [37], and Yeh [52] proposed parallel implementations of the conventional covariance KF by decomposing the KF equations into a set of matrix–matrix, matrix–vector, vector–vector operations, and matrix inversion. One of the difficulties of this approach is the implementation of a matrix inversion on a systolic array. Papadourakis and Taylor [37] used an iterative approach which takes a significant number of cycles. To simplify a matrix inversion and other matrix operations, Yeh [52] employed the Fadeeva algorithm [9]. The Fadeeva algorithm avoids the direct matrix inverse computation, and can be easily implemented on a systolic architecture.

A parallel architecture for the measurement update part of the SRCF has been presented by Joves and Kailath [21]. They used a square–root free QR decompsosition technique. Sung and Hu [48] proposed an architecture for the complete SRCF, using a set of dedicated processor arrays. Gaston and Irwin [14] developed parallel implementations of the SRCF using a QR systolic array. Chen and Yao [6], S.Y. Kung and Hwang [31], and Gaston and Irwin [13] developed architectures for the modified SRCF which estimates the accuracy of the state estimates in terms of the inverse of the square root of the error covariance matrix.

All the architectures developed so far implement different versions of the standard Kalman filter for general filtering applications. None of them implements the extended KF for tracking applications. Furthermore, in developing these architectures, the special properties of the applications to which they are applied are not utilized. Hence, these architectures are general enough to be employed for

any KF application. However, the optimization of a parallel architecture through the use of application–specific properties may be desirable, even though the use of an architecture would be limited for specific applications. The reasons are that general–purpose parallel architectures have a significant system overhead to handle all applications, and that the use of properties associated with application generally simplify architectures significantly.

In this section, we briefly review three parallel architectures for different versions of the Kalman filter. These architectures examplify what has been done for parallel implementations of the KF. We present an architecture for the covariance KF by Papadourakis and Taylor [37] in Section 3.3.1, and another architecture for the SRCF by Sung and Hu [48] in Section 3.3.2. Finally, an architecture for the modified SRCF, proposed by S.Y. Kung and Hwang [31], is described in Section 3.3.3.

3.3.1. Parallel Implementation of the Covariance Kalman Filter

To save computational requirements, Papadourakis and Taylor [37] first determined common terms in the covariance KF equations, and then reformulated the KF equations such that the common terms are calculated only once and reused whenever necessary.

The reformulated covariance KF equations are

$$a = \phi(k)\ K(k) \tag{3.1}$$

$$X(k+1) = \phi(k)\ \hat{X}(k) + a\ [Z(k) - H(k)\ \hat{X}(k)] \tag{3.2}$$

$$P(k+1) = (\phi(k) - a\ H(k))\ P(k)\ \phi^{T}(k) + G(k)\ Q(k)\ G^{T}(k) \tag{3.3}$$

$$b = P(k)\ H^{T}(k) \tag{3.4}$$

$$K(k) = b\ (\ R(k) + H(k)\ b\ )^{-1} \tag{3.5}$$

Note that the common term, a = ϕ(k) K(k), is calculated in Equation (3.1), and used in two equations (3.2) and (3.3). Similarly, the common term, b = P(k) H^T(k), is calculated in Equation (3.4), and used twice in Equation (3.5).

Papadourakis and Taylor decomposed the above KF equations into a set of matrix–matrix, matrix–vector, vector–vector operations and a matrix inversion, and then they determined the sequence of operations to minimize computational time. They avoided loading the same matrix twice and loading the output of operation immediately after it is calculated, because it takes time to unload and to load matrices. An iterative algorithm [22] was used to invert a matrix in Equation (3.5), but this algorithm is very computationally demanding.

All the decomposed matrix operations for the KF were implemented on a single systolic array. This array has the same structure as one in Figure 3.1. It consists of n–by–n processing elements whose functions are slightly different from the functions of the processing cells in Figure 3.1. The letter n denotes the number of state elements.

Although Papadourakis and Taylor minimized the computational requirements of the covariance KF by eliminating the repetition of the same calculations and by finding the optimal sequence of operations in the KF equations, their implementation requires a significant number of clock cycles to complete one iteration of filtering. The updates of the state estimate error matrix P(k) and the calculation of the Kalman gain K(k) together require more than 16n clock cycles. The reason for the demanding computational requirements is that they used one systolic array, and the matrix inversion is iteratively performed. The matrix inversion requires close to half of the total number of clock cycles [37].

3.3.2. Parallel Implementation of the Square Root Covariance Filter

To increase parallelism and to determine the optimal sequence of calculations, Sung and Hu [48] examined dependencies among matrices and vectors in the square root covariance filter (SRCF) and found that the SRCF can be divided into two loosely related sets of equations as follows:

Set A

$$K(k) = G^T(k) \,/\, F^T(k) \tag{2.15}$$

$$\hat{X}(k) = \hat{X}(k|k-1) + K(k) \,(\, Z(k) - H(k)\, \hat{X}(k|k-1) \,) \tag{2.16}$$

$$\hat{X}(k+1|k) = \phi(k)\, \hat{X}(k) \tag{2.18}$$

Set B

$$\begin{bmatrix} F(k) & G(k) \\ 0 & S^T(k) \end{bmatrix} = Q_1 \begin{bmatrix} V^T(k) & 0 \\ S^T(k|k-1)H^T(k) & S^T(k|k-1) \end{bmatrix} \tag{2.14}$$

$$\begin{bmatrix} S^T(k+1|K) \\ 0 \end{bmatrix} = Q_1 \begin{bmatrix} S^T(k)\, \phi^T(k) \\ U^T(k) \end{bmatrix} \tag{2.17}$$

The equations in Set A compute state vectors $\hat{X}(k)$ and $\hat{X}(k|k-1)$, whereas the equations in Set B update the square root of the state estimate error covariance matrix $P^{1/2}(k|k+1)$. The Kalman gain, calculated in Equation (2.15) in Set A, is defined in terms of G(k) and F(k) which are determined in Equation (2.14) in Set B. This exhibits dependencies between the two sets of equations. Sets A and B can be executed in parallel, as Set A receives G(k) and F(k) from Set B.

Sung and Hu avoided using a computationally demanding iterative algorithm to invert a matrix in Equation (2.15) by performing the computations

$$K(k) = G^T(k)\ (F^T(k))^{-1} \tag{2.15}$$

$$\hat{X}(k) = \hat{X}(k|k-1) + K^T(k)\ (\ Z(k) - H(k)\hat{X}(k|k-1)\ ) \tag{2.16}$$

as follows:

a) Calculate $W(k) = Z(k) - H(k)\ \hat{X}(k|k-1)$

b) Calculate $Y(k) = (F^T(k))^{-1}W(k)$ by the method of back substitution using F(k) and W(k)

c) Calculate $\hat{X}(k) = \hat{X}(k|k-1) + G^T(k)\ Y(k)$

Based on the SRCF which are divided into two sets of equations, Sung and Hu proposed an architecture that consists of two parts. Figures 3.3 (a) and 3.3 (b) show a block diagram of the implementation of Set A and that of Set B respectively. Each part is composed of a number of systolic arrays which are represented as squares. Separate systolic arrays are employed for different operations in the SRCF, and they are connected according to the data flow.

Data are passed from one systolic array to the next without going through the same array more than once during an iteration. This sequential connection is useful for pipeline processing, but it prevents systolic arrays from being used more than once during an iteration. Note that matrices F(k) and G(k), needed to calculate K(k) in Set A, are passed from an architecture in Figure 3.3 (b) to an architecture in Figure 3.3 (a).

Sung and Hu employed a triangular systolic array, shown in Figure 3.2, for the time update of state estimate error covariance matrix P(k) using the QR decomposition. When the number of state elements is n and the number of measurements is m, the size of a triangular systolic array needd for the time

update is n–by–n with n(n+1)/2 processing elements. For the measurement update, a trapezoidal systolic array of m rows of processing elements with n+m processing elements on the top row and m+1 elements at the bottom is employed. This trapezoidal systolic array, shown in Figure 3.4, is obtained by the utilization of sparse nature of an input matrix in the measurement update.

Sung and Hu's implementation of the SRCF requires max[(4n+2m, 2n+4m–2)] time steps to complete one SRCF iteration. The processing of Equations (2.14), (2.15), and (2.17) requires 4n+m–1 time steps.

The architecture of Sung and Hu, and the architecture of Papadourakis and Taylor differ a great deal from each other. The latter architecture shares a systolic array for all the matrix operations, whereas the former architecture uses seperate systolic arrays for different equations and connects systolic arrays according to the data flow. Hence the Sung–Hu architecure requires more systolic arrays than the Papadourakis–Taylor architecture. However, the Sung–Hu architecture requires a smaller number of clock cycles to complete one KF iteration than the Papadourakis–Taylor architecture. The reasons are that in the former implementaion an iterative algorithm is not used to invert a matrix, and that parallelism is achieved by dividing the KF equations into two groups.

3.3.3. Parallel Implementation of the modified Square Root Covariance Filter

The modified square root covariance filter is a version of the Kalman filter that expresses the estimated accuracy of state estimates in terms of the inverse of the square root of an error covariance matrix ([4],[25]), as described in Chapter 1. Paige and Saunders [36] have reformulated the modified SRCF by using an orthogonal transformation. In contrast to the conventional modified SRCF, in the Paige and Saunder's modified SRCF, the inverse of a transition

matrix $\phi(k)$ is not required, and the inverse of the square root of a covariance matrix calculated in measurement update is directly used without multiplication by $\phi(k)$ in time update. As a result, both time and measurement updates can be performed continuously in the same orthogonalization. However, Paige and Saunder's modified SRCF requires additional computation. The whitening procedure, diagonalizing the system dynamics noise variance matrix Q(k) and the measurement noise variance matrix R(k), has to be applied to the transition and measurement matrices, and to measurements.

In developing a parallel architecture for the KF, S.Y. Kung and Hwang [31] selected Paige and Saunder's SRCF over other versions of the KF. They employed a systolic array for the QR decomposition, described in Section 3.2.2, to implement the required orthogonal triangularization, and two triangular systolic arrays to whiten the transition and measurement matrices.

In S.Y.Kung and Hwang's design approach, the implementation has be significantly simplified by the selection of a suitable algorithm for parallel architectures, and by the reformulation of algorithms to be more easily adaptable to parallel architectures.

Figure 3.3 Block diagram of Sung–Hu architecture

a) Set A

b) Set B

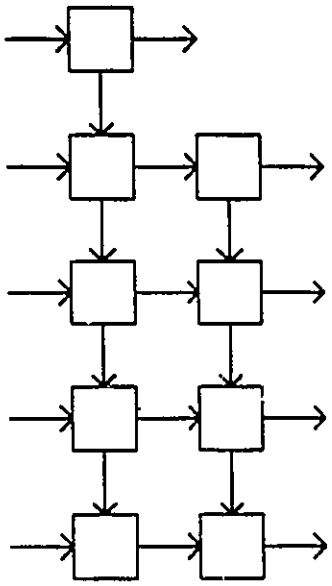


Figure 3.4 Measurement update of P(k)

3.4 Summary

In this chapter, we have described parallel architecture design principles which can be used as guidelines in designing parallel architectures. Parallel architectures should be regular and modular to reduce design errors, time, and cost. Processing elements in parallel architecture should be connected to their neighboring cells for the increased computing power to be balanced with the data input and output rate.

To gain insight into and appreciation of parallel processing, we have studied two systolic arrays. These two systolic arrays are found to have various desirable properties of parallel architecture; regularity, modularity, and local interconnection. However, these arrays are found to have a nonnegligible latency, because data may have to travel through a large number of processing cells, and because input data are fed into systolic arrays in a skewed manner.

We have reviewed three parallel architectures for different versions of the Kalman filter. Papadourakis and Taylor minimized computational requirements by reformulating the covariance KF equations such that the common terms are calculated only once and reused whenever necessary. However, their architecture is not efficient, because the required matrix inversion is performed iteratively and all the required operations are calculated sequentially on a single systolic array; no parallelism among KE equations is thus explored.

Sung and Hu examined dependencies among matrices and vectors in the square root covariance filter and found that the SRCF equations can be separated into two loosely connected sets of equations. Based on this, Sung and Hu proposed an architecture that consists of two major blocks which can be executed in parallel; parallelism is thus enhanced. Sung and Hu eliminated the iterative calculation of a matrix inversion by rearranging Equations (2.51) and (2.16).

The Sung–Hu architecture is more efficient than the Papadourakis–Taylor architecture, because the former architecture exploits parallelism among KF equations and does not use an iterative algorithm to perform a matrix inversion.

In developing a parallel architecture for the KF, S.Y. Kung and Hwang [31] selected Paige and Saunder's modified SRCF and proposed an architecture which consists of three systolic arrays: one array for the QR decomposition and two arrays for whitening. S.Y. Kung's architecture is less complex than Sung and Hu's architecture. The reason for this is that Paige and Saunder's SRCF is more suitable for parallel implementation than the conventional SRCF. This means that the adaptability of a filter to a parallel architecture shoud be considered in selecting a version of the KF for a parallel implementation.

CHAPTER 4

SIMPLIFICATION OF THE KALMAN FILTER FOR TRACKING APPLICATIONS

As described in Chapters 2 and 3, the Kalman filter is computationally demanding, and its parallel implementations are complex. This is mainly due to the fact that the KF requires in each iteration a large number of matrix–matrix and matrix–vector multiplications, and a matrix inversion.

We have shown in Chapter 3 that the exploration of parallelism and the reformulation of the KF equations simplifies parallel implementations for the KF. In this chapter, we present methods of simplifing the extended KF and the extended SRCF. These methods not only reduce computational requirements but also increase parallelism for parallel implementations.

It was found that the extended covariance KF can be simplified through the use of a decoupling technique ([2], [8], [35]). The decoupling technique results in the elimination of a computationally demanding matrix inversion and the propagation of three decoupled $(\frac{n}{3})$–dimensional covariance matrices P(k)'s instead of one coupled n–dimensional covariance matrix for 3–dimensional tracking, where n is the number of state elements in the coupled KF. The propagation of three decoupled covariance matrices require less computation than that of one coupled covariance matrix.

We have extended the use of the decoupling technique to the extended

square root covariance filter (ESRCF) to simplify the ESRCF. The decoupled ESRCF updates three decoupled $(\frac{n}{3})$-dimensional square roots $P^{1/2}(K)$'s of the covariance matrix P(k), instead of 1 coupled n-dimensional $P^{1/2}(k)$ for 3-dimensional tracking, where n is the number of state elements.

Further, properties of the tracking KF can be used to simplify the implementation of the KF. Specifically, a typical transition matrix is a band, sparse matrix, and the measurement matrix is sparse. A band matrix is a matrix whose nonzero elements are all concentrated near the main diagonal. A sparse matrix is a matrix whith most of its elements equal to zero.

In Section 4.1 we discuss the decoupling technique, and in Section 4.2 we describe properties of the tracking KF. We then simplify the Extended Covariance KF (ECKF) and the Extended SRCF (ESRCF) by applying the decoupling technique and special properties in Sections 4.3 and 4.4, respectively. In those two sections, we also discuss the computational reduction afforded by the simplification. In Section 4.5, we compare the performance of the simplified KF with that of the standard KF on the basis of tracking accuracy. Finally, in Section 4.6, we summarize this chapter.

4.1 The Decoupling Technique

In this section, we explain how the state estimate error covariance matrix P(k) is coupled in one coordinates system and how it becomes decoupled in another coordinate system. Then, we develop relationships between these coordinate sytems. These relationships are used in decoupling the KF. We assume a 2-dimensional tracking for simplicity without a loss in generality.

Before discussing the decoupling technique, we first define three different coordinate systems: reference coordinate, line-of-sight coordinate, polar coordinate

systems.

The reference Cartesian coordinate system is defined as a Cartesian coordinate system with its origin at the location of a radar used for tracking, and its x–axis set arbitrarily. The y–axis is set automatically perpendicular to the x–axis, once the x–axis is set. This coordinate system is depicted in Figure 4.1. In this system, the location of a target is expressed by a pair of numbers, (x,y). This coordinate system does not vary over time as the target moves.

The line–of–sight (LOS) Cartesian coordinate system is defined as a Cartesian coordinate system with the location of the radar defined as its origin and the line of sight to the target defined as its x–axis. This coordinate sytem is depicted in Figure 4.2. In this system, the location of the target is expressed by a pair of numbers, (x_o,y_o). However, y_o is always zero, because the target is always on the x–axis. In contrast to the reference Cartesian coordinate system, this coordinate system varies as the target moves.

The polar coordinate system is based on its origin set at the location of the radar and the reference line coincides with the x–axis of the reference Cartesian coordinate system, as shown in Figure 4.3. In this third coordinate system, the location of the target is expressed in terms of its distance r, known as range, from the origin and the counterclockwise angle θ, known as a bearing angle, made by the line of sight and the reference line. This coordinate system does not vary as the target moves. We have defined so far three different coordinate systems for two–dimensional tracking. The coordinate systems for three–dimensional tracking can be similarly developed.

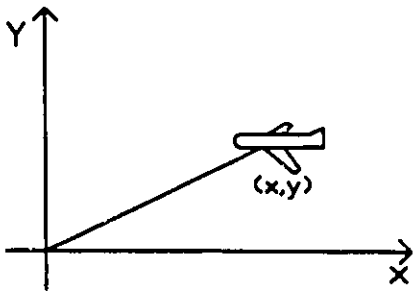


Figure 4.1 The reference Cartesian coordinates system

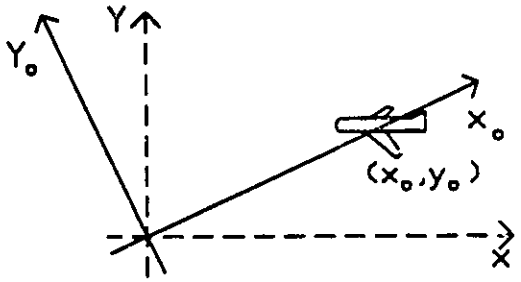


Figure 4.2 The line–of–sight Cartesian coordinates system

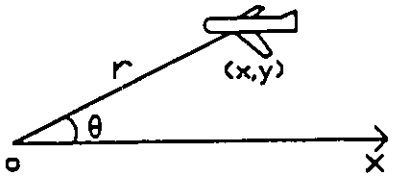


Figure 4.3 The polar system

We indicate the type of a coordinate system by a subscript. The subscripts o and p denote LOS coordinate and polar coordinate systems, respectively. However, we do not use any subscript to denote a reference Cartesian coordinate system.

In a tracking system of interest, radar measurements are ordinarily made in polar coordinates of range and bearing angle. If we assume that the measurement errors in range and bearing angle are independent, which is valid, then the measurement error covariance matrix $R_p(k)$, in the polar coordinate form, is

$$R_p(k) = \begin{bmatrix} \sigma_r^2 & 0 \\ 0 & \sigma_\theta^2 \end{bmatrix} \tag{4.1}$$

where σ_r^2 and σ_θ^2 are the variances of the range and bearing angle measurement errors, respectively. Note that the subscript p denotes polar form. The measurement error covariance matrix $R_o(k)$ in LOS Cartesian coordinates is defined as

$$R_o(k) = \begin{bmatrix} \sigma_{o-x}^2 & \sigma_{o-xy}^2 \\ \sigma_{o-xy}^2 & \sigma_{o-y}^2 \end{bmatrix} \tag{4.2}$$

where σ_{o-x}^2, σ_{o-xy}^2, and σ_{o-y}^2 can be expressed in terms of σ_r^2, σ_θ^2, and r^2 (where r denotes range). Note that the subsrcipt o denotes the LOS Catesian coordinate system. The following relationship:

$$\sigma_{o-x}^2 = \sigma_r^2 \tag{4.3}$$

$$\sigma^2_{o-y} = (r\sigma_\theta)^2 \tag{4.4}$$

$$\sigma^2_{o-xy} = 0 \tag{4.5}$$

results in

$$R_o(k) = \begin{bmatrix} \sigma_r^2 & 0 \\ 0 & (r\sigma_\theta)^2 \end{bmatrix} \tag{4.6}$$

The term σ^2_{o-xy} is zero, because σ_r^2 and σ_θ^2 are independent.

We now develop from $R_o(k)$ the measurement error covariance matrix R(k) in the reference Cartesian coordinates using the relation:

$$x = r\cos\theta \tag{4.7}$$

$$y = r\sin\theta \tag{4.8}$$

where r and θ denote the range and bearing angle of a target, as shown in Figure 4.3. We first express σ_x and σ_y in terms of σ_{o-x} and σ_{o-y} as follows:

$$\begin{aligned} \sigma_x &= \frac{\partial x}{\partial r}\sigma_r + \frac{\partial x}{\partial \theta}\sigma_\theta \\ &= \cos\theta\ \sigma_r - r\ \sin\theta\ \sigma_\theta \end{aligned} \tag{4.9}$$

$$\begin{aligned} \sigma_y &= \frac{\partial y}{\partial r}\sigma_r + \frac{\partial y}{\partial \theta}\sigma_\theta \\ &= \sin\theta\ \sigma_r + r\ \sin\theta\ \sigma_\theta \end{aligned} \tag{4.10}$$

Equations (4.9) and (4.10) may be expressed in a matrix form, as shown below:

$$\begin{bmatrix} \sigma_x \\ \sigma_y \end{bmatrix} = \begin{bmatrix} \cos\theta & -r\sin\theta \\ \sin\theta & r\cos\theta \end{bmatrix} \begin{bmatrix} \sigma_r \\ \sigma_\theta \end{bmatrix}$$

$$= \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \sigma_r \\ r\sigma_\theta \end{bmatrix}$$

$$= \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \sigma_{o-x} \\ \sigma_{o-y} \end{bmatrix} \qquad (4.11)$$

Let

$$F_1 = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \qquad (4.12)$$

then

$$\begin{bmatrix} \sigma_x \\ \sigma_y \end{bmatrix} = F_1 \begin{bmatrix} \sigma_{o-x} \\ \sigma_{o-y} \end{bmatrix} \qquad (4.13)$$

R(k) is developed in terms of $R_o(k)$ as follows:

$$R(k) = \begin{bmatrix} \sigma_x^2 & \sigma_{xy}^2 \\ \sigma_{xy}^2 & \sigma_y^2 \end{bmatrix}$$

$$= \begin{bmatrix} \sigma_x \\ \sigma_y \end{bmatrix} \begin{bmatrix} \sigma_x & \sigma_y \end{bmatrix}$$

$$= F_1 \begin{bmatrix} \sigma_{o-x} \\ \sigma_{o-y} \end{bmatrix} \begin{bmatrix} \sigma_{o-x} & \sigma_{o-y} \end{bmatrix} F_1^T$$

$$= F_1 \begin{bmatrix} \sigma_{o-x}^2 & 0 \\ 0 & \sigma_{o-y}^2 \end{bmatrix} F_1^T$$

$$= F_1 \, R_o \, F_1^T \tag{4.14}$$

As a result, we find that

$$\sigma_x^2 = \sigma_r^2 \cos^2\theta(k) + r^2(k) \sin^2\theta(k) \, \sigma_\theta^2 \tag{4.15}$$

$$\sigma_y^2 = \sigma_r^2 \sin^2\theta(k) + r^2(k) \cos^2\theta(k) \, \sigma_\theta^2 \tag{4.16}$$

$$\sigma_{xy}^2 = \frac{1}{2} \sin 2\theta(k) \, [\, \sigma_r^2 - r^2(k) \, \sigma_\theta^2 \,] \tag{4.17}$$

It is important to note that R(k) has a nonzero cross product term σ_{xy}^2. This nonzero term indicates that in the reference Cartesian coordinate system the measurement errors in the x-axis direction are related to those in the y-axis direction. The measurement error covariance matrix R(k) is said to be coupled in this case.

Note that in contrast to R(k), the measurement error covariance matrix

$R_o(k)$ for the LOS coordinate system has a zero cross product σ^2_{o-xy}. That is, $R_o(k)$ is decoupled.

Similar to R(k), the state error covariance matrix P(k) can also be decoupled in the LOS frame. The relationship between the state estimate error covariance matrix $P_o(k)$ for the LOS coordinate frame and that P(k) for the reference coordinate frame is found in the same way that the relationship between R_o and R(k) is developed. This relationship is as follows:

$$P(k) = \begin{bmatrix} F_1(k) & 0 & 0 \\ 0 & F_1(k) & 0 \\ 0 & 0 & F_1(k) \end{bmatrix} P_o(k) \begin{bmatrix} F_1(k) & 0 & 0 \\ 0 & F_1(k) & 0 \\ 0 & 0 & F_1(k) \end{bmatrix}^T \tag{4.18}$$

For 2–dimensional tracking, we have

$$F_1(k) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{4.19}$$

where θ denotes azimuth angle. θ_A and θ_E will be hereafter used to denote azimuth and elevation angles, respectively. For 2–dimensional tracking, θ, which has been used to denote bearing angle, will be replaced by θ_A.

For 3–dimensional tracking, we have

$$F_1(k) = \begin{bmatrix} \cos\theta_A\cos\theta_E & -\sin\theta_A\cos\theta_E & -\cos\theta_A\sin\theta_E \\ \sin\theta_A\cos\theta_E & \cos\theta_A\cos\theta_E & -\sin\theta_A\sin\theta_E \\ \sin\theta_E & 0 & \cos\theta_E \end{bmatrix} \tag{4.20}$$

It should be noted that P(k) and $P_o(k)$ are based on the rearranged state vector $X(k) = [\, x(k)\ y(k)\ \dot{x}(k)\ \dot{y}(k)\ \ddot{x}(k)\ \ddot{y}(k)\,]$, instead of $[x(k)\ \dot{x}(k)\ \ddot{x}(k)\ y(k)\ \dot{y}(k)\ \ddot{y}(k)\,]$. This is done in order to express P(k) in terms of $P_o(k)$ and $F_1(k)$ in a simpler form.

Similarly, the gain matrix K(k) for the reference Cartesian coordinates can be obtained from the gain matrix $K_o(k)$ for the LOS frame, using the coordinate transformation as follows:

$$
\begin{aligned}
K(k) &= P(k|k-1)\, H^T(k)\, (\, H(k)\, P(k|k-1)\, H^T(k) + R(k)\,)^{-1} \\
&= P(k)\, H^T(k)\, R^{-1}(k) \\
&= \begin{bmatrix} F_1(k) & 0 & 0 \\ 0 & F_1(k) & 0 \\ 0 & 0 & F_1(k) \end{bmatrix} P_o(k) \begin{bmatrix} F_1(k) & 0 & 0 \\ 0 & F_1(k) & 0 \\ 0 & 0 & F_1(k) \end{bmatrix}^T H^T(k)\, R^{-1}(k) \\
&= \begin{bmatrix} F_1(k) & 0 & 0 \\ 0 & F_1(k) & 0 \\ 0 & 0 & F_1(k) \end{bmatrix} P_o(k)\, H_o^T(k)\, R^{-1}(k) \\
&= \begin{bmatrix} F_1(k) & 0 & 0 \\ 0 & F_1(k) & 0 \\ 0 & 0 & F_1(k) \end{bmatrix} K_o(k)
\end{aligned}
\tag{4.21}
$$

The decoupling technique discussed here requires that the LOS frame does not vary significantly over time [35]. Fortunately, this condition is usually met in typical radar tracking applications ([2],[8],[35]).

Time and measurement updates of the decoupled $P_o(k)$ in the LOS frame

can be performed separately for each axis, whereas the updates of the coupled P(k) in the reference Cartesian frame have to be performed together for all the axes. The updates of $P_o(k)$ separately for each axis are found to require less computation than those of P(k). This computational reduction is discussed in detail in Section 4.3. The fact that $P_o(k)$ can be separately updated for each axis indicates that the updates can be performed simultaneously on a parallel architecture. Since the state of a target, such as its position and velocity, is required in the reference Cartesian frame, X(k) should be expressed in this frame. Hence, to simplify the computation of the KF, some of the KF equations should be performed in the LOS frame, others in the reference Cartesian frame, and the decoupling should be accounted for by the Jacobian transformation between the LOS and reference Cartesian frames, defined in Equations (4.19) and (4.20). This separation is summarized below:

a) Processing in the LOS frame:

Time update of $P_o(k)$

Measurement update of $P_o(k)$

Calculation of the Kalman gain $K_o(k)$

b) Transformation of the Kalman gain:

$$K(k) = \begin{bmatrix} F_1(k) & 0 & 0 \\ 0 & F_1(k) & 0 \\ 0 & 0 & F_1(k) \end{bmatrix} K_o(k)$$

c) Processing in the reference Cartesian frame:

Estimate a state vector X(k)

Predict a state vector X(k+1|k)

4.2 Special Properties of the Tracking KF

A typical state vector X(k) for 3-dimensional tracking is defined in Section 2.1 as follows:

$$X(k) = \begin{bmatrix} x(k) \\ \dot{x}(k) \\ \ddot{x}(k) \\ y(k) \\ \dot{y}(k) \\ \ddot{y}(k) \\ z(k) \\ z(k) \\ \ddot{z}(k) \end{bmatrix} = \begin{bmatrix} \text{x-position} & \text{at time k} \\ \text{x-velocity} & \text{at time k} \\ \text{x-acceleration} & \text{at time k} \\ \text{y-position} & \text{at time k} \\ \text{y-velocity} & \text{at time k} \\ \text{y-acceleration} & \text{at time k} \\ \text{z-position} & \text{at time k} \\ \text{z-velocity} & \text{at time k} \\ \text{z-acceleration} & \text{at time k} \end{bmatrix}$$

A typical transition matrix $\phi(k)$ which relates state vectors X(k) and X(k+1) is found to be in Section as follows:

$$\phi_x = \phi_y = \phi_z = \begin{bmatrix} 1 & T & 0 \\ 0 & 1 & 1 \\ 0 & 0 & \rho \end{bmatrix} \tag{4.22}$$

where T is the sampling period, the subscripts x, y, and z denote axes, and ρ is a correlation coefficient for acceleration.

The correlation coefficient ρ for acceleration is used to specify the acceleration characteristics of a target. For example, when the maneuvering of a target at one sampling instant is highly correlated with that at the previous sampling instant, the correlation coefficient ρ is set large. Hence, the transition matrix defined above can be used to track a target with various acceleration

characteristics by setting the correlation coefficient ρ accordingly. Note that the transition matrix in Equation (4.22) is a band, sparse matrix.

The measurement matrix H(k) in the reference Cartesian coordinate system is defined in Section 2.2.4 for 3-dimensional tracking as follows:

$$H(k) = \begin{bmatrix} \frac{x}{r} & \frac{y}{r} & \frac{z}{r} & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{-y}{(x^2+y^2)} & \frac{x}{(x^2+y^2)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{-xy}{r^2(x^2+y^2)^{1/2}} & \frac{-yz}{r^2(x^2+y^2)^{1/2}} & \frac{x^2+y^2}{r^2} & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

In the LOS frame whose x-axis is defined as the line of sight to the target, the measurement matrix H(k) becomes a sparse matrix as shown below:

$$H_o(k) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & r^{-1} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & r^{-1} & 0 & 0 \end{bmatrix} \tag{4.23}$$

where r denotes the target range. This $H_o(k)$ is much simpler than H(k).

The matrix $H_o(k)$ can be divided for each axis, as follows:

$$H_{o-x} = [\, 1 \quad 0 \quad 0 \,] \tag{4.24}$$

$$H_{o-y} = [\, r^{-1} \quad 0 \quad 0 \,] \tag{4.25}$$

$$H_{o-z} = [\, r^{-1} \quad 0 \quad 0 \,] \tag{4.26}$$

4.3 Simplification of the Extended Covariance KF

4.3.1 Simplification using the decoupling technique

The application of the decoupling technique to the Extended Covariance KF results in the decoupled extended covariance KF, summarized as follows:

a) Processing in the LOS frame computes $K_o(k)$, $P_o(k)$ and $P_o^-(k+1)$, as shown by

$$K_o(k) = P_o(k|k-1)\, H_o^T(k)\, (H_o(k)\, P_o(k|k-1)\, H_o^T(k) + R\,(k))^{-1} \tag{4.27}$$

$$P_o(k) = (\, I - K_o(k)\, H_o(k)\,)\, P_o(k|k-1) \tag{4.28}$$

$$P_o(k+1|k) = \phi(k)\, P_o(k)\, \phi^T(k) + Q_o(k) \tag{4.29}$$

b) Coordinate transformation computes K(k) from $K_o(k)$ by using the Jacobian transformation:

$$K(k) = \begin{bmatrix} F_1 & 0 & 0 \\ 0 & F_1 & 0 \\ 0 & 0 & F_1 \end{bmatrix} K_o(k) \tag{4.30}$$

where F_1 is defined for 2- and 3-dimensional tracking in Equations (4.19) and (4.20), respectively.

c) Processing in the reference Cartesian coordinate frame computes $\hat{X}(k)$ and $\hat{X}(k+1|k)$, as shown by

$$\hat{X}(k) = \hat{X}(k|k-1) + K(k)\,(\, Z(k) - h(\hat{X}(k|k-1))\,) \tag{4.31}$$

$$\hat{X}(k+1|k) = \phi(k)\, \hat{X}(k) \tag{4.32}$$

The examination of the decoupled ECKF equation (4.27) reveals that the decoupled ECKF requires a scalar division, instead of a matrix inversion. The elimination of a matrix inversion reduces the computational requirements significantly, because a matrix inversion is computationally very demanding.

The decoupled ECKF requires the propagation of m decoupled $(\frac{n}{m})$-dimensional covariance matrices instead of one n-dimensional covariance matrix, where n is the number of state elements in the coupled KF, and m is two for 2-dimensional tracking, or three for 3-dimensional tracking. These m decoupled covariance matrices for each axis can be propagated simultaneously; the decoupling technique thus increases parallelism.

Tables 4.1 (a) and 4.1 (b) list the number of operations required to perform one iteration of each KF equation for the coupled ECKF. Table 4.1 (a) is based on a general m-dimensional tracking with n state elements, whereas Table 4.1 (b) is based on a typical 3-dimensional tracking with 9 state elements. A matrix inversion, required for the coupled ECKF, is assumed to be implemented using an iterative algorithm [22]. Its coumputational requirements are listed separately from the KF equation (2.20) to demonstrate the effects of the decoupling technique more clearly. Note that the number of required operations for Equation (2.20) which is one of the ECKF equations is not included in Table 4.1 (a), because it is not possible to be expressed in terms of m and n.

Equation (2.21) requires a coordinate transformation from Cartesian to polar coordinates, h(X(k)). The components of h(X(k)) for 2-dimensional tracking are

$$r(k) = ((x(k))^2 + (y(k))^2)^{1/2} \tag{2.4}$$

$$\theta_A(k) = \tan^{-1} (y(k)/x(k)) \tag{2.5}$$

These two equations indicate that the coordinate transformation requires a square–root operation and an arc–tangent operation. However, these operations are difficult to implement, and yet commonly required to be evaluated in various digital signal processing applications.

Volder proposed the coordinate digital computer (CORDIC) algorithm and its implementation in [49] to perform 2–dimensional coordinate transformation and to compute trigonometric functions $\sin\theta$ and $\cos\theta$, given an angle θ. Volder's CORDIC algorithm is very easy to implement because it requires only shifting, adding, subtracting, and the recall of prestored constants. For more details, refer to Appendix A. We have found that Volder's CORDIC algorithm can be extended to 3–dimensional coordinate transformation by performing two CORDIC operations in sequence with scaling. The various uses of the CORDIC have been suggested by Walther [50].

In this thesis, we use the CORDIC operations to perform coordinate transformations and to evaluate trigonometric functions.

Tables 4.2 (a) and 4.2 (b) list the number of operations required to perform one iteration of each KF equation for the decoupled ECKF. Table 4.2 (a) is based on a general m–dimensional tracking with n state elements, whereas Table 4.2 (b) is based on a typical 3–dimensional tracking with 9 state elements.

A comparison of Tables 4.1 (a) and 4.2 (a) shows that the number of multiplications and additions required for the decoupled ECKF equations (4.27), (4.28), and (4.29), excluding the requirements for a matrix inversion, is reduced by approximately a factor of m^2 over that for the corresponding coupled ECKF equations (2.20), (2.22), and (2.23), where m denotes the number of tracking dimensions.

The reduction factor of approximately m^2 in the required number of

multiplications for Equations (4.27), (4.28), and (4.29) can be easily estimated by considering the matrix–matrix multiplications that are predominant in these equations. When the size of a matrix is n–by–n, a matrix–matrix multiplication requires n^3 multiplications. Hence, when the dimension of a matrix decreases by a factor m, the total number of required multiplications decreases by a factor of m^3. For the decoupled system, Equations (4.27), (4.28) and (4.29) have to be processed m times. As a result, the decoupled system requires m times as many matrix–matrix multiplications as the coupled system. Nevertheless, since the matrix dimension in the decoupled system is m times smaller than that for the coupled system, the overall multiplication requirement for matrix–matrix multiplications decrease by a factor of m^2 ($=m^3/m$). Since the required number of multiplications for the matrix–matrix multiplications which are prevalent in Equations (4.27), (4.28), and (4.29) decreases by a factor of m^2, the overall reduction in the required number of multiplications in Equations (4.27), (4.28), and (4.29) by the decoupling technique is close to m^2.

Similarly, the decrease in the number of required additions for the decoupled EKF Equations (4.27), (4.28), and (4.29) by approximately a factor of m^2 can be explained by considering the required number of additions in the matrix–matrix multiplications which are predominant in Equations (4.27), (4.28), and (4.29). The matrix–matrix multiplication of two n–by–n matrices requires n^3-n^2 additions; n^3-n^2 is an order of n^3 operations. It was shown above that an order of n^3 multiplications required for the matrix–matrix multiplication is reduced by a factor of approximately m^2 by the decoupling technique. Hence, the decoupling technique reduces the required order of n^3 additions for the matrix–matrix multiplication by approximately a factor of m^2.

Since the coupled and decoupled EKF's use the same equations (2.21)

and (4.31) to estimate state, and the same equations (2.24) and (4.32) to predict state, there is no difference in computational requirements for a state estimation and prediction. However, the decoupled ECKF requires the additional transformation of the Kalman gain which is performed in Equation (4.30). It requires m–1 CORDIC operations plus either nm+2m–2 or nm+2m multiplications; nm+2m–2 is for 3–dimensional tracking and nm+2m is for 2–dimensional tracking. The additional calculations for Equation (4.30) represent a small price to pay for the reduction in calculating the decoupled ECKF equations (4.27), (4.28), and (4.29).

Table 4.1 (b) and 4.2 (b) indicate that for 3–dimensional tracking the total reduction rate in the number of multiplication by the decoupling technique is approximately 20, as shown by:

$$\frac{8538}{427} = 19.995 \approx 20,$$

and the total reduction rate in the number of additions is approximately 10:

$$\frac{2422}{243} = 9.967 \approx 10.$$

However, for 3–dimensional tracking the decoupled ECKF requires 5 scalar divisions and two more CORDIC operations than the coupled ECKF. These additional computations by the decoupled ECKF are negligible compared to the computational reduction by the decoupling technique.

Table 4.1 Computational requirements for the coupled ECKF
(a) for m–dimensional tracking with n state elements
(b) for 3–dimensional tracking with 9 state elements

Table 4.1 (a)

Equation number	Number of operations per iteration Mult.	Add.	CORDIC.
2.20a	$2n^2m+2nm^2$	$2n^2m-3nm+2nm^2$	0
2.20b	$8n^3$	$2n^3-n^2$	0
2.21	nm	nm	m–1
2.22	$2n^2m$	$2n^2m-nm$	0
2.23	$2n^3$	$2n^3-n^2$	0
2.24	n^2	n^2-n	0
Total*	$10n^3+n^2+4n^2m$ $+2nm^2+nm$	$4n^3-n^2+n+4n^2m$ $+2nm^2-3nm$	m–1

Mult. = Multiplication
Add. = Addition
CORDIC = CORDIC operations

2.20a = 2.20 without a matrix inversion
2.20b = only a matrix inversion in 2.20

* The total does not include the number of operations for Equation (2.19).

Table 4.1 (b)

Equation number	Number of operation per iteration				
	Mult.	Div.	Sq-Root	Add.	CORDIC.
2.19	6	8	1	1	0
2.20a	648	0	0	567	0
2.20b	5832	0	0	648	0
2.21	27	0	0	27	2
2.22	486	0	0	459	0
2.23	1458	0	0	648	0
2.24	81	0	0	72	0
Total	8538	8	1	2422	2

Div. = Division

Sq-Root = Squar root operation

Table 4.2 Computational requirements for the decoupled ECKF
(a) for m–dimensional tracking with n state elements
(b) for 3–dimensional tracking with 9 state elements

Table 4.2 (a)

Equation number	Number of operations per iteration Mult.	Div.	Add.	CORDIC.
4.27	$2n^2/m+2n$	$2m-1$	$2n^3/m-n$	0
4.28	$2n^2/m$	0	$2n^2/m-n$	0
4.29	$2n^3/m^2$	0	$n^3/m^2-n^2/m$	0
4.30*	$nm+(m-1)2$	0	0	$m-1$
4.31	nm	0	nm	$m-1$
4.32	n^2	0	n^2-n	0
Total*	$n^2+2n+2n^3/m^3$ $+4n^2/m+2nm+2m-2$	$2m-1$	$n^2-3n+nm$ $+n^3/m^2+3m^2/m$	$2(m-1)$

Mult. = Mutiplication
Div. = Division
Add. = Addition
CORDIC = CORDIC operation

* For 2–dimensional tracking with 6 state elements, the number of required additions for Equation (4.30) is nm and the total number of required additions for the decoupled ECKF is $n^2+2n+2n^3/m^3+4n^2/m+2nm$.

Table 4.2 (b)

Equation number	Number of operation per iteration			
	Mult.	Div.	Add.	CORDIC.
4.27	72	5	45	0
4.28	54	0	45	0
4.29	162	0	54	0
4.30	31	0	0	2
4.31	27	0	27	2
4.32	81	0	72	0
Total	427	5	243	4

4.3.2 Simplication using the special properties of the tracking KF

The decoupled ECKF equation (4.27) for the x–axis can be simplified by exploiting the sparse nature of $H_{o-x}(k)$, as follows:

$$K_o(k) = \bar{P}_o(k)\, H_o^T(k)\, (\, H_o(k)\, P_o(k|k-1)\, H_o^T(k) \;+\; R(k)\,)^{-1} \qquad (4.27)$$

$$K_{o-x}(k) =$$

$$\begin{bmatrix} \bar{p}_{xx} & \bar{p}_{x\dot{x}} & \bar{p}_{x\ddot{x}} \\ \bar{p}_{\dot{x}x} & \bar{p}_{\dot{x}\dot{x}} & \bar{p}_{\dot{x}\ddot{x}} \\ \bar{p}_{\ddot{x}x} & \bar{p}_{\ddot{x}\dot{x}} & \bar{p}_{\ddot{x}\ddot{x}} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} ([1\ 0\ 0\ ] \begin{bmatrix} \bar{p}_{xx} & \bar{p}_{x\dot{x}} & \bar{p}_{x\ddot{x}} \\ \bar{p}_{\dot{x}x} & \bar{p}_{\dot{x}\dot{x}} & \bar{p}_{\dot{x}\ddot{x}} \\ \bar{p}_{\ddot{x}x} & \bar{p}_{\ddot{x}\dot{x}} & \bar{p}_{\ddot{x}\ddot{x}} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + \sigma_r^2)^{-1}$$

$$= \begin{bmatrix} \bar{p}_{xx} \\ \bar{p}_{\dot{x}x} \\ \bar{p}_{\ddot{x}x} \end{bmatrix} (p_{xx} + \sigma_r^2)^{-1} \qquad (4.27\text{–x})$$

The superscripts, "–", assigned to the elements of P(k|k–1) are used to denote prediction hereafter. Similar to the KF equation (4.27) for the x–axis, the KF equations (4.27) for the y and z–axes can be simplified. The simplified equations for the y– and z–axes are as follows:

$$K_{o-y}(k) = \begin{bmatrix} \bar{p}_{yy} \\ \bar{p}_{\dot{y}y} \\ \bar{p}_{\ddot{y}y} \end{bmatrix} (\, (p_{yy}/r^2 + \sigma_{\theta_A}^2)\, r\,)^{-1} \qquad (4.27\text{–y})$$

$$K_{o-z}(k) = \begin{bmatrix} \bar{p}_{zz} \\ \bar{p}_{\dot{z}z} \\ \bar{p}_{\dot{z}\cdot z} \end{bmatrix} \left((\bar{p}_{zz}/r^2 + \sigma^2_{\theta_E}) \, r \right)^{-1} \qquad (4.27\text{-}z)$$

This simplification eliminates matrix–vector multiplications, and the simplified equations require only scaling of the first column vector of $P_o(k|k-1)$.

The decoupled ECKF equation (4.28) for the x–axis can be simplified as follows, using the sparse nature of $H_o(k)$:

$$\begin{aligned} P_o(k) &= (I - K_o(k) \, H_o(k)) \, P_o(k|k-1) \\ &= P_o(K|K-1) - K_o(K) \, H_o(k) \, P_o(k|k-1) \end{aligned} \qquad (4.28)$$

$$P_{o-x}(k) = \begin{bmatrix} \bar{p}_{xx} & \bar{p}_{x\dot{x}} & \bar{p}_{x\dot{x}\cdot} \\ \bar{p}_{\dot{x}x} & \bar{p}_{\dot{x}\dot{x}} & \bar{p}_{\dot{x}\dot{x}\cdot} \\ \bar{p}_{\dot{x}\cdot x} & \bar{p}_{\dot{x}\cdot\dot{x}} & \bar{p}_{\dot{x}\cdot\dot{x}\cdot} \end{bmatrix} - \begin{bmatrix} k_{xr} \\ k_{\dot{x}r} \\ k_{\dot{x}\cdot r} \end{bmatrix} [1 \; 0 \; 0] \begin{bmatrix} \bar{p}_{xx} & \bar{p}_{x\dot{x}} & \bar{p}_{x\dot{x}\cdot} \\ \bar{p}_{\dot{x}x} & \bar{p}_{\dot{x}\dot{x}} & \bar{p}_{\dot{x}\dot{x}\cdot} \\ \bar{p}_{\dot{x}\cdot x} & \bar{p}_{\dot{x}\cdot\dot{x}} & \bar{p}_{\dot{x}\cdot\dot{x}\cdot} \end{bmatrix}$$

$$= \begin{bmatrix} \bar{p}_{xx} & \bar{p}_{x\dot{x}} & \bar{p}_{x\dot{x}\cdot} \\ \bar{p}_{\dot{x}x} & \bar{p}_{\dot{x}\dot{x}} & \bar{p}_{\dot{x}\dot{x}\cdot} \\ \bar{p}_{\dot{x}\cdot x} & \bar{p}_{\dot{x}\cdot\dot{x}} & \bar{p}_{\dot{x}\cdot\dot{x}\cdot} \end{bmatrix} - \begin{bmatrix} k_{xr} \\ k_{\dot{x}r} \\ k_{\dot{x}\cdot r} \end{bmatrix} [\bar{p}_{xx} \; \bar{p}_{x\dot{x}} \; \bar{p}_{x\dot{x}\cdot}] \qquad (4.28\text{-}x)$$

Similarly, the KF equation (4.28) for the y– and z–axes can be simplified. The simplified equations for the y– and z–axes are as follows:

$$P_{o-y}(k) = \begin{bmatrix} \bar{p}_{yy} & \bar{p}_{y\dot{y}} & \bar{p}_{y\dot{y}\cdot} \\ \bar{p}_{\dot{y}y} & \bar{p}_{\dot{y}\dot{y}} & \bar{p}_{\dot{y}\dot{y}\cdot} \\ \bar{p}_{\dot{y}\cdot y} & \bar{p}_{\dot{y}\cdot\dot{y}} & \bar{p}_{\dot{y}\cdot\dot{y}\cdot} \end{bmatrix} - \begin{bmatrix} k_{y\theta} \\ k_{\dot{y}\theta} \\ k_{\dot{y}\cdot\theta} \end{bmatrix} [\, \bar{p}_{yy} \; \bar{p}_{y\dot{y}} \; \bar{p}_{y\dot{y}\cdot}] \qquad (4.28\text{–}y)$$

$$P_{o-z}(k) = \begin{bmatrix} \bar{p}_{zz} & \bar{p}_{z\dot{z}} & \bar{p}_{z\dot{z}\cdot} \\ \bar{p}_{\dot{z}z} & \bar{p}_{\dot{z}\dot{z}} & \bar{p}_{\dot{z}\dot{z}\cdot} \\ \bar{p}_{\dot{z}\cdot z} & \bar{p}_{\dot{z}\cdot\dot{z}} & \bar{p}_{\dot{z}\cdot\dot{z}\cdot} \end{bmatrix} - \begin{bmatrix} k_{z\theta} \\ k_{\dot{z}\theta} \\ k_{\dot{z}\cdot\theta} \end{bmatrix} [\, \bar{p}_{zz} \; \bar{p}_{z\dot{z}} \; \bar{p}_{z\dot{z}\cdot}] \qquad (4.28\text{–}z)$$

This simplication again eliminates a matrix–vector multiplication; and the simplified equations also require only a row–column vector multiplication and matrix–matrix subtraction. The subscript θ in Equation (4.28–y) denotes an azimuth angle, while the subscript $\dot{\theta}$ in Equation (4.28–z) denotes an elevation angle.

Table 4.3 (a) lists the reduced computational requirements of the simplified extended covariance Kalman filter (ECKF) through the use of the decoupling technique and special properties for a m–dimensional tracking with n state elements, whereas Table 4.3 (b) lists the computational requirements of the simplified ECKF's for a typical 3–dimensional tracking system with 9 state elements.

A comparison of Tables 4.1 (b) and 4.3 (b) reveals that for 3–dimensional tracking the rate of reduction in the number of multiplications from the coupled to the simplified ECKF is approximately 24:

$$\frac{8538}{350} = 24.39 \approx 24,$$

and the number of additions is reduced by approximately 13:

$$\frac{2422}{183} = 13.23 \approx 13,$$

Note that like the decoupled ECKF, the simplified ECKF requires 2 more CORDIC operations than the coupled ECKF. These additional operations are negligible, when compared to the computational reduction afforded by the use of the decoupling technique and properties of the tracking KF.

Table 4.3 Computational requirements for the simpified ECKF
(a) for m–dimensional tracking with n state elements
(b) for 3–dimensional tracking with 9 state elements

Table 4.3 (a)

Equation number	Number of operations per iteration Mult.	Div.	Add.	CORDIC.
4.27	$n+2m-2$	$2m-1$	m	0
4.28	$n^2/m+m$	0	n^2/m	0
4.29	$2n^3/m^2$	0	$n^3/m^2-n^2/m$	0
4.30*	$nm+2m-2$	0	0	$m-1$
4.31	nm	0	nm	$m-1$
4.32	n^2	0	n^2-n	0
Total*	$2n^3/m^3+n^2/m+n^2$ $2nm+2n+4m-4$	$2m-1$	n^3/m^2+nm+n^2-n+m	$2m-2$

Mult. = Multiplication
Div. = Division
Add. = Addition
CORDIC = CORDIC operations

* For 2–dimensional tracking with 6 state elements, the number of required additions for Equation (4.30) is $nm+2m$ and the total number of required additons is $2n^3/m^3+n^2/m+n^2+2nm+2n+4m-2$.

Table 4.3 (b)

Equation number	Number of operation per iteration			
	Mult.	Div.	Add.	CORDIC.
4.27	13	5	3	0
4.28	36	0	27	0
4.29	162	0	54	0
4.30	31	0	0	2
4.31	27	0	27	2
4.32	81	0	72	0
Total	350	5	183	4

4.4 Simplification of the Extended SRCF

4.4.1 Simplification using the decoupling technique

Using the relationship between the Kalman gain $K_o(k)$ for the LOS frame and the Kalman gain K(k) for the reference Cartesian frame, developed in the previous section 4.1, the extended SRCF summarized in Equations (2.25 .. 2.30) may be decoupled.

The Kalman gain is developed in the SRCF in terms of the matrices developed during the measurement update of the square root S(k) of the state estimate error covariance matrix P(k). Hence, the extended SRCF may be decoupled as follows:

a) Processing in the LOS frame:

Time update of $S_o(k)$

Measurement update of $S_o(k)$

Calculation of the Kalman gain $K_o(k)$

b) Transformation of the Kalman gain:

$$K(k) = \begin{bmatrix} F_1(k) & 0 & 0 \\ 0 & F_1(k) & 0 \\ 0 & 0 & F_1(k) \end{bmatrix} K_o(k)$$

c) Processing in the reference Cartesian frame:

Estimate a state vector X(k)

Predict a state vector X(k+1 | k)

This decoupling is similar to that for the extended covariance KF, except that $S_o(k)$ is updated in the decoupled SRCF, whereas $P_o(k)$ is updated in the decoupled extended covariance KF. The decoupled extended SRCF is summarized in detail below:

a) Processing in the LOS frame computes $K_o(k)$, $S_o(k)$ and $S_o(k+1|k)$, as shown by

$$\begin{bmatrix} S_o^T(k|k-1) \\ 0 \end{bmatrix} = Q_1 \begin{bmatrix} S_o^T(k-1) & \phi^T(k-1) \\ U_o^T(k) & \end{bmatrix} \tag{4.33}$$

$$\begin{bmatrix} F_o(k) & G_o(k) \\ 0 & S_o^T(k) \end{bmatrix} = Q_1 \begin{bmatrix} V^T(k) & 0 \\ S_o^T(k|k-1)H_o^T(k) & S_o^T(k|k-1) \end{bmatrix} \tag{4.34}$$

$$K_o(k) = G_o^T(k) \;/\; F_o^T(k) \tag{4.35}$$

The matrix Q_1 denotes an orthogonal matrix that upper–triangularizes a matrix on its righthand side.

b) Coordinate transformation computes $K(k)$ from $K_o(k)$ by using the Jacobian transformation:

$$K(k) = \begin{bmatrix} F_1 & 0 & 0 \\ 0 & F_1 & 0 \\ 0 & 0 & F_1 \end{bmatrix} K_o(k) \tag{4.36}$$

where F_1 is defined in Equations (4.19) and (4.20) for 2– and 3–dimensional tracking systems, respectively.

c) Processing in the reference Cartesian coordinates frame computes $\hat{X}(k)$ and $\hat{X}(k+1|k)$, as shown by

$$\hat{X}(k) = \hat{X}(k|k-1) + K(k) \left(Z(k) - h(\hat{X}(k|k-1)) \right) \tag{4.37}$$

$$\hat{X}(k+1|k) = \phi(k) \hat{X}(k) \tag{4.38}$$

As in the ECKF, the decoupling technique, applied to ESRCF, allows the ESRCF to propagate m $(\frac{n}{m})$–dimensional $S_o(k)$'s separately for each axis, instead of one n–dimensional $S_o(k)$, where m is 2 for 2–dimensional tracking, or 3 for 3–dimensional tracking. This decoupling technique not only reduces computational requirements but also increases parallelism, because the propagation of m $S_o(k)$'s can be performed for each axis in parallel.

Tables 4.4 (a) and 4.4 (b) present the computational requirements of the coupled ESRCF; the former is based on a general m–dimensional tracking system with n state elements, and the latter is based on a typical 3–dimensional tracking system with 9 state elements. Simlilarly, Tables 4.5 (a) and 4.5 (b) present the computational requirements of the decoupled ESRCF for the two above cases. In these tables, the computational requirements for time and measurement updates are separated into two parts to illustrate the effects of the decoupling technique more clearly. For example, Equation (2.29)

$$\begin{bmatrix} S^T(k|k-1) \\ 0 \end{bmatrix} = Q_1 \begin{bmatrix} S^T(k-1) & \phi^T(k-1) \\ U^T(k) & \end{bmatrix} \tag{2.29}$$

is split into a matrix–matrix multiplication $[S^T(k-1)\ \phi^T(k-1)]$ (2.29a) and an orthogonal upper triangularization (2.29b). The QR decomposition is assumed to be employed for an orthogonal upper triangularization. Furthermore, the ESRCF equations (2.27) and (2.28) are assumed to be calculated in the following sequence to avoid a matrix inversion of $F^T(k)$ using the method of backward substitution:

a) Perform CORDIC operation on $\hat{X}(k|k-1)$:

$$Z(k|k-1) = h(\hat{X}(k|k-1))$$

b) Calculate $a(k) = Z(k) - Z(k|k-1)$

c) Calculate $b(k) = (F^T(k))^{-1}a(k)$ by back substitution

d) Calculate $\hat{x}(k) = \hat{x}(k|k-1) + G^T(k)b(k)$

Note that the number of required operations for Equation (2.25) is not included in Table 4.4 (a), because it is impossible to express it in terms of m and n.

We now examine how the use of the decoupling technique reduces computational requirements. The time and measurement updates of S(k) in the coupled ESRCF require a combination of an order of m^3 $\{0(n^3)\}$ and an order of n^2 $\{0(n^2)\}$ operations. For example, as summarized in Table 4.4 (a), the orthogonal upper triangularization for time update in Equation (2.29b) requires $4n^3+n^2$ multiplications, $2n^2$ divisions, n^2 square root operations, and $2n^3+n^2$ additions.

In Section 4.3, we have found that the decoupling technique decreases the computational requirements of an order of n^3 $\{0(n^3)\}$ operations by a factor of m^2. Similarly, the decoupling technique reduces an order of n^2 $\{0(n^2)\}$ operations by a factor of m. When the coupled system requires n^2 operations, the decoupled

system requires $(n/m)^2$ operations m times. Hence, the decoupled system requires total n^2/m operations which is m times less than n^2.

A comparison of Tables 4.4 (a) and 4.5 (a) confirms that the decoupling technique reduces an order of n^3 operations by a factor of m^2 and an order of n^2 operations by a factor of m. For example, the decoupled ESRCF equation (4.33b) which corresponds to the coupled ESRCF equation (2.29b) requires $4n^3/m^2+n^2/m$ multiplications, $2n^2/m$ divisions, n^2/m square root operations, and $2n^3/m^2+n^2/m$ additions.

Table 4.6 expresses the computational reduction for 3–dimensional tracking by the decoupling technique as the ratio of the required number of operations for the coupled ESRCF to that for the decoupled ESRCF. For multiplications and additions, the reduction ratios are approximately 7.0 and 6.4 respectively. These reduction factors are between 9 ($=m^2$) and 3 (=m), as expected.

The overall reduction ratios of 7.0 and 6.4 for the ESRCF are less than the ratios of 20 and 10 for the ECKF, respectively. This is mainly because in the ECKF the computationally intensive matrix inversion is eliminated by the decoupling technique, whereas in the ESRCF the matrix inversion is already avoided by the use of forward substitution in the coupled ESRCF.

4.4.2 Simplification using the special properties

The sparse nature of the measurement matrix $H_o(k)$ in the LOS frame could be utilized in the multiplication of $S_o^T(k|k-1)$ and $H_o^T(k)$. For example, the sparse measurement matrix $H_{o-x}(k)$ for the x–axis turns the multiplication of $S_{o-x}^T(k|k-1)$ and $H_{o-x}^T(k)$ into the extraction of the first column from $S_{o-x}^T(k|k-1)$. That is, the multiplication of $S_{o-x}^T(k|k-1)$ and $H_{o-x}^T(k)$ does not

require any multiplication or addition, as shown below:

$$S^{T}_{o-x}(k|k-1)\ H^{T}_{o-x}(k) = \begin{bmatrix} S^{-}_{xx}(k) & S^{-}_{x\dot{x}}(k) & S^{-}_{x\dot{x}\cdot}(k) \\ 0 & S^{-}_{\dot{x}\dot{x}}(k) & S^{-}_{\dot{x}\dot{x}\cdot}(k) \\ 0 & 0 & S^{-}_{\dot{x}\cdot\dot{x}\cdot}(k) \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} S^{-}_{xx}(k) \\ 0 \\ 0 \end{bmatrix} \qquad (4.39\text{–}x)$$

Note that there is only one nonzero element in the product of $S^{T}_{o-x}(k|k-1)$ and $H^{T}_{o-x}(k)$

Simlilarly, the multiplication of $S^{T}_{o-y}(k|k-1)$ and $H^{T}_{o-y}(k)$, and that of $S^{T}_{o-z}(k|k-1)$ and $H^{T}_{o-z}(k)$ are simplified, as shown below:

$$S^{T}_{o-y}(k|k-1)\ H^{T}_{o-y}(k) = \begin{bmatrix} S^{-}_{yy}(k) & S^{-}_{y\dot{y}}(k) & S^{-}_{y\dot{y}\cdot}(k) \\ 0 & S^{-}_{\dot{y}\dot{y}}(k) & S^{-}_{\dot{y}\dot{y}\cdot}(k) \\ 0 & 0 & S^{-}_{\dot{y}\cdot\dot{y}\cdot}(k) \end{bmatrix} \begin{bmatrix} 1/r \\ 0 \\ 0 \end{bmatrix}$$

$$= \frac{1}{r} \begin{bmatrix} S^{-}_{yy}(k) \\ 0 \\ 0 \end{bmatrix} \qquad (4.39\text{–}y)$$

$$S^T_{o-z}(k|k-1)\, H^T_{o-z}(k) = \begin{bmatrix} S^-_{zz}(k) & S^-_{z\dot{z}}(k) & S^-_{z\dot{z}\cdot}(k) \\ 0 & S^-_{\dot{z}\dot{z}}(k) & S^-_{\dot{z}\dot{z}\cdot}(k) \\ 0 & 0 & S^-_{\dot{z}\cdot\dot{z}\cdot}(k) \end{bmatrix} \begin{bmatrix} 1/r \\ 0 \\ 0 \end{bmatrix}$$

$$= \frac{1}{r} \begin{bmatrix} S^-_{zz}(k) \\ 0 \\ 0 \end{bmatrix} \qquad (4.39\text{–z})$$

These multiplications are reduced to the extraction of the first column of $S^T_{o-y}(k|k-1)$ and $S^T_{o-z}(k|k-1)$, followed by a scalar division. Note that as in Equation (4.39–x), the product of $S_{o-y}(k|k-1)$ and $H^T_{o-y}(k)$, and that of $S_{o-z}(k|k-1)$ and $H^T_{o-y}(k)$ have only one nonzero element.

The fact that the product of $S^T_o(k|k-1)$ and $H^T_o(k)$ has only one non–zero element may be used to simplify the orthogonal upper triangularization needed for the measurement update in Equation (4.34):

$$\begin{bmatrix} F_o(k) & G_o(k) \\ 0 & S^T_o(k) \end{bmatrix} = Q_1 \begin{bmatrix} V^T_o(k) & 0 \\ S^T_o(k|k-1)H^T_o(k) & S^T_o(k|k-1) \end{bmatrix} \qquad (4.34)$$

The orthogonal triangularization for the measurement update in the x–axis is expressed in detail below:

$$\begin{bmatrix} f_{11} & g_{11} & g_{12} & g_{13} \\ 0 & s_{11} & s_{21} & s_{31} \\ 0 & 0 & s_{22} & s_{32} \\ 0 & 0 & 0 & s_{33} \end{bmatrix} = Q_1 \begin{bmatrix} \sigma_f & 0 & 0 & 0 \\ \cancel{s^-_{11}} & s^-_{11} & s^-_{21} & s^-_{31} \\ 0 & 0 & s^-_{22} & s^-_{32} \\ 0 & 0 & 0 & s^-_{33} \end{bmatrix} \qquad (4.34\text{–x})$$

This equation shows that only one element $\bar{s}_{11}$, marked by a cross X, is to be nullified in this orthogonalization.

Similarly, only one element is to be nullified in the orthogonal triangularizations for measurement updates in the y– and z–axes, as shown below:

For y–axis, we have

$$\begin{bmatrix} f_{11} & g_{11} & g_{12} & g_{13} \\ 0 & s_{11} & s_{21} & s_{31} \\ 0 & 0 & s_{22} & s_{32} \\ 0 & 0 & 0 & s_{33} \end{bmatrix} = Q_1 \begin{bmatrix} r\sigma_\theta & 0 & 0 & 0 \\ \bar{s}_{11}/r & \bar{s}_{11} & \bar{s}_{21} & \bar{s}_{31} \\ 0 & 0 & \bar{s}_{22} & \bar{s}_{32} \\ 0 & 0 & 0 & \bar{s}_{33} \end{bmatrix} \qquad (4.34\text{–y})$$

For z–axis, we have

$$\begin{bmatrix} f_{11} & g_{11} & g_{12} & g_{13} \\ 0 & s_{11} & s_{21} & s_{31} \\ 0 & 0 & s_{22} & s_{32} \\ 0 & 0 & 0 & s_{33} \end{bmatrix} = Q_1 \begin{bmatrix} r\sigma_\theta & 0 & 0 & 0 \\ \bar{s}_{11}/r & \bar{s}_{11} & \bar{s}_{21} & \bar{s}_{31} \\ 0 & 0 & \bar{s}_{22} & \bar{s}_{32} \\ 0 & 0 & 0 & \bar{s}_{33} \end{bmatrix} \qquad (4.34\text{–z})$$

The angle θ in Equation (4.34–y) denotes θ_A, whereas θ in Equation (4.34–z) deonotes θ_E.

Table 4.7 (a) and 4.7 (b) present the computational requirements of the simplified ESRCF through the use of the decoupling technique and special properties in the KF; the former is based on a general m–dimensional tracking system with n state elements, and the latter is based on a typical 3–dimensional tracking system with 9 state elements. Table 4.8 summarizes the reduction rates

for the computational requirements of the ESRCF by using the decoupling technique and properties of the tracking systems. The reduction rate for multiplication is 8.4, and that for addition is 7.5.

Table 4.4 Computational requirements for the coupled ESRCF
(a) for m–dimensional tracking with n state elements
(b) for 3–dimensional tracking with 9 state elements

Table 4.4 (a)

Equation number	Number of operation per iteration				
	Mult.	Div.	Sq–Root.	Add.	CORDIC.
2.26a	$m[n(n+1)/2]$	0	0	$m(n-1)/2$	0
2.26b	$[2+4(n+m)]nm$	$2nm$	nm	$[1+2(n+m)]nm$	0
2.27 & 2.28	$nm+m^2/2-m/2$	m	0	$nm-n+m^2+m/2$	$m-1$
2.29a	$n^2(n+1)/2$	0	0	$n^2(n-1)/2$	0
2.29b	$4n^3+n^2$	$2n^2$	n^2	$2n^3+n^2$	0
2.30	n^2	0	0	$n(n-1)$	0
Total*	$\frac{9}{2}n^3+\frac{5}{2}n^2+\frac{9}{2}n^2m$ $+\frac{7}{2}nm+4nm^2+\frac{m^2}{2}-\frac{m}{2}$	$2n^2+2nm+m$	n^2+nm	$3n^3+\frac{3}{2}n^2-2n+2n^2m$ $+\frac{5}{2}nm+2nm^2+\frac{m^2}{2}$	$m-1$

Mult. = Multiplication
Div. = Division
Sq–Root. = Square root operation
Add. = Addition
CORDIC.= CORDIC operations

* The total does not include the number of operations for Equation (2.25).

Table 4.4 (b)

Equation number	Number of operation per iteration				
	Mult.	Div.	Sq-Root.	Add.	CORDIC.
2.25	6	8	1	1	0
2.26a	135	0	0	12	0
2.26b	1350	54	27	675	0
2.27 & 2.28	30	3	0	24	2
2.29a	405	0	0	324	0
2.29b	2997	162	81	1539	0
2.30	81	0	0	72	0
Total	5004	227	109	2647	2

Table 4.5 Computational requirements for the decoupled ESRCF
(a) for m-dimensional tracking with n state elements
(b) for 3-dimensional tracking with 9 state elements

Table 4.5 (a)

Equation number	Number of operation per iteration				
	Mult.	Div.	Sq-Root.	Add.	CORDIC.
4.33a	$(n^2/2m)(n/m+1)$	0	0	$(n^2/2m)(n/m-1)$	0
4.33b	$4n^3/m^2+n^2/m$	$2n^2/m$	n^2/m	$2n^3/m^2+n^2/m$	0
4.34a	$(n/2)(n/m+1)$	$m-1$	0	$(n/2)(n/m-1)$	0
4.34b	$4n^2/m+5n$	$2n$	n	$2n^2/m+3n$	0
4.35	0	n	0	0	0
4.36*	$nm+(m-1)2$	0	0	0	$m-1$
4.37	nm	0	0	$nm+n$	$m-1$
4.38	n^2	0	0	n^2-n	0
Total*	$\frac{9n^3}{2m^2}+\frac{6n^2}{m}+\frac{11n}{2}$ $+n^2+2nm+2m-2$	$\frac{2n^2}{m}+m$ $+3n-1$	$\frac{n^2}{m}+n$	$\frac{5n^3}{2m^2}+\frac{3n^2}{m}+\frac{5n}{2}$ $+n^2+nm$	$2m-2$

* For 2-dimensional tracking with 6 state elements, the number of required multiplications for Equation (4.36) is nm and the total number of required multiplications for the decoupled ESRCF is $\frac{9n^3}{2m^2}+\frac{6n^2}{m}+\frac{11n}{2}+n^2+2nm$.

Table 4.5 (b)

Equation number	Number of operation per iteration				
	Mult.	Div.	Sq-Root.	Add.	CORDIC.
4.33a	54	0	0	27	0
4.33b	351	54	27	189	0
4.34a	18	2	0	9	0
4.34b	153	18	9	81	0
4.35	0	9	0	0	0
4.36	31	0	0	0	2
4.37	27	0	0	36	2
4.38	81	0	0	72	0
Total	715	83	36	414	4

Table 4.6 Computational reduction ratios for the ESRCF by the use of the decoupling technique

Mult.	Div.	Sq-Root.	Add.	CORDIC.
7.00	2.73	3.03	6.39	0.5

Table 4.7 Computational requirements for the simplified ESRCF
(a) for m–dimensional tracking with n state elements
(b) for 3–dimensional tracking with 9 state elements

Table 4.7 (a)

Equation number	Number of operation per iteration				
	Mult.	Div.	Sq–Root.	Add.	CORDIC.
4.33a	$(n^2/2m)(n/m+1)$	0	0	$(n^2/2m)(n/m-1)$	0
4.33b	$4n^3/m^2+n^2/m$	$2n^2/m$	n^2/m	$2n^3/m^2+n^2/m$	0
4.34a	m–1	m–1	0	0	0
4.34b	5m+4n	2m	m	3m+2n	0
4.35	0	n	0	0	0
4.36*	nm+(m–1)2	0	0	0	m–1
4.37	nm	0	0	nm+n	m–1
4.38	n^2	0	0	n^2-n	0
Total*	$\frac{9n^3}{2m^2}+\frac{3n^2}{2m}+4n$ $+n^2+2nm+8m-3$	$\frac{2n^2}{m}+n$ $+3m-1$	$\frac{n^2}{m}+m$	$\frac{5n^3}{2m^2}+\frac{n^2}{2m}+2n$ $+n^2+nm+3m$	2m–2

* For 2–dimensional tracking with 6 state elements, the number of required multiplications for Equation (4.36) is nm and the total number of required multiplications for the decoupled ESRCF is $\frac{9n^3}{2m^2}+\frac{6n^2}{m}+4n+n^2+2nm+6m-1$.

Table 4.7 (b)

Equation number	Number of operation per iteration				
	Mult.	Div.	Sq-Root.	Add.	CORDIC.
4.33a	54	0	0	27	0
4.33b	351	54	27	189	0
4.34a	2	2	0	0	0
4.34b	51	6	3	27	0
4.35	0	9	0	0	0
4.36	31	0	0	0	2
4.37	27	0	0	36	2
4.38	81	0	0	72	0
Total	597	71	30	351	4

Table 4.8 Computational reduction ratios for the ESRCF by the use of the decoupling technique and properties of the tracking KF

Mult.	Div.	Sq-Root.	Add.	CORDIC.
8.38	3.20	3.63	7.54	0.5

4.5 Simulation of the Simplified Kalman Filter

In this section, we compare the performance of the simplified extended Kalman filter to that of the conventional coupled extended Kalman filter. We showed in the previous sections that the use of the decoupling technique and properties of the tracking Kalman filter reduces computational requirments and increases parallelism significantly. However, these benefits would not be justified if they were achieved at the cost of a significant degradation.

The tracking performance depends on target dynamics and the accuracy of measurements. A target that does not manoeuvre severely is easy to track and accurate measurements help tracking. The tracking performance also depends on the filter used, because the accuracy depends on the nature of the filter algorithm and its numerical properties. The accuracy of the model of target dynamics and measurement systems used in a filter also affects tracking performance. Inaccurate models degrade performance, and sometimes lead to divergence (12], [38]).

A number of authors ([2],[8],[35]) have studied the performance of the decoupled extended covariance Kalman filter through simulation. It was found in [2] and [8] that the performance of the decoupled ECKF is generally comparable to that of the coupled ECKF in typical tracking situations, while it was found in [8] that the decoupling technique reduces the effects of truncation and round-off errors due to finite word-length arithmetic. However, the performance of the decoupled ECKF was found to degrade somewhat, when the condition that the orientation of a LOS frame does not vary significantly between filtering updates, needed for the decoupling technique, does not hold well[35]. The rate of change in the orientation of the LOS frame is determined by the characteristics of the tracking environment such as the filtering interval and the direction of target movement.

In this section we investigate the effects of simplification, including the decoupling technique, on the performance of the Precise Radar Aided Navigation (PRAN) system. The PRAN system is being developed for ship's navigation in confined waterways at the Communications Research Laboratory, McMaster University, in conjunction with the Transport Development Center of Montreal, Canada. The PRAN system utilizes passive polarimetric reflectors, strategically placed along a confined waterway, as reference points to achieve accurate positioning.

The passive polarimetric reflector, patented by A. Macikunas, S. Haykin, and T Greenlay [33] changes the polarization of the incident electromagnetic signal through 90°, whereas most natural and man-made objects reflect the incident signal in the same polarization. As a result, in the PRAN system which transmits a horizontally polarized signal and receives on a vertically polarized antenna, the passive polarimetric reflectors are easily located in the presence of clutter which does not rotate the polarization of the incident signal.

In the course of this project, a software package simulating the PRAN system has been developed. This package receives as input the PRAN system configuration parameters such as the location of reflectors, a ship's path and information on the ship's dynamics. It then generates measurements as it simulates the ship's movement along the path. This package is designed to be flexible to handle various configurations of reflectors and a multitude of ship's dynamics. For more details, see Appendix B.

Measurements generated by the PRAN system simulation package are processed by the coupled and decoupled ECKF's. The results of filtering with these two filters are compared to study the effects of decoupling. Note that the coupled and decoupled ECKF's are employed to perform filtering, instead of the

coupled and decoupled ESRCF's. It is because the effect of decoupling on the reduction of numerical ill–conditioning was predicted to be more noticeable in using the ECKF than using the ESRCF.

4.5.1 Simulation Configurations

Two typical scenarios one would encounter in the PRAN system are: a) a ship travelling in a straight line at a virtually constant speed, b) a ship turning at a virtually constant speed. The term, "virtually constant speed", refers to a speed which fluctuates around a particular predefined constant speed. This fluctuation is represented as a random acceleration in the PRAN system simulation program.

A ship experiences pitching, rolling, and yawing motions. However, yawing motion is usually negligible, so that only pitching and rolling motions are included in the simulation. Pitching and rolling motions do not affect the position of a ship, but affect the orientation of an antenna, which introduces errors in measurements. Pitching and rolling motions are simultated using a sinusoidal function in the PRAN system simulation program, for pitching and rolling motions can be approximated as sinusoidal motions [51].

Tables 4.9 and 4.10 show the ship's dynamics and the characteristics of the measurements used in the simulation respectively. These characteristics are obtained from the system specification set out for the PRAN system [7].

Two typical PRAN system scenerios used in simulation are shown in Figures 4.4 and 4.5. Figure 4.4 shows a ship which is set to move in a straight line at 45° from the x–axis in the reference Cartesian coordinates. In simulation, the ship deviates a little bit from the path it is set to travel along, due to the variations in the ships dynamics, random acceleration. The reason for choosing 45° is that the coupling between x– and y–axes in the measurement error

covariance matrix R(k) and the state estimate error matrix P(k), expressed in the reference Cartesian coordinate system, is greatest at this point. Hence, this scenario is ideal to show how well the decoupling technique works. Furthermore, the orientation of an LOS frame in this scenario does not vary greatly as the ship moves, because the LOS remains close to 45°, as the ship travels. Hence, the condition that the orientation of the LOS frame does not change significantly, needed for the decoupling technique to be effective, is met.

Figure 4.5 shows a ship turning around a reflector. In this scenario, an LOS frame rotates as the ship turns. Therefore we can study the effect of the rotating orientation of the LOS frame on tracking performance.

As described in Chapter 2, a model of the target dynamics and that of the measurement system are required in the Kalman filter. The accuracy of the models used in filtering affects tracking performance. We first use appropriate models to see how the coupled and decoupled ECKF's will perform. Then we gradually degrade the models to study the effects of inaccurate modeling on the performance of the coupled and decoupled ECKF's.

The Kalman filter parameters, specifying appropriate models, are summarized in Table 4.11. Inaccurate models are generated by varying σ_θ from 0.5° through 2.5° to 10.0°, with the other parameters unchanged as in Table 4.11. σ_θ specifies the standard deviation of the errors in bearing angle measurements.

Table 4.9 Ship's dynamics

Speed in a straight line motion	5m/sec
Random acceleration	$0.00625m/sec^2$
Turning speed	0.1471°/sec
Amplitude of pitching	5°
Period of pitching	11sec
Ampilitude of rolling	15°
Period of rolling	29sec

Table 4.10 Measurement system Characteristics

Scan period	2sec
Pulse repetition frequency	800Hz
Standard diviation of range measurement error	3.5m
Standard diviation of bearing angle measurement error	0.5°

Table 4.11 Parameters used in the Kalman Filter

Standard deviation of range measurement error	3.5m
Standard deviation of bearing angle measurement error	0.5°
Standard deviation of target dynamics noise	$0.01 m/sec^2$
Correlation coefficient of acceleration	0.778801

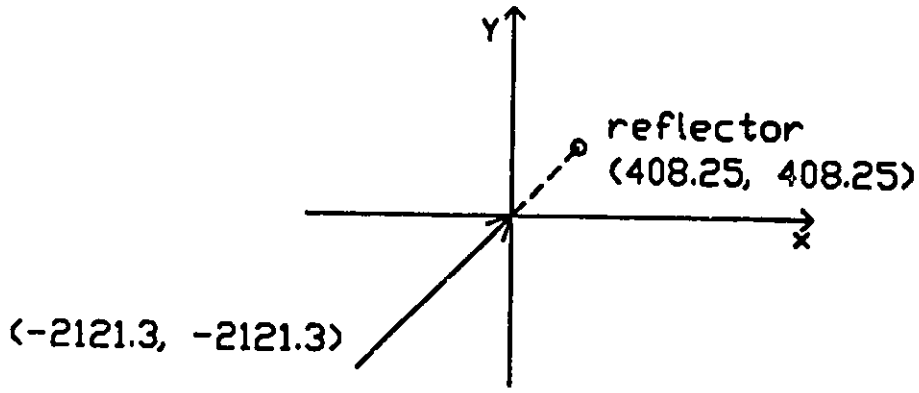


Figure 4.4 Straight line scenario in the PRAN system

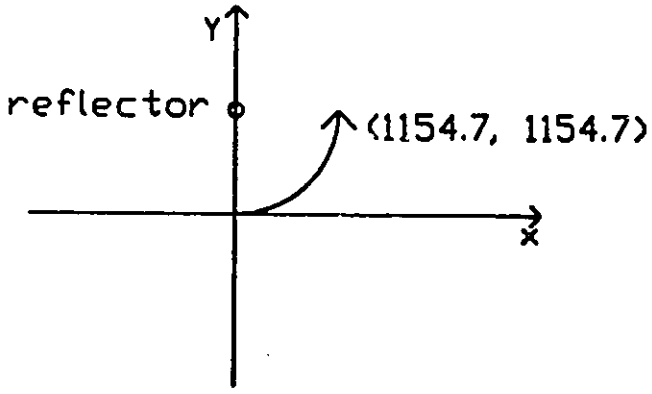


Figure 4.5 Turn scenario in the PRAN system

4.5.2 Simulation Results

Figures 4.6 (a) and 4.6 (b) show the results of tracking with an appropriate model specified in Table 4.11, for the scenario depicted in Figure 4.4. Figure 4.6 (a) is for errors in x–position estimates, whereas Figure 4.6 (b) is for errors in y–position estimates. Each figure plots two curves of ensemble–averaged squared errors; the curve drawn in a solid line is for the standard ECKF, and the curve drawn in a dashed line is for the simplified ECKF. These two types of curves are used in all the plots to compare the standard and simplified ECKF's. The ensemble–averaging is performed over 100 independent trials. No apparent difference between the performance of the standard ECKF and that of the simplified ECKF is observed. Actually, the two curves for the standard and simplified ECKF's overlap each other, such that the only solid curve is shown. It indicates that the decoupling technique does not degrade performance in this scenario, when it is applied to the system with the greatest level of coupling due to the ship's 45° travelling direction.

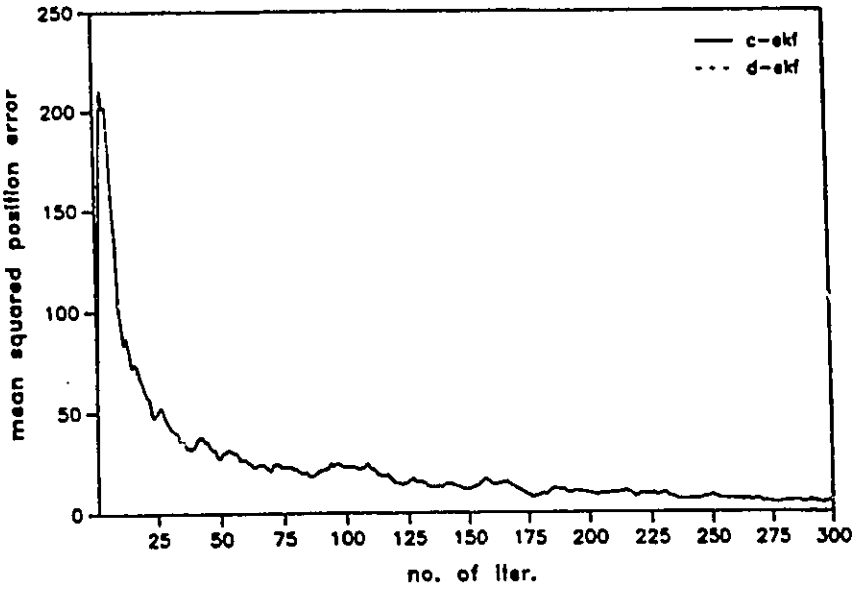
The tracking accuracy is found to improve as the ship appoaches the reflector. This is due to the fact that the accuracy is a function of σ_r and $r\sigma_\theta$, and r decreases as the ship approaches the reflector. σ_r and σ_θ denote, respectively, the standard deviation of the errors in range and bearing angle measurements, and r denotes a range measurement.

Figures 4.7 (a) and 4.7 (b) show the results of tracking with the model specified in Table 4.11, for a turning scenario as depicted in Figure 4.5. Figures 4.7 (a) and 4.7 (b) plot, respectively, errors in x–axis and y–axis position estimates. These two figures with the two indistinguishable curves for the standard and simplified ECKF's show that when appropriate models are used in both the simplified ECKF and the standard ECKF, the decoupled ECKF performs as well

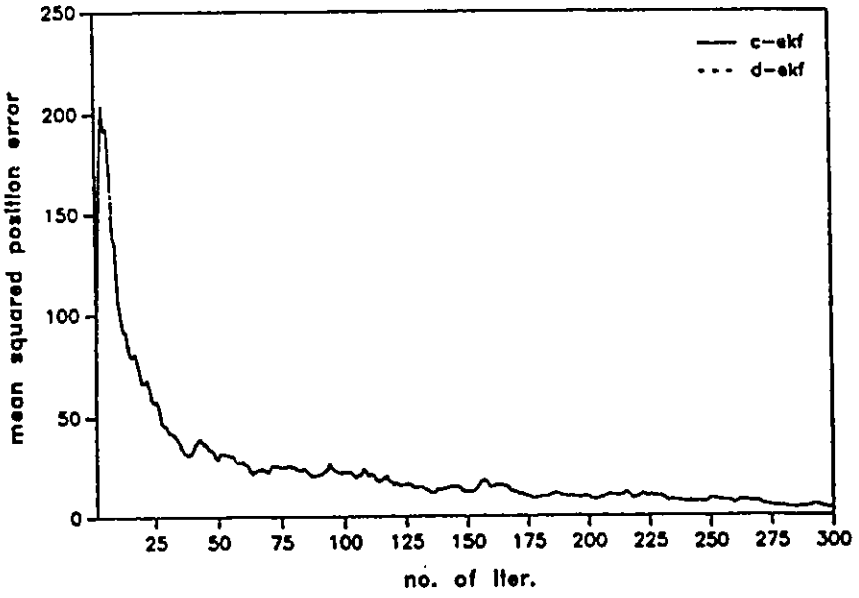
as the coupled ECKF for a typical turning scenario as that in Figure 4.5. Since the orientation of the LOS frame rotates as the ship turns, the comparable performance of the simplified ECKF to that of the standard ECKF for a turning scenario in the PRAN system verifies that the rotation of the LOS frame does not degrade performance if the rotation rate is not significant.

In the scenario of the ship turning, the x-position estimates improve while the y-position estimates degrade. This is because $r\sigma_\theta$ (= 10.0459 = 1154.7*0.0087, 0.0087 rad = 0.5 degree) is greater than σ_r (=3.5), and as the ship turns the x-position estimate errors become more a function of σ_r and less function of $r\sigma_\theta$, whereas the y-position estimate errors become more a function of $r\sigma_\theta$ and less a function of σ_r. This is depicted in Figure 4.8.

In summary, the simulations with scenarios in Figures 4.4 and 4.5 have shown that the performance of the simplified ECKF is comparable to that of the standard ECKF for tracking in the PRAN system, when appropriate models are used.

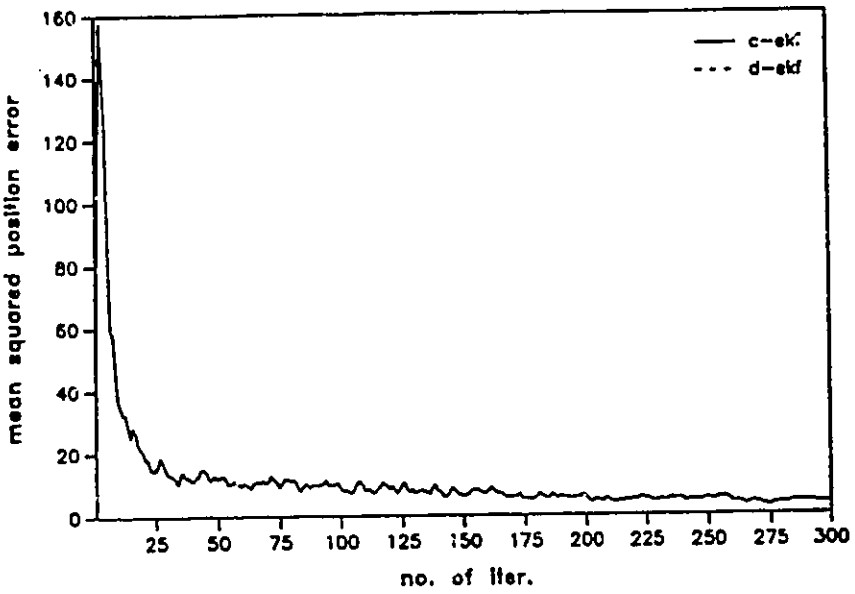


(a)

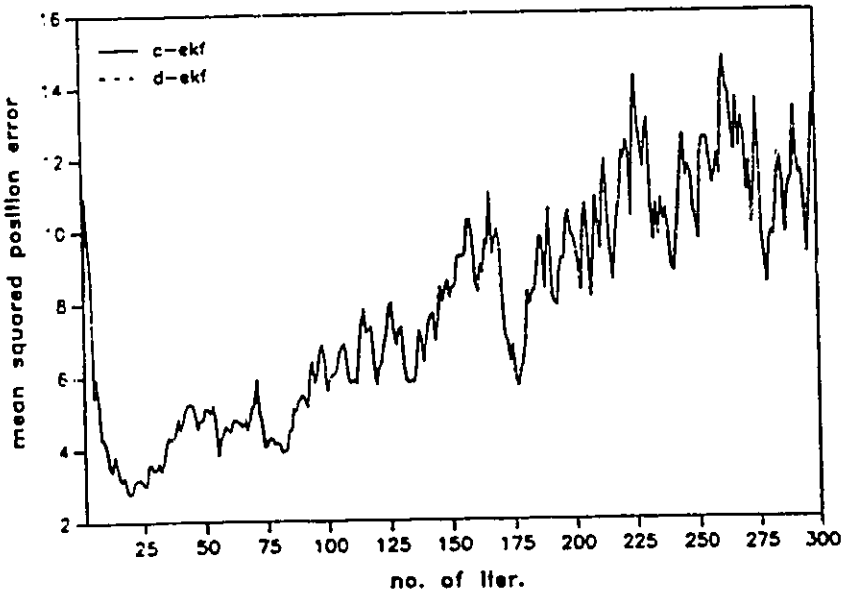


(b)

Figure 4.6 Ensemble-averaged square errors in position estimates for the straight line motion scenario
a) errors in x-position estimates
b) errors in y-position estimates



(a)



(b)

Figure 4.7 Ensemble-averaged square errors in position estimates for the turn scenario, $\sigma^2_{\theta_b}=(0.5°)^2$

a) errors in x-position estimates

b) errors in y-position estimates

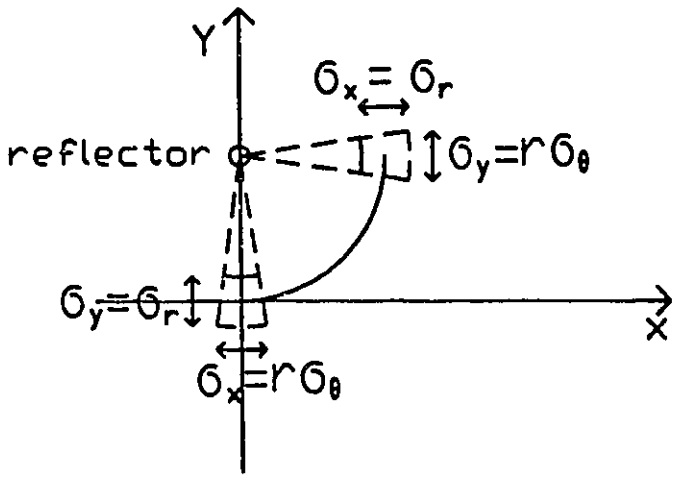
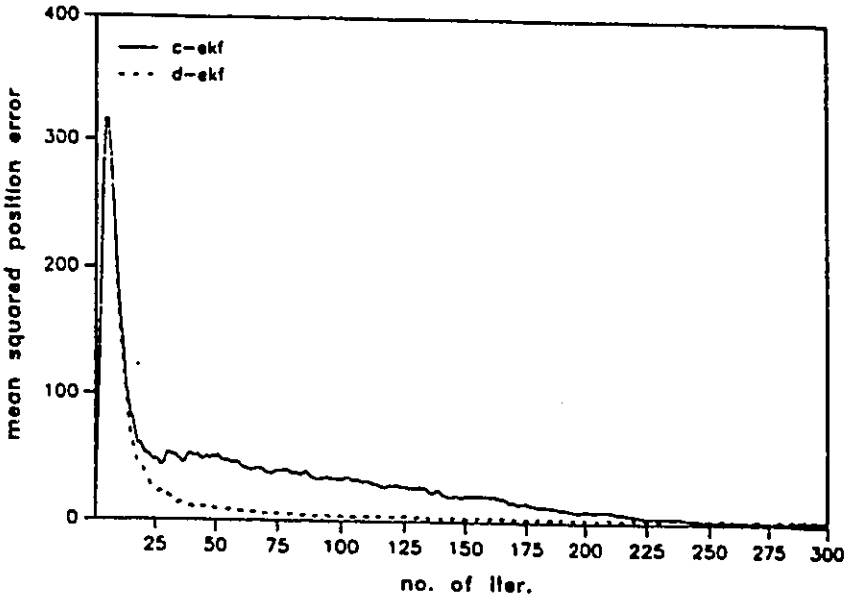
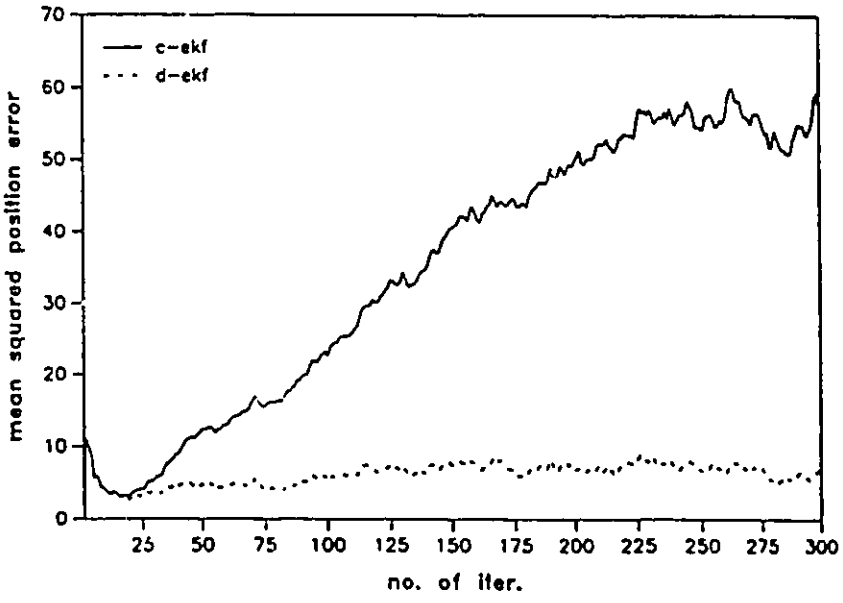


Figure 4.8 Position estimate errors in a turn scenario in the PRAN system

Figures 4.7a and 4.7b show simulation results with σ_θ = 0.5°, Figure 4.9a and 4.9b are for σ_θ = 2.5°, and Figures 4.10a and 4.10b are for σ_θ = 10.0°. They all correspond to the turning scenario shown in Figure 4.5. A comparison of the simulation results shows that as the tracking model becomes inaccurate, the performance of both the coupled and decoupled ECKF's degrades, yet the former's performance degrades more than the latter's. To determine if this phenomenon is due to differences in numerical properties of these two filters, these simulations are repeated using double precision arithmetic. The increase in the number of significant bits from a single precision to double precision does not change the simulation results. This indicates that this phenomenon is caused by the different effects of inaccurate models on these two filters. In other words, the decoupled ECKF is more robust than the coupled ECKF. This can be explained by the fact that the decoupling reduces the propagation of the effects of inaccurate parameters in the ECKF.



(a)

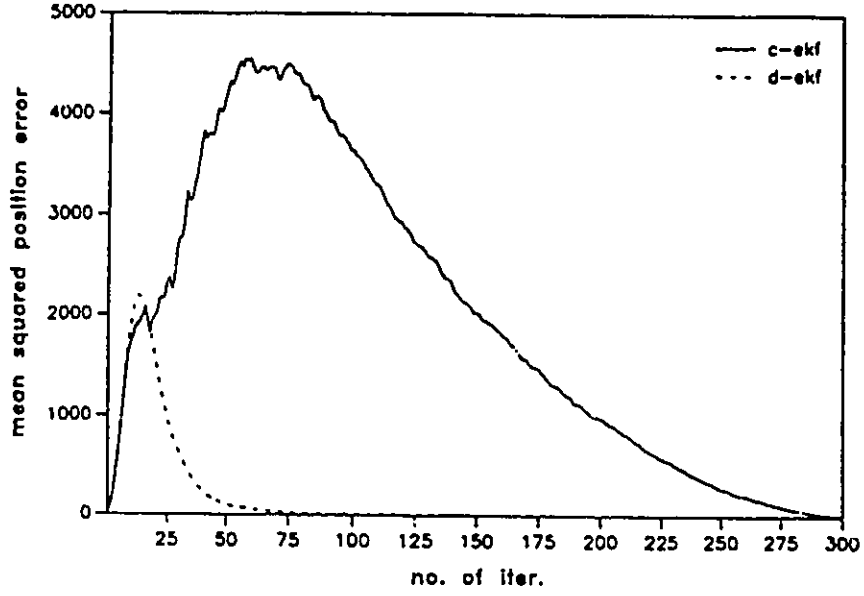


(b)

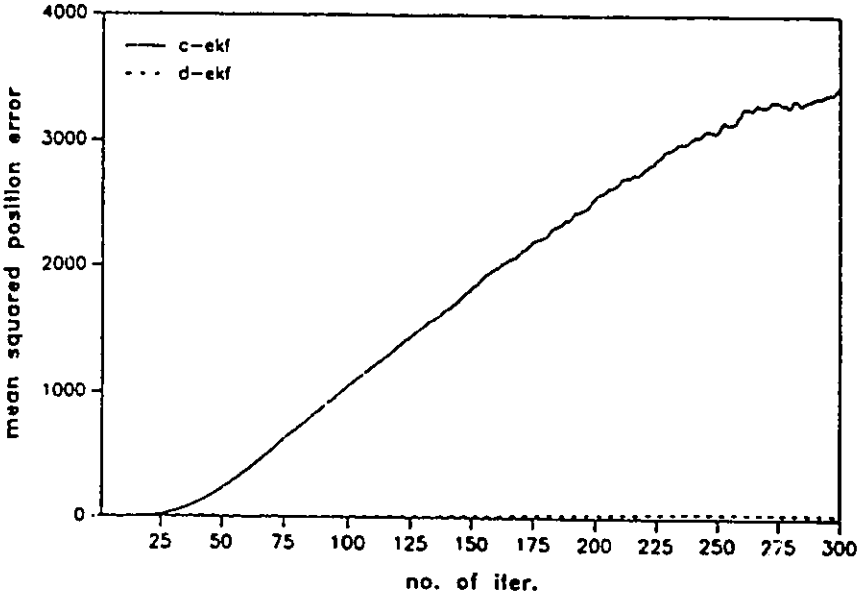
Figure 4.9 Ensemble-averaged square errors in position estimates for the turn scenario, $\sigma^2_{\theta_b} = (2.5°)^2$

a) errors in x-position estimates

b) errors in y-position estimates



(a)



(b)

Figure 4.10 Ensemble-averaged square errors in position estimates for the turn scenario, $\sigma^2_{\theta_b} = (10.0°)^2$

a) errors in x-position estimates

b) errors in y-position estimates

4.6 Summary

In this chapter, we first presented the decoupling technique which simplifies the extended covariance KF, and then we extended the use of this decoupling technique to the extended square root covariance KF.

We have shown how the use of the decoupling technique and special properties of tracking systems can simplify the extended covariance KF and the extended SRCF.

For the extended KF, the decoupling technique eliminates the need for a matrix inversion and reduces the required number of operations for the computationally demanding time and measurement updates by a factor of m^2, where m is the dimension of a tracking system. This reduction is due to the fact that the propagation of m decoupled $(\frac{n}{m})$–dimensional covariance matrices P(k)'s requires m^2 times less operations than the propagation of one coupled n–dimensional covariance matrix P(k), where n is the number of state elements and m is the tracking dimension..

We have found that in the extended SRCF the number of required operations for the propagartion of m decoupled square roots $P^{1/2}(k)$'s is less than that for the propagation of 1 coupled $P^{1/2}(k)$ by a factor of between m^2 and m. The reason for the reduction factor between m^2 and m is that the required orthogonalization for the propogation of $P^{1/2}(k)$ is a combination of an order of n^3 and n^2 operations.

We have shown that the use of special properties of tracking systems reduces computational requirements furthermore. The overall reduction ratios by the use of both the decoupling technique and special properties are found to be 24 and 13 for the required numbers of multiplications and additions for the extended covariance KF, respectively, when the number of state elements is 9 and

the dimension of a tracking system is 3. The overall reduction ratios are 8.4 and 7.5 for the required numbers of multiplications and additions for the extended SRCF, respectively.

The overall reduction ratios of 8.4 and 7.5 for the number of multiplications and additions for the ESRCF are less than the ratios of 24 and 13 for the ECKF. One of the reasons for this is that in the ECKF the computationally intensive matrix inversion is eliminated by the decoupling technique, whereas in the ESRCF the matrix inversion is already avoided by the use of forward substitution in the coupled ESRCF. Another reason is that the decoupling technique reduces the computational requirements of the propagation of P(k) more than those of the propagation of $P^{1/2}(k)$. The propagation of P(k) requires an order of n^3 operations, whereas the propagation of $P^{1/2}(k)$ requires a combination of orders of n^3 and n^2 operations.

The decoupling technique not only reduces computational requirements but also increases parallelism. In the decoupled KF, the time and measurement updates of the decoupled state estimate error covariance matrices $P_o(k)$'s can be performed separately for each axis, whereas the updates of the coupled P(k) in the coupled KF have to be performed together for all the axes. The fact that $P_o(k)$'s in the decoupled KF can be separately updated for each axis means that the updates can be performed for each axis in parallel. The parallel updates of m decoupled $P_o(k)$'s require m times less computational time than the serial update of m $P_o(k)$'s. In the development of parallel architectures where parallelism is explored to reduce computational time, the parallelism in the updates of $P_o(k)$ may be utilized.

We have found that the performance of the simplified ECKF is generally comparable to that of the conventional ECKF, using two typical tracking

examples in a Precise Radar Aided Navigation system. Furthermore, we have found that the simplified ECKF is more robust to errors in target modelling than the standard ECKF, since its decoupled nature reduces the propagation of the effects of inaccurate modelling.

CHAPTER 5

PARALLEL IMPLEMENTATION OF THE EXTENDED COVARIANCE KF

The computational requirements of the Kalman filter should be minimized and parallelism should be maximized to reduce the required hardware and computational time for the parallel implementation of the KF. Parallel processing decreases the required computational time by allowing more than one operation to be performed simultaneously.

In Chapter 3, we have shown that the computational requirements of the KF can be reduced by reformulating the KF equations such that the common terms are calculated only once and reused whenever necessary, and we have shown that parallelism can be doubled by separating the KF into two loosely independent sets of the KF equations.

In Chapter 4, we have shown that the use of the decoupling technique and special properties of certain matrices in the tracking Kalman filter reduces significantly the computational requirements of the extended covariance Kalman filter and increases parallelism by the separation of the state estimate error covariance matrix P(k) into a number of smaller decoupled matrices. In the decoupled KF, the updates of matrix P(k) can be separately performed for each axis in parallel.

The reduction in computational requirements and the increase in parallelism by the application of the decoupling technique and special properties

of certain matrices in the tracking KF are much greater than those by the reformulation and separation of the KF equations, described in Chapter 3. Hence, the simplified extended covariance KF through the use of the decoupling technique and special properties is selected in the parallel implementation of the extended covariance KF for tracking applications.

We develop a parallel architecture for the simplified ECKF, bearing in mind desirable characteristics for parallel architectures such as modularity, regularity, local communication, and high degree of pipeling and parallelism. However, when a pipeline processing architecture with local communication has a significant time delay and requires a large number of delay elements to feed input data in a skewed manner, a parallel system with global communication is considered. In the parallel system with global communication, data are broadcasted to all the necessary processing elements at the same time, so that the time delay is minimal and additional delay elements are not necessary.

In Section 5.1, we first give an overview of the proposed architecture, and then describe the architecture in detail. In Section 5.2, we examine how the decoupling technique reduces hardware and computational time requirements to gain insights into the effects of the decoupling technique on a parallel architecture. In Section 5.3, we summarize the required number of processing elements and operations. In Section 5.4, we evaluate the proposed architecture in comparison to other architectures. Finally, in Section 5.5, we present the summary of this chpater.

Throughout this chapter, 2–dimensional tracking is employed for simplicity in explaining implementaion, unless 2– and 3–dimensional implementations are both needed to be described.

5.1 Architecture Description

The decoupled extended covariance KF, described in Equations (4.27–4.32), consists of 3 parts, as follows:

a) Processing in the LOS frame.

b) Coordinate transformation of the Kalman gain.

c) Processing in the reference Cartesian coordinate frame.

The architecture for the decoupled extended covariance KF which consists of 3 parts may be split into 3 parts correspondingly, as shown below:

a) The processor for the LOS frame computes $K_o(k)$, $P_o(k)$ and $P_o(k+1|k)$, as shown by

$$K_o(k) = P_o(k|k-1)\, H_o^T(k)\, (\, H_o(k)\, P_o(k|k-1)\, H_o^T(k) + R_o(k)\,)^{-1} \qquad (5.1)$$

$$P_o(k) = (\, I - K_o(k)\, H_o(k)\,)\, P_o(k|k-1) \qquad (5.2)$$

$$P_o(k+1|k) = \phi(k)\, P_o(k)\, \phi^T(k) + Q_o(k) \qquad (5.3)$$

b) The processor for the coordinate transformation computes K(k) from $K_o(k)$ by using the Jacobian transformation:

$$K(k) = \begin{bmatrix} F_1 & 0 & 0 \\ 0 & F_1 & 0 \\ 0 & 0 & F_1 \end{bmatrix} K_o(k) \qquad (5.4)$$

For 2–dimensional tracking, we have

$$F_1 = \begin{bmatrix} \cos\theta_A & -\sin\theta_A \\ \sin\theta_A & \cos\theta_A \end{bmatrix}$$

where θ_A denotes azimuth angle.

For 3--dimensional tracking, we have

$$F_1 = \begin{bmatrix} \cos\theta_A\cos\theta_E & -\sin\theta_A\cos\theta_E & -\cos\theta_A\sin\theta_E \\ \sin\theta_A\cos\theta_E & \cos\theta_A\cos\theta_E & -\sin\theta_A\sin\theta_E \\ \sin\theta_E & 0 & \cos\theta_E \end{bmatrix}$$

where θ_A and θ_E denote azimuth and elevation angles, respectively.

c) The Processor for the reference Cartesian coordinates frame computes $\hat{X}(k)$ and $\hat{X}(k+1|k)$, as shown by

$$\hat{X}(k) = \hat{X}(k|k-1) + K(k) \left(Z(k) - h(\hat{X}(k|k-1)) \right) \quad (5.5)$$

$$\hat{X}(k+1|k) = \phi(k)\, \hat{X}(k) \quad (5.6)$$

Figure 5.1 shows a block diagram of the tracking KF implementation. In this block diagram, as described above, there are three processors. Data flows among processors are indicated by arrows. The three processors from the left to right are referred to as Processor 1, 2, and 3, from now on. Processor 1, located on the lefthand side in Figure 5.1, updates the state estimate error covariance matrix $P_o(k)$ for the LOS frame, calculates the Kalman gain $K_o(k)$ for the LOS frame, and passes $K_o(k)$ to Processor 2. After receiving $K_o(k)$ from Processor 1, Processor 2 computes K(k) for the reference Cartesian frame by applying the

Jacobian transformation to $K_o(k)$. Processor 3 makes the state estimate $\hat{X}(k)$ as a combination of $\hat{X}(k|k-1)$ and $Z(k)$, using $K(k)$ from Processor 2. Note that Processor 3 includes a coordinate transformation, $h(\hat{X}(k|k-1))$, of $\hat{X}(k|k-1)$ from the reference Cartesian to polar coordinates, and the calculation of the correction term, $Z(k)-h(\hat{X}(k|k-1))$.

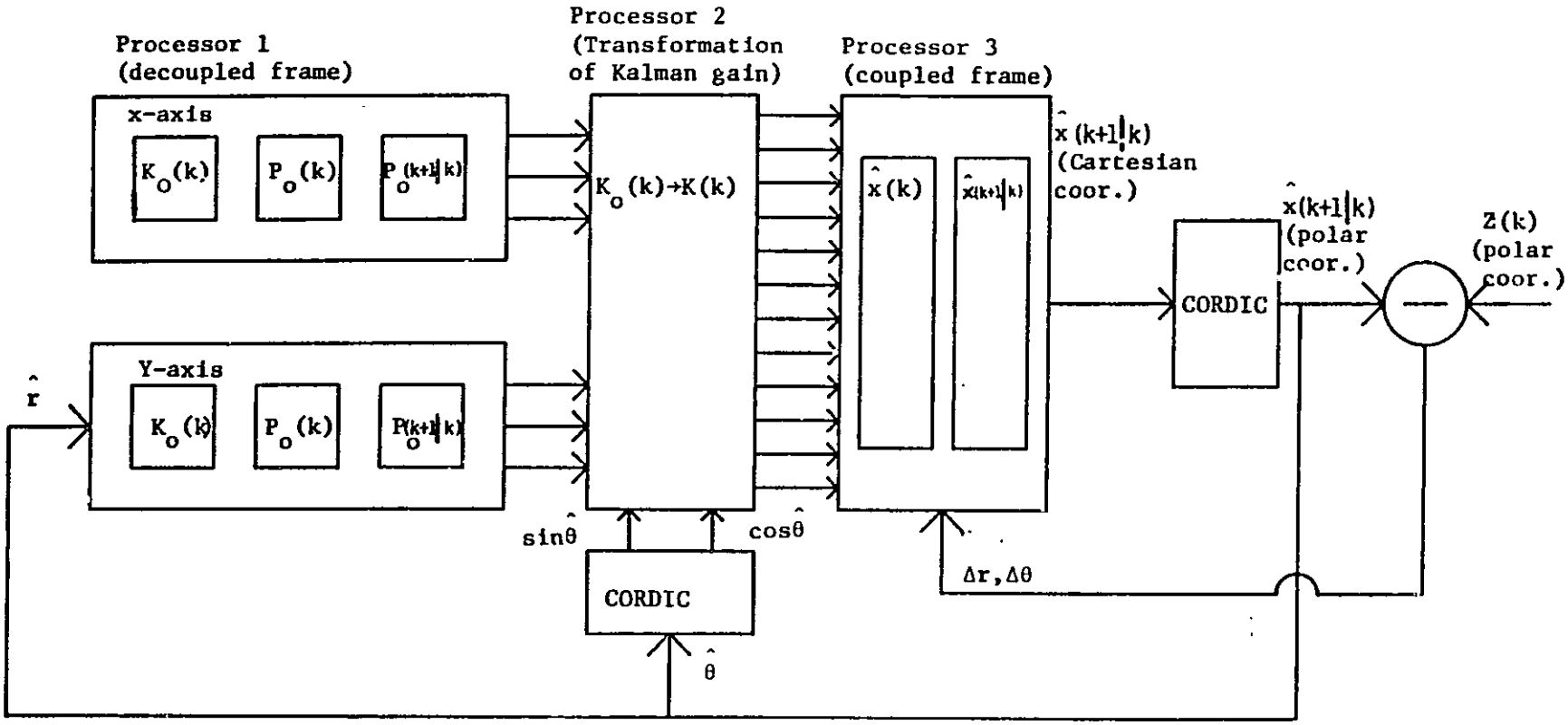


Figure 5.1 Block diagram of the ECKF

5.1.1 Processor for the LOS coordinates

The examination of Equations (5.1), (5.2), and (5.3) indicates that the processor implementing these equations should be designed to perform matrix–matrix and matrix–vector multiplications. The proposed processor for Equations (5.1), (5.2), and (5.3) is shown in Figure 5.2. It consists of a 2–dimensional orthogonally connected systolic array and external multipliers. This processor possesses desirable characteristics for parallel architectures such as regularity, local communication, and high degree of pipelining and parallelism. The 2–dimensional array is first initialized to $P_o(0|-1)$, and stores $P_o(k)$ and $P_o(k|k-1)$ alternately.

We now describe how Equations (5.1), (5.2), and (5.3) are implemented on the proposed architecture. Due to similarities between the processing for the x–, y– and z–axes, the implementations for only the x–axis is described.

Equation (5.1) for the x–axis

$$K_o(k) = P_o(k|k-1)\ H_o^T(k)\ (\ H_o(k)\ P_o(k|k-1)\ H_o^T(k)\ +\ R_o(k)\ )^{-1} \qquad (5.1)$$

seems very complex to implement, for this equation requires a number of matrix–matrix multiplications and a matrix inversion. However, it was found in Section 4.3.2 that Equation (5.1) for the x–axis can be simplified using the sparse nature of $H_{o-x}(k)$ as follows:

$$K_{o-x}(k) = \begin{bmatrix} \bar{p}_{xx} \\ \bar{p}_{\dot{x}x} \\ \bar{p}_{\dot{x}\cdot x} \end{bmatrix} (\bar{p}_{xx} + \sigma_r^2)^{-1} \qquad (5.1\text{–}x)$$

The simplified equation (5.1) above is implemented on the proposed architecture by reading out either the first row or column of P(k|k–1), (p_{xx}(k|k–1), $p_{\dot{x}x}$(k|k–1), $p_{\dot{x}\cdot x}$(k|k–1)), stored in the systolic array and dividing it by (p_{xx}(k|k–1) + σ_r^2), as shown in Figure 5.3. Both the first column and the first row of P(k|k–1) store the same vector, because P(k|k–1) is a symmetric matrix. Note that one of the most difficult KF equations to implement, which requires a large number of matrix–matrix multiplications, is implemented by reading out the row of the processor and scaling it. This confirms that the simplified extended covariance KF is much easier to implement than the standard extended covariance KF.

The implementation of Equation (5.1) requires total 5 clock cycles: 1 cycle for reading out (p_{xx}(k|k–1), $p_{\dot{x}x}$(k|k–1), $p_{\dot{x}\cdot x}$(k|k–1)) and 4 cycles for calculation of (p_{xx}(k|k–1) + σ_r^2) and dividing (p_{xx}(k|k–1), $p_{\dot{x}x}$(k|k–1), $p_{\dot{x}\cdot x}$(k|k–1)) by (p_{xx}(k|k–1) + σ_r^2).

Equation (5.2) which updates the state estimate error covariance matrix

$$P_o(k) = (I - K_o(k)\, H_o(k))\, P_o(k|k-1) \tag{5.2}$$

requires a number of matrix–matrix multiplications, as it is. However, as described in Section 4.3.2, the simplified equation (5.2) requires only a row–column vector multiplication amd matrix–matrix subtraction. The simplified equation (5.2) for the x–axis is shown below:

$$P_{o-x}(k) = \begin{bmatrix} \bar{p}_{xx} & \bar{p}_{x\dot{x}} & \bar{p}_{x\dot{x}\cdot} \\ \bar{p}_{\dot{x}x} & \bar{p}_{\dot{x}\dot{x}} & \bar{p}_{\dot{x}\dot{x}\cdot} \\ \bar{p}_{\dot{x}\cdot x} & \bar{p}_{\dot{x}\cdot\dot{x}} & \bar{p}_{\dot{x}\cdot\cdot\dot{x}} \end{bmatrix} - \begin{bmatrix} k_{xr} \\ k_{\dot{x}r} \\ k_{\dot{x}\cdot r} \end{bmatrix} [\, p_{xx}\ p_{x\dot{x}}\ p_{x\dot{x}\cdot}] \tag{5.2–x}$$

Figure 5.4 shows the implementation of the simplified equation (5.2). In Figure 5.4, Equation (5.2) is evaluated by an outer product multiplication of vectors, $(-p_{xx} \; -p_{x\dot{x}} \; -p_{x\dot{x}\cdot})$ and $(k_{xr} \; k_{\dot{x}r} \; k_{\dot{x}\cdot r})$, and adding each product to what has already been stored $P_o(k|k-1)$. The former is fed in from the top, and the latter from the left. It is worth noting that the processing of Equation (5.2) does not require any vector fetch from the systolic array, because the required two vectors are already available from Equation (5.1).

The implementation of the simplified equation (5.2) in Figure 5.4 requires 6 clock cycles from the time the vectors $(-p_{xx} \; -p_{x\dot{x}} \; -p_{x\dot{x}\cdot})$ and $(k_{xr} \; k_{\dot{x}r} \; k_{\dot{x}\cdot r})$ enter the systolic array to the time of completion of Equation (5.2). This computational time requirement of 6 clock cycles is small compared to the computational time requirement for the implementation of Equation (5.2) without simplication.

Equation (5.3)

$$P_o(k+1|k) = \phi(k) \; P_o(k) \; \phi^T(k) + Q(k)$$

can be implemented in various ways ([28],[29]). For example, the term, $\phi(k) \; P_o(k) \; \phi^T(k)$, can be implemented in the sequence of two matrix–matrix multiplications, described in Section 3.2.1. This method requires 3n–2 clock cycles for each matrix–matrix multiplication. The total required time is thus 6n–4 clock cycles for two matrix–matrix multiplications.

However, the properties of the band matrix ϕ can be exploited, as described below:

Property 1:

$$\phi(k)\ P(k) = \begin{bmatrix} a_{11} & a_{12} & 0 \\ 0 & a_{22} & a_{23} \\ 0 & 0 & a_{33} \end{bmatrix} \begin{bmatrix} r1 \text{ ——————} \\ r2 \text{ ——————} \\ r3 \text{ ——————} \end{bmatrix} = \begin{bmatrix} a_{11}r1 + a_{12}r2 \text{ ——} \\ a_{22}r2 + a_{23}r3 \text{ ——} \\ a_{33}r3 \text{ ——————} \end{bmatrix}$$

Property 2:

$$C(k)\ \ \phi(k) = \begin{bmatrix} c1 & c2 & c3 \\ | & | & | \\ | & | & | \end{bmatrix} \begin{bmatrix} b_{11} & 0 & 0 \\ b_{21} & b_{22} & 0 \\ 0 & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} b_{11}c1+b_{21}c2 & | & | \\ | & b_{22}c2+b_{32}c3 & | \\ | & | & b_{33}c3 \end{bmatrix}$$

Property 1 indicates that each row of the output matrix is a linear combination of one or two rows of the matrix P(k). Therefore, this band matrix multiplication can be performed as follows:

a) Move the elements of each row upward, as shown in Figure 5.5a.

b) Multiply the element already stored in a processing element by the corresponding element of $\phi(k)$, and multiply the element moved upward in step (a) by another element of $\phi(k)$ that corresponds to it.

c) Sum the results of the two multiplications in step (b).

Similarly, Property 2 shows that each column of the output matrix is a linear combination of columns of Q(k). Consequently, it may be implemented in the same way as property 1, except for the elements of each column being moved to the left, as shown in Figure 5.5c. This scheme requires bidirectional horizontal and vertical interconnections among processing elements. Figures 5.5b and 5.5d show the functionality of each processing element.

This band matrix multiplication method is much more efficient than the conventional method described in Section 3.2.1 for two reasons: (a) It does not require that any matrix is read into the systolic array for the multiplication. (b) It takes only an order of 1 steps, exactly 2 multiplications and 1 addition, compared to an order of n steps for the conventional method with an order of n^2 processing elements, where n is the size of a matrix.

The addition of Q(k) to the output of 2 band matrix multiplications may be easily realized by adding the element of the prestored former matrix to the element of the latter matrix at each processing element.

Note that $\phi(k)$, $\phi^T(k)$,and Q(k) are prestored in the local memory of each processing element. This can be easily realized because $\phi(k)$ and Q(k) are both constant.

The total required number of clock cycles for Equation (5.3) is 5: 2 cycles for the multiplication of $\phi(k)$ and P(k), 2 cycles for the multiplication of the product $\phi(k)P(k)$ and $\phi^T(k)$, and 1 cycle for the addition of the product $\phi(k)P(k)\phi^T(k)$ and Q(k).

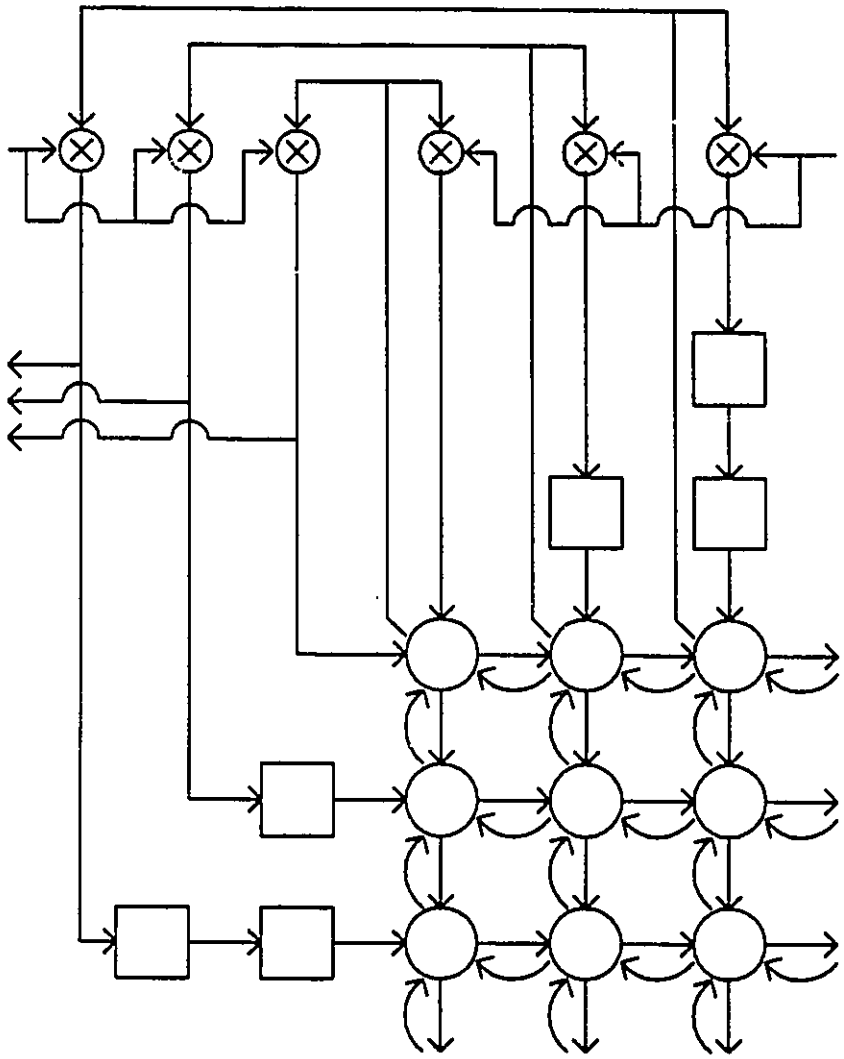


Figure 5.2 Processor for the LOS frame

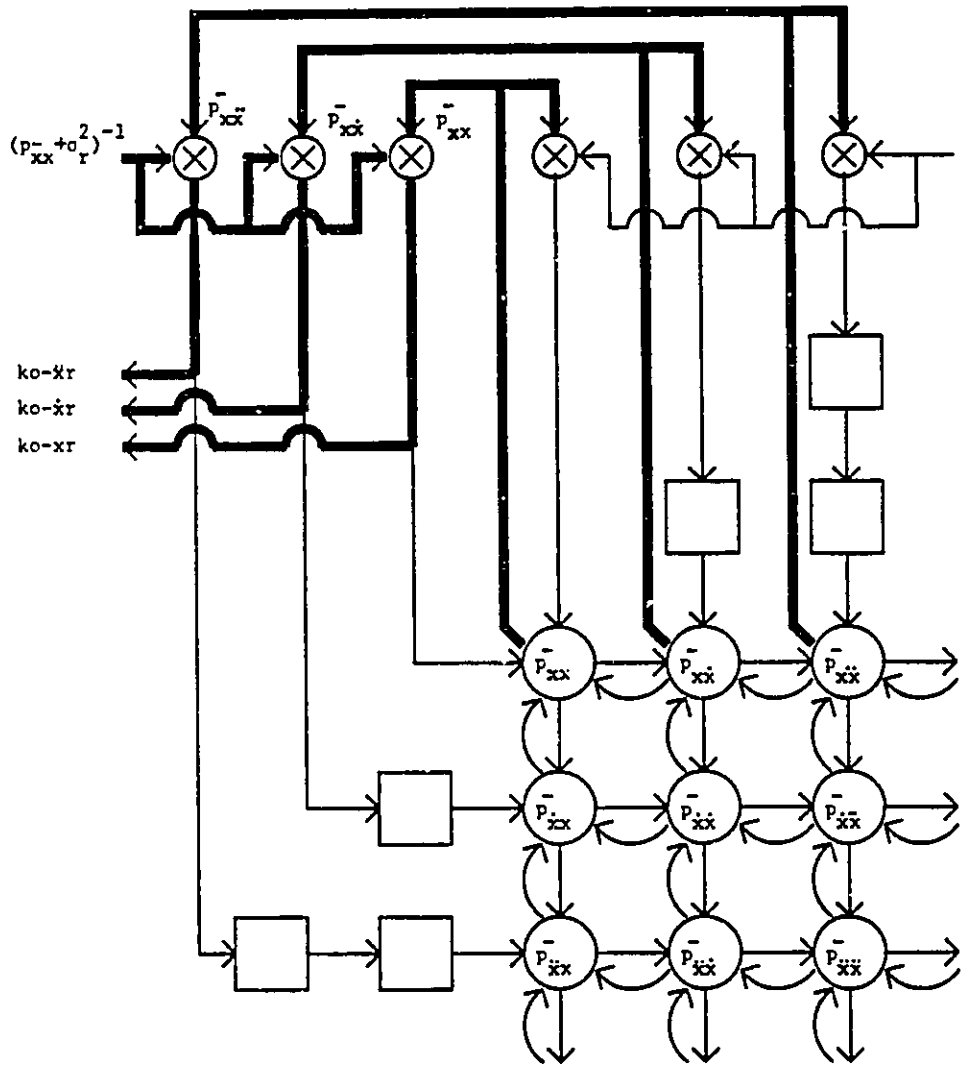


Figure 5.3 Calculation of the Kalman gain in the LOS frame $K_o(k)$

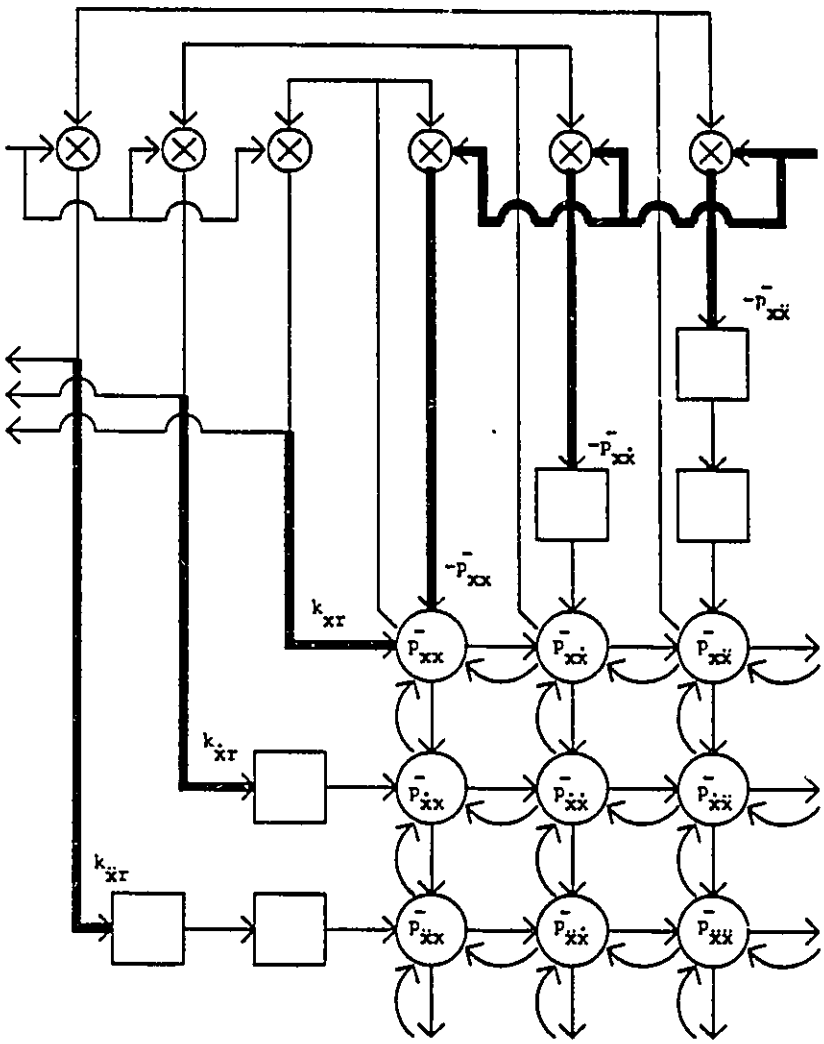
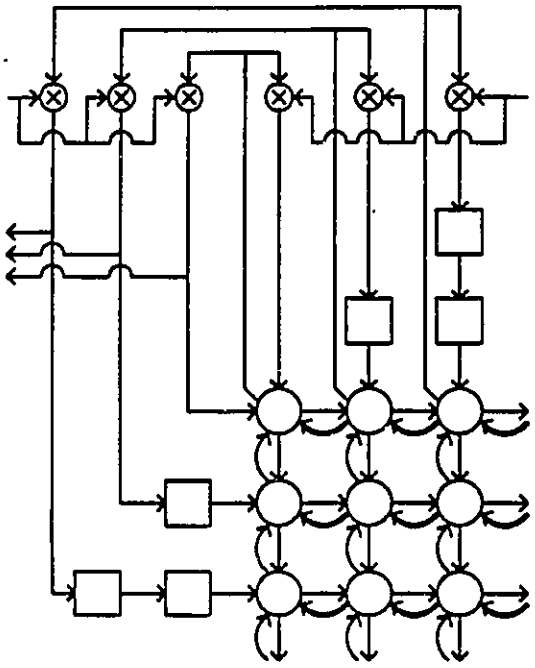
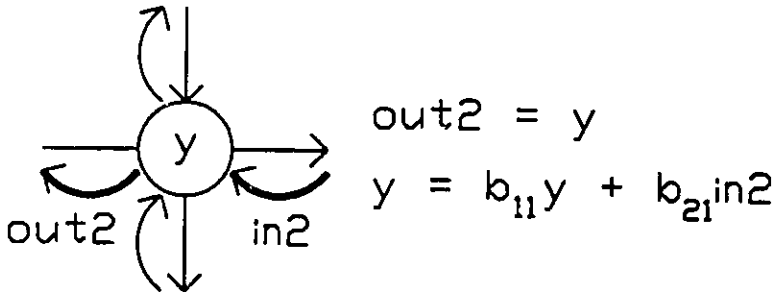


Figure 5.4 Calculation of $P_o(k)$



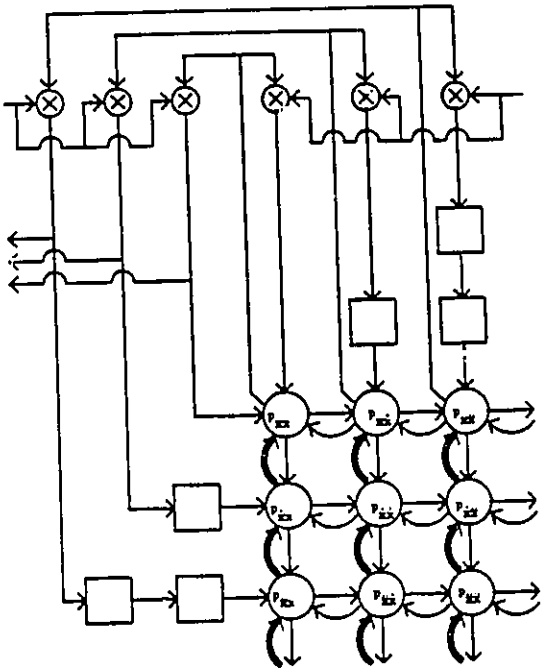
(a)



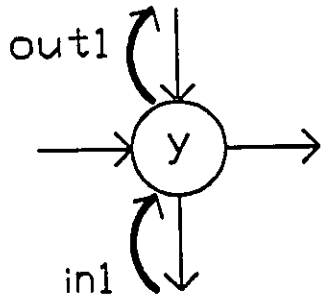
(b)

Figure 5.5 Calculation of $P_o(k+1|k)$

a) multiplication of $\phi(k)$ and P(k)
b) funcion of the cell for Step (a)
c) multiplication of C(k) and $\phi(k)$
d) function of the cell for Step (c)



(c)



$out1 = y$

$y = a_{11}y + a_{12}in1$

(d)

5.1.2 Processor for the Coordinate Transformation of the Kalman Gain

After receiving the Kalman gain $K_o(k)$ for the LOS frame from Processor 1, Processor 2 which implements Equation (5.4) transforms the Kalman gain from the LOS frame to the reference Cartesian frame, and passes the Kalman gain K(k) for the reference Cartesian frame to Processor 3.

Equation (5.4) for 2–dimensional tracking is expanded as follows:

$$K(k)=\begin{bmatrix} k_{xr} & k_{x\theta_A} \\ k_{yr} & k_{y\theta_A} \\ k_{\dot{x}r} & k_{\dot{y}\theta_A} \\ k_{\dot{y}r} & k_{\dot{y}\theta_A} \\ k_{\dot{x}\cdot r} & k_{\dot{y}\cdot\theta_A} \\ k_{\dot{y}\cdot r} & k_{\dot{y}\cdot\theta_A} \end{bmatrix} = \begin{bmatrix} F_1 & | & 0 & | & 0 \\ & | & & | & \\ 0 & | & F_1 & | & 0 \\ & | & & | & \\ 0 & | & 0 & | & F_1 \\ & | & & | & \end{bmatrix} \begin{bmatrix} k_{o-xr} & 0 \\ 0 & k_{o-y\theta_A} \\ k_{o-\dot{x}r} & 0 \\ 0 & k_{o-\dot{y}\theta_A} \\ k_{o-\dot{x}\cdot r} & 0 \\ 0 & k_{o-\dot{y}\cdot\theta_A} \end{bmatrix}$$

$$F_1(k) = \begin{bmatrix} \cos\theta_A & -\sin\theta_A \\ \sin\theta_A & \cos\theta_A \end{bmatrix}$$

This expanded form indicates that each component of the Kalman gain matrix K(k) is a product of a component of matrix $K_o(k)$ and a component of matrix $F_1(k)$. For example, the element $k_{xr}(k)$ of matrix K(k)

$$k_{xr}(k) = \cos\theta_A(k)\, k_{o-xr}(k)$$

This means that the implementation of Equation (5.4) should consist of a set of

multipliers and a CORDIC to evaluate $\sin\theta_A$ and $\cos\theta_A$ in $F_1(k)$, given θ_A.

The implementation of Equation (5.4) for 2-dimensional tracking is shown in Figure 5.6; it requires 12 multipliers and 1 CORDIC. In this implementation, the elements of the Kalman gain matrix $K_o(k)$ for the LOS frame and the elements of the rotation matrix $F_1(k)$ are simultaneously fed from the left and from the top respectively. The trigonometric functions, $\sin\theta$ and $\cos\theta$ of $F_1(k)$ are performed using a CORDIC. The implementation of a CORDIC is discussed in Section 5.1.4. In this implementation, all the elements of the Kalman gain matrix K(k) are generated at the same time.

The processing of Equation (5.4) requires 2 clock cycles from the broadcasting of input data to the generation of K(k): one cycle to evaluate $F_1(k)$, and another cycle to multiply $F_1(K)$ and $K_o(k)$.

The parallel processing architecture with global communication for Equation 5.4 is referred to as a broadcast processing architecture, since data are broadcasted to all the necessary processing elements. The parallel processing architecture in Figure 5.6 may be modified to produce the gain matrix elements in a pipelined manner, as shown in Figure 5.7. In Figure 5.7, the data, the elements of $K_o(k)$, $\sin\theta$ and $\cos\theta$, are passed one element at a time through the multipliers. The architecture in Figure 5.7 is referred to as a pipeline processing architecture. Generally, the pipeline architecture requires a simpler data bus connection and fewer processing elements than the broadcast architecture. However, the former needs to introduce a skew in input data streams to synchronize incoming data streams at processing elements, and the former takes more computation time than the latter.

For the implementation of Equation (5.4), the broadcast processing architecture in Figure 5.6 seems more suitable than the one in Figure 5.7,

because the former produces K(k) without any time delay, and it does not need to introduce a skew in the input data streams for synchronization, requiring additional hardware. Yet, the suitable implementaion should be selected on the basis of design criteria.

Similar to Equation (5.4) for 2–dimensional tracking, Equation (5.4) for 3–dimensional tracking is expanded as follows:

$$K(k) = \begin{bmatrix} k_{xr} & k_{x\theta_A} & k_{x\theta_E} \\ k_{yr} & k_{y\theta_A} & k_{y\theta_E} \\ k_{zr} & k_{z\theta_A} & k_{z\theta_E} \\ k_{\dot{x}r} & k_{\dot{x}\theta_A} & k_{\dot{x}\theta_E} \\ k_{\dot{y}r} & k_{\dot{y}\theta_A} & k_{\dot{y}\theta_E} \\ k_{\dot{z}r} & k_{\dot{z}\theta_A} & k_{\dot{z}\theta_E} \\ k_{\dot{x}\cdot r} & k_{\dot{x}\cdot\theta_A} & k_{\dot{x}\cdot\theta_E} \\ k_{\dot{y}\cdot r} & k_{\dot{y}\cdot\theta_A} & k_{\dot{y}\cdot\theta_E} \\ k_{\dot{z}\cdot r} & k_{\dot{z}\cdot\theta_A} & k_{\dot{z}\cdot\theta_E} \end{bmatrix}$$

$$= \left[\begin{array}{c|c|c} & & \\ F_1 & 0 & 0 \\ & & \\ \hline & & \\ 0 & F_1 & 0 \\ & & \\ \hline & & \\ & & F_1 \\ & & \end{array}\right] \begin{bmatrix} k_{o-xr} & 0 & 0 \\ 0 & k_{o-y\theta_A} & 0 \\ 0 & 0 & k_{o-z\theta_E} \\ k_{o-\dot{x}r} & 0 & 0 \\ 0 & k_{o-\dot{y}\theta_A} & 0 \\ 0 & 0 & k_{o-\dot{z}\theta_E} \\ k_{o-\dot{x}\cdot r} & 0 & 0 \\ & k_{o-\dot{y}\theta_A} & 0 \\ 0 & & k_{o-\dot{z}\theta_E} \end{bmatrix}$$

where

$$F_1 = \begin{bmatrix} \cos\theta_A\cos\theta_E & -\sin\theta_A\cos\theta_E & -\cos\theta_A\sin\theta_E \\ \sin\theta_A\cos\theta_E & \cos\theta_A\cos\theta_E & -\sin\theta_A\sin\theta_E \\ \sin\theta_E & 0 & \cos\theta_E \end{bmatrix}$$

Similar to a 2–dimensional tracking application, the expanded form of Equation (5.4) indicates that each component of the Kalman gain K(k) for 3–dimensional tracking is a product of components of $K_o(k)$ and $F_1(k)$. For example,

$$k_{xr}(k) = (\cos\theta_A \cos\theta_E) \quad (k_{o-xr})$$

The only difference between 2– and 3–dimensional tracking systems is that a component of $F_1(k)$ for 3–dimensional tracking is a product of two trigonometric functions, whereas that for 2–dimensional tracking is one trigonometric function.

A broadcast processing implementation of Equation (5.4) for 3–dimensional tracking is shown in Figure 5.8; it requires 28 multipliers and 2 CORDIC's. This architecture is similar to the architecture in Figure 5.6, except that this architecture has 2 CORDIC's instead of 1 CORDIC, and more multipliers. The 2 CORDIC's and 4 multipliers at the top of Figure 5.8 implement trigonometric functions, $\sin\theta_A$, $\cos\theta_A$, $\sin\theta_E$, and $\cos\theta_E$, given θ_A and θ_E, and the multiplications of these trigonmetric functions. This implementation requires 3 clock cycles to evaluate Equation 5.4: 2 cycles to implement $F_1(k)$ and 1 cycle to multiply $F_1(k)$ and $K_o(k)$.

This architecture can be easily modified to be a pipeline processing architecture in the same way the broadcast processing implementation for 2–dimensional tracking was modified to be a pipeline processing architecture.

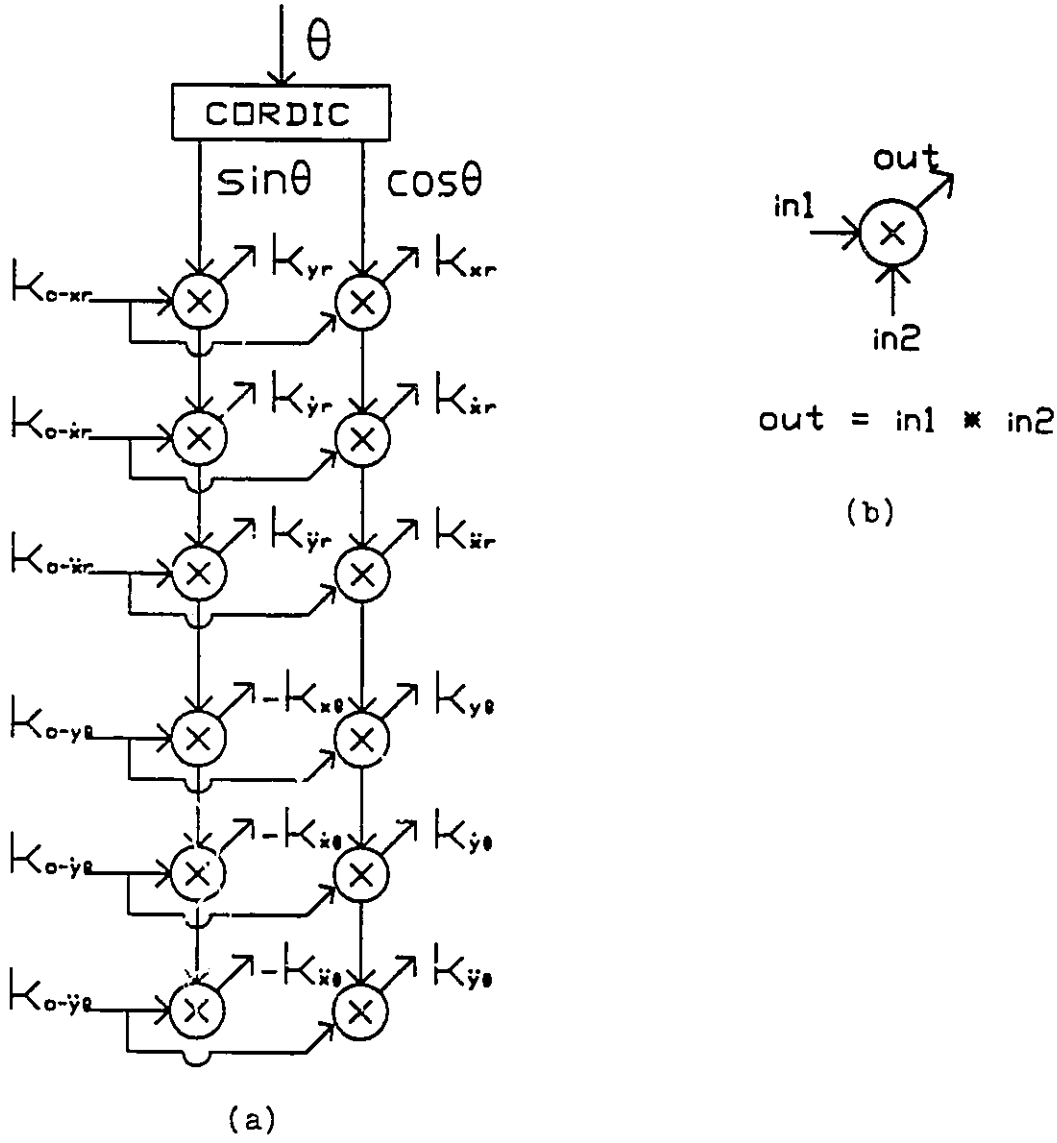


Figure 5.6 Transformation of the Kalman gain in a broadcasting manner
a) structure
b) function of a cell

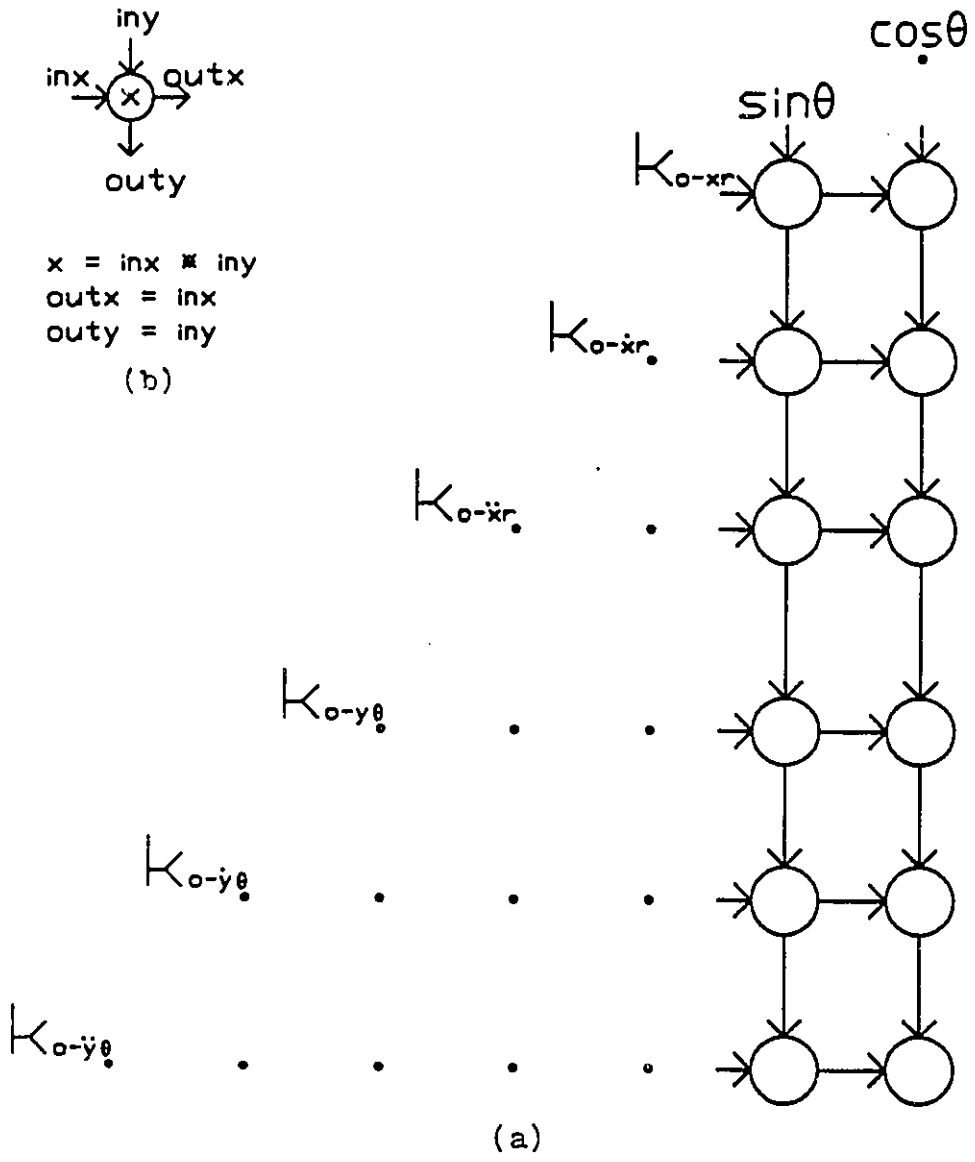


Figure 5.7 Transformation of the Kalman gain in a pipelining manner
a) structure
b) function of a cell

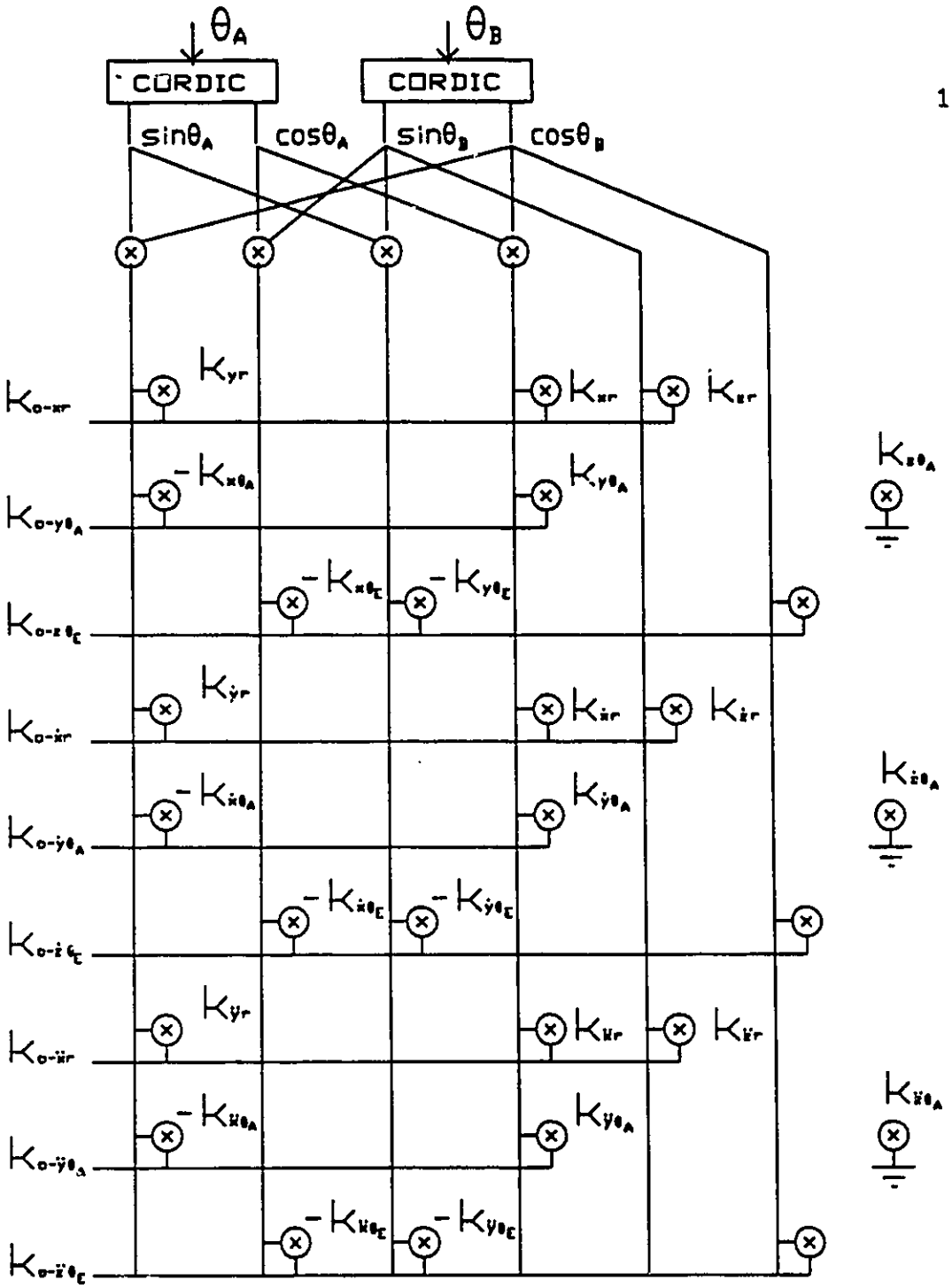


Figure 5.8 Transformation of the Kalman gain for 3-dimensional tracking in a broadcasting manner

5.1.3 Processor for the Reference Cartesian Coordinates.

A processor for the reference Cartesian coordinates estimates and predicts a state vector using Equations (5.5) and (5.6). These two equations require a coordinate transformation, matrix–vector multiplication, and vector–vector addition.

Equation (5.5) which estimates the state vector

$$\hat{X}(k) = \hat{X}(k|k-1) + K(k)\,(Z(k) - h(\hat{X}(k|k-1))) \tag{5.5}$$

may be separated into the following three parts:

a) $Z(k|k-1) = h(X(k|k-1))$

b) $\Delta Z = Z(k) - Z(k|k-1)$

c) $\hat{X}(k) = \hat{X}(k|k-1) + K(k)\,\Delta Z$

The first part (a) is a coordinate transformation of $\hat{X}(k|k-1)$. It requires 1 CORDIC for 2–dimensional coordinate transformation, and 2 CORDIC's for 3–dimensional coordinate transformation, as explained in Section 5.1.4. The computational requirements of the coordinate transformation are 1 and 2 clock cycles for 2– and 3–dimensional tracking systems, respectively. Note that a CORDIC operation is assumed to take 1 clock cycle.

The second part (b) calculates the correction vector ΔZ. It requires 2 adders for 2–dimenstional tracking, and 3 adders for 3–dimenstional tracking.

The third part (c) of Equation (5.5) may be expanded as follows:

$$\hat{X}(k) = \hat{X}(k|k-1) + K(k)\,\Delta Z$$

$$\begin{bmatrix} x(k) \\ \dot{x}(k) \\ \dot{x}^{\cdot}(k) \\ y(k) \\ \dot{y}(k) \\ \dot{y}^{\cdot}(k) \end{bmatrix} = \begin{bmatrix} x(k|k-1) \\ \dot{x}(k|k-1) \\ \dot{x}^{\cdot}(k|k-1) \\ y(k|k-1) \\ \dot{y}(k|k-1) \\ \dot{y}^{\cdot}(k|k-1) \end{bmatrix} + \begin{bmatrix} k_{xr} & k_{x\theta} \\ k_{\dot{x}r} & k_{\dot{x}\theta} \\ k_{\dot{x}^{\cdot}r} & k_{\dot{x}^{\cdot}\theta} \\ k_{yr} & k_{y\theta} \\ k_{\dot{y}r} & k_{\dot{y}\theta} \\ k_{\dot{y}^{\cdot}r} & k_{\dot{y}^{\cdot}\theta} \end{bmatrix} \begin{bmatrix} \Delta r \\ \Delta\theta \end{bmatrix}$$

$$= \begin{bmatrix} x(k|k-1) \\ \dot{x}(k|k-1) \\ \dot{x}^{\cdot}(k|k-1) \\ y(k|k-1) \\ \dot{y}(k|k-1) \\ \dot{y}^{\cdot}(k|k-1) \end{bmatrix} + \Delta r \begin{bmatrix} k_{xr} \\ k_{\dot{x}r} \\ k_{\dot{x}^{\cdot}r} \\ k_{yr} \\ k_{\dot{y}r} \\ k_{\dot{y}^{\cdot}r} \end{bmatrix} + \Delta\theta \begin{bmatrix} k_{x\theta} \\ k_{\dot{x}\theta} \\ k_{\dot{x}^{\cdot}\theta} \\ k_{y\theta} \\ k_{\dot{y}\theta} \\ k_{\dot{y}^{\cdot}\theta} \end{bmatrix}$$

where ΔZ is defined as follows:

$$\Delta Z = \begin{bmatrix} \Delta r \\ \Delta\theta \end{bmatrix} = \begin{bmatrix} \text{correction factor in range} \\ \text{correction factor in azimuth angle} \end{bmatrix}$$

The expansion shows that the state estimate vector $\hat{X}(k)$ is a sum of the state prediction vector $\hat{X}(k|k-1)$, the first column vector of K(k) scaled by Δr, and the second column vector of K(k) scaled by $\Delta\theta$.

Figure 5.9 shows a broadcast implementation of the part (c) of Equation (5.5) for 2–dimensional tracking. The architecture in Figure 5.9 consists of two columns of six multipliers and one column of six adders with memory. In this architecture, Δr, $\Delta\theta$, and the elements of K(k) from Processor for the coordinate transformation of the Kalman gain are broadcasted to necessary multipliers. Six

multipliers on the left implement the multiplication of the first column of K(k) by the broadcasted Δr, while six multipliers on the right implement the multiplication of the second column of K(k) by $\Delta\theta$. Six adders with memory in the middle add two following vectors which are the results of the multiplications at the two columns of multipliers:

$$\Delta r \begin{bmatrix} k_{xr} \\ k_{\dot{x}r} \\ k_{\dot{x}\cdot r} \\ k_{yr} \\ k_{\dot{y}r} \\ k_{\dot{y}\cdot r} \end{bmatrix} \quad \text{and} \quad \Delta\theta \begin{bmatrix} k_{x\theta} \\ k_{\dot{x}\theta} \\ k_{\dot{x}\cdot\theta} \\ k_{y\theta} \\ k_{\dot{y}\theta} \\ k_{\dot{y}\cdot\theta} \end{bmatrix}$$

to $\hat{X}(k|k-1)$, prestored in memory. The sum of this addition is $\hat{X}(k)$, and it is stored in the memory of adders. The implementation of equation

$$\hat{X}(k) = \hat{X}(k|k-1) + K(k)\ \Delta Z(k)$$

requires 2 clock cycles, since the multiplication of K(k) and $\Delta Z(k)$ requires 1 clock cycle using two sets of mutlipliers, and the addition of $\hat{X}(k|k-1)$ and the product, $K(k)\ \Delta Z(k)$, requires another clock cycle.

The implementation of Equation (5.6), which predicts the state vector

$$\hat{X}(k+1|k) = \phi(k)\hat{X}(k) \tag{5.6}$$

requires a matrix–vector multipication. However, the implementation of Equation (5.6) may be simplified using the property of the band transition matrix $\phi(k)$ in

the same way as in the implementation of Equation (5.3). Figure 5.10 illustrates an implementation of Equation (5.6) for 2–dimensional tracking using the property of the band transition matrix $\phi(k)$. This implementation consists of six processing elements; each processing element alternatively stores an element of the state vector $X(k)$ and that of the predicted state vector $X(k+1|k)$ and performs additions and multiplications. The detailted implementation of Equation (5.6) is as follows:

a) Move the elements of the state vector $\hat{X}(k)$, stored in processing elements, upward.

b) Multiply the element already stored in a processing element by the corresponding element of $\phi(k)$, and multiply the element moved upward in step (a) by another element $\phi(k)$ that corresponds to it.

c) Add the results of the two multiplications in step (b).

The implementation in Figure 5.10 requires only 2 clock cycles on 6 processing elements to perform Equation (5.6). This requirement is very small compared to that of a conventional implementation of a matrix–vector multiplication. The conventional implementation requires an order of n $\{O(n)\}$ clock cycles on a systolic array of two orders of n $\{O(n^2)\}$ processing elements, where n is the dimension of an array. Hence, the use of the property of $\phi(k)$ reduces the requirement of a matrix–vector multiplication by an order of magnitude.

Note that $\phi(k)$ and $\hat{X}(k)$ are prestored in the local memory of each processing element. This can be easily realized, because $\phi(k)$ is constant over time, and $\hat{X}(k)$ was calculated and stored at the previous step for Equation (5.5).

Structures in Figures 5.9 and 5.10 may be combined to form a single structure shown in Figure 5.11. The structure in Figure 5.11 alternatively provides the prestored $\hat{X}(k+1|k)$ for the calculation of $\hat{X}(k)$ in Equation (5.5) at the end of Equation (5.6), and $\hat{X}(k)$ for the calculation of $\hat{X}(k+1|k)$ in Equation (5.6) at the end of Equation (5.5).

Equations (5.5) and (5.6) can also be implemented in a pipeline manner as shown in Figures 5.12 and 5.13, using ([28], [29]). The functionality of a processing element in Figures 5.12 and 5.13 is shown in Figure 5.12 (b). Processing elements in Figure 5.12 generate the elements of $\hat{X}(k)$ in a pipeline manner, as they receive in a skewed manner the Kalman gain matrix K(k) from the top, the predicted state vector $\hat{X}(k|k-1)$ from the right, and Δr and $\Delta\theta$ from the left. Note that skew and delay are introduced in input data streams for synchronization. Similarly, processing elements in Figure 5.13 implement Equation (5.6) in a pipeline manner. Figures (5.12) and (5.13) can be combined to form an architecture in Figure 5.14.

The implementations of Equations (5.5) and (5.6) confirm that the pipeline processing architecture in Figure 5.14 requires a simpler bus connection, fewer processing elements, additional hardware to skew the input data stream, and longer computational time than the broadcast processing architecture in Figure 5.11. In other words, each architecture has advantages and disadvantages over the other. Hence, the selection of an appropriate architecture should be based on design criteria. However, the comparison of the structures in Figure 5.11 and 5.14 shows that the broadcast structure in Figure 5.11 is preferrable in this particular case, because it does not require significantly more hardware than the other structure, and because it is faster and easier to control than the other.

5.1.4 Coordinate Rotation Digital Computer

The matrix $F_1(k)$ in Equation (5.4) requires the evaluation of trigonometric function, whereas the function $h(X(k|k-1))$ in Equation (5.5) requires a coordinate transformation from Cartesian to polar coordinates. The evaluation of trigonometric functions and coordinate transformations are generally difficult to implement. However, the coordinate rotation digital computer (CORDIC), proposed by Volder [49] and discussed in Section 4.3, is suitable for a 2–dimensional coordinate transformation and the evaluation of trigometric functions, since it requires only shifting, adding, subtracting, and the recall of prestored constants. They are shown in Figures 5.15a and 5.15b respectively.

We have found that Volder's scheme can be extended to a 3–dimensional coordinate transformation by placing two CORDIC's and 2 scalers in series, as shown in Figure 5.15c. Hence, the coordinate transformation, $h(X(k|k-1))$, for 3–dimensional tracking can be implemented using an architecture in Figure 5.15c.

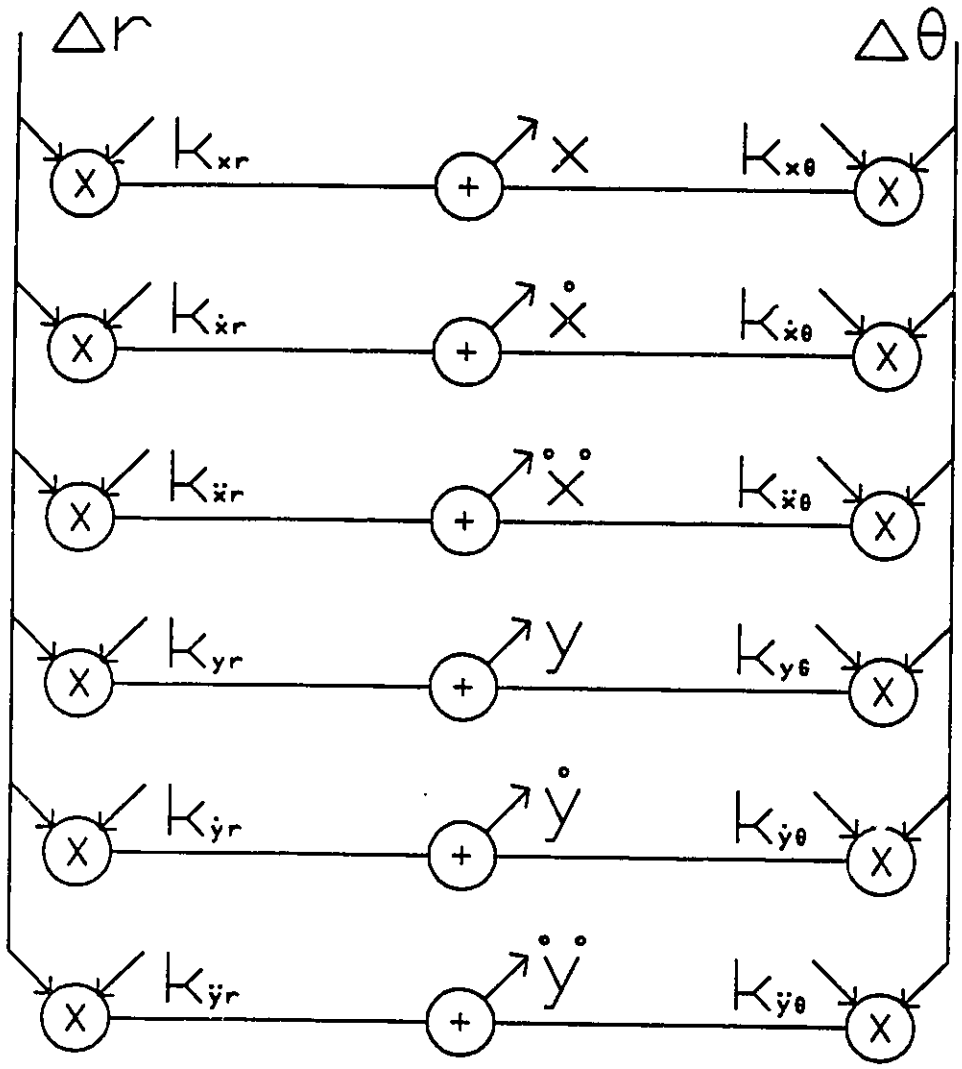


Figure 5.9 Estimation of the state X(k) in a broadcasting manner

Figure 5.10 Prediction of the state X(k+1) in a broadcasting manner

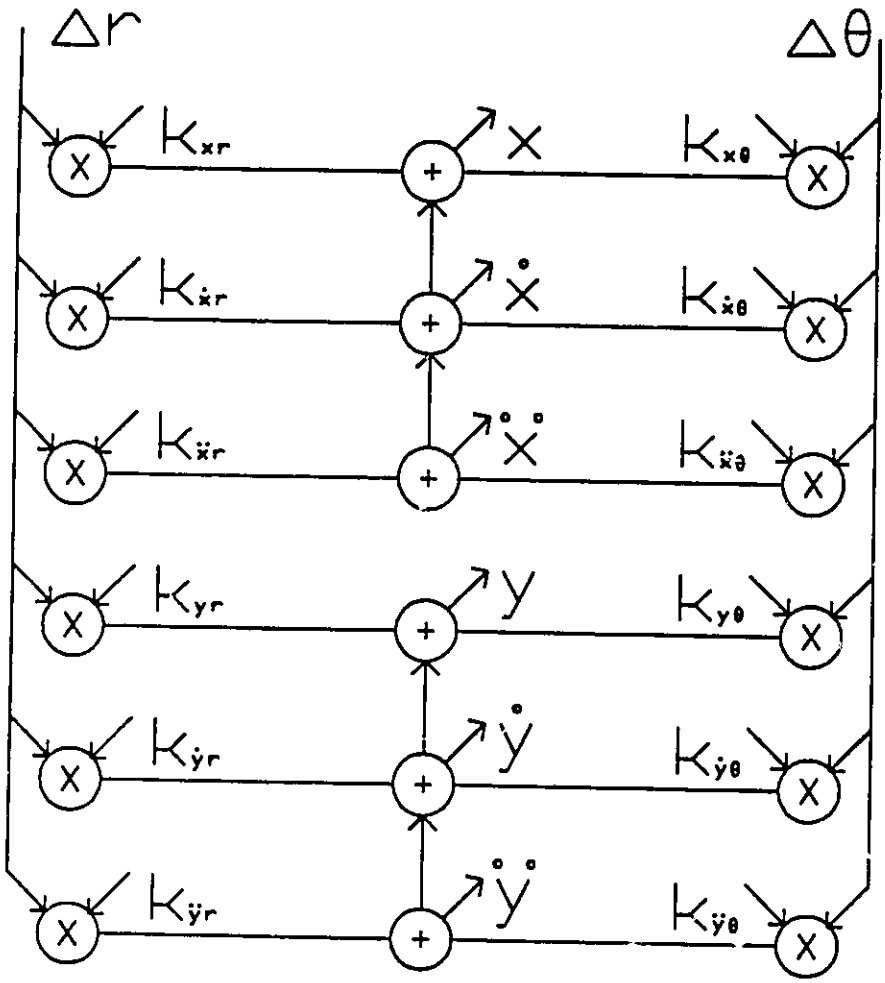


Figure 5.11 Estimation of the state X(k) and prediction of the state X(k+1) in a broadcasting manner

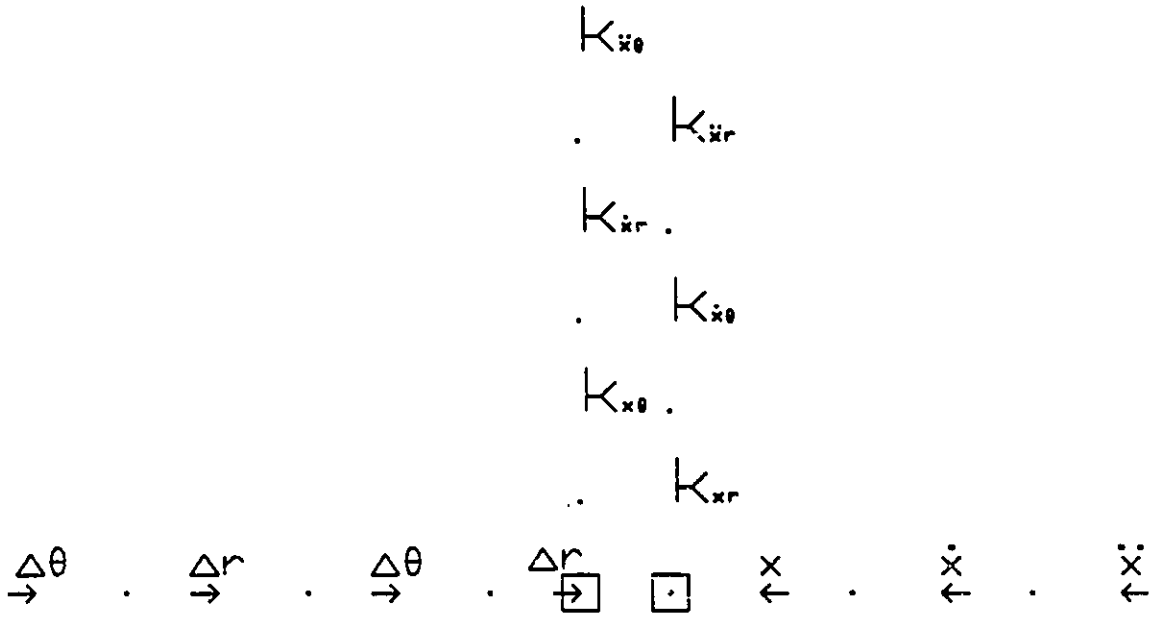


Figure 5.12 Estimation of the state X(k) in a pipelining manner

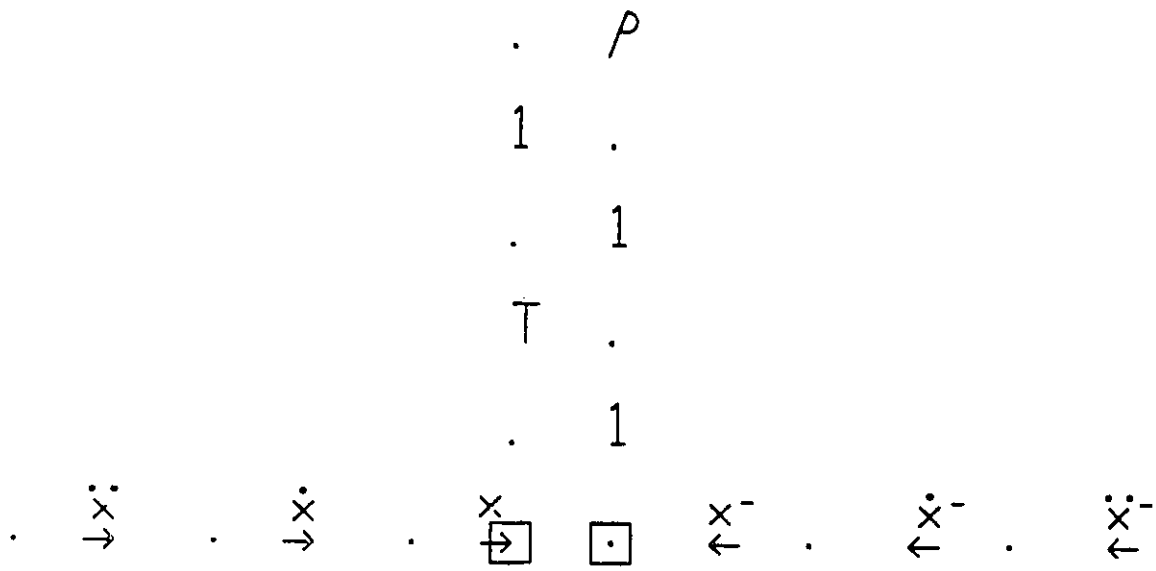


Figure 5.13 Prediction of the state X(k+1) in a pipelining manner

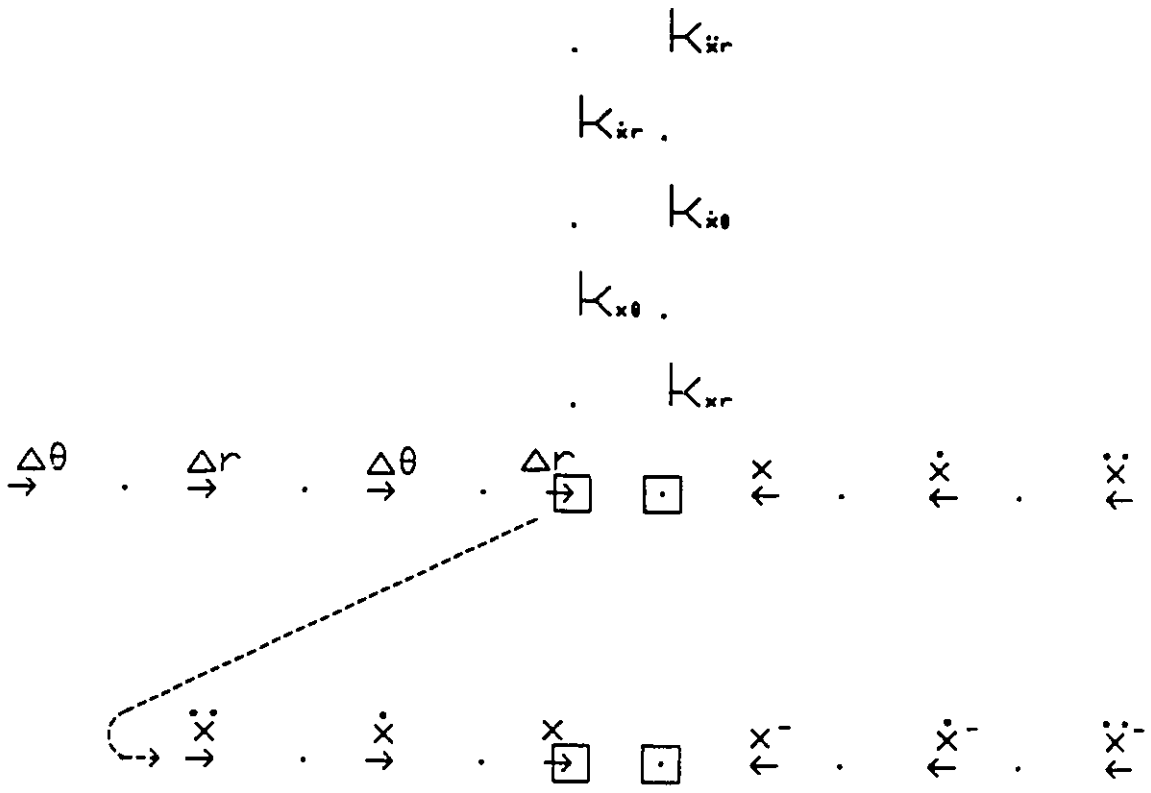


Figure 5.14 Combination of Figures 5.12 and 5.13

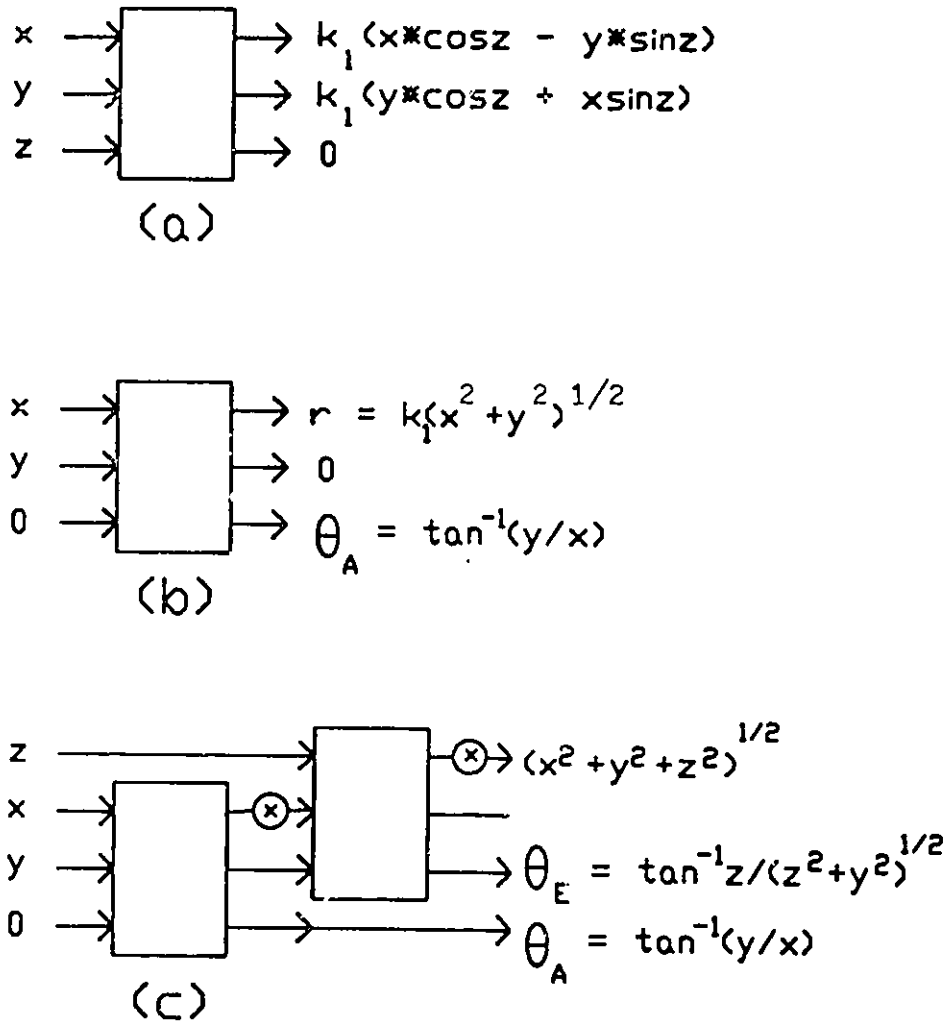


Figure 5.15 Coordinate digital computer (CORDIC)
a) CORDIC for the evaluation of trigonometric functions
b) CORDIC for the 2–dimensional coordinate transformation
c) Series of two CORDIC's for the 3–dimensinonal coordinate transformation

5.2 The Benefits of the Decoupling Technique

In the previous section, we have developed an architecture for the ECKF, utilizing the decoupling technique and special properties in the tracking KF. In this section, we examine how the computational reduction by the decoupling technique, discussed in Chapter4, is translated into the reduction in hardware and computational time requirements.

We consider a matrix–matrix multiplication which is predominant in the time and measurement updates of the state estimate error covariance matrix P(k) in the KF. The multiplication of two n–by–n matrices requires n^3 multiplications. In Section 4.3, we have found that in the decoupled KF a n–by–n matrix–matrix multiplication is separated into m (n/m)–by–(n/m) matrix–matrix multiplications, and that m (n/m)–by–(n/m) matrix–matrix multiplications require m^2 times less number of multiplications than a n–by–n matrix–matrix multiplication.

If we assume that the systolic array, described in Section 3.2.1, is used for a matrix multiplication, a n–by–n matrix–matrix multiplication requires 3n–2 clock cycles on a n–by–n systolic array, whereas m (n/m)–by–(n/m) matrix–matrix multiplications require (3n/m – 2) clock cycles on m (n/m)–by–(n/m) systolic arrays. A n–by–n systolic array has n^2 processing elements, whereas m (n/m)–by–(n/m) systolic arrays have n^2/m processing elements. Hence, m small matrix–matrix multiplications require m times less number of processing elements and approximately m times less number of clock cycles than one large matix–matrix multiplication. Note that the product of the reduction ratio for the number of required processing elements and that for the computational time is m^2. It is equal to the reduction ratio for the required number of multiplication, found in Section 4.3. This means that the computational reduction of m^2 is translated into the reduction of processing elements and clock cycles by a factor

of m.

Since a matrix-matrix multiplication is predominant in the time and measurement updates of P(k) in the KF, we can apply the finding of this section to the time and measurement updates of P(k). That is, the decoupling technique reduces the hardware and computational time requirements for the time and meausrement updates by a factor of m.

5.3 Requirements of Hardware and Computational Time

In this section, we summarize the requirements of hardware and computational time for the parallel implementation of the simplified extended KF, using the decoupling technique and special properties in the tracking KF. We have so far presented the hardware requirements for each processor separately, as we discussed the implementation of each KF equation on the proposed architecture.

Table 5.1 (a) presents the hardware requirements for each processor and the total requirements in terms of m and n, where m denotes the dimension of a tracking system, and n denotes the number of elements in the state vector X(k). Since this table expresses the hardware requirements in terms of m and n, it can be used to find the hardware requirements for both 2- and 3-dimensional tracking systems. The requirements in Table 5.1 (a) are based on the assumption that Equations (5.4), (5.5), and (5.6) are implemented using a broadcast processing structure. The total hardware requirements are $n^2/m + m$ processing elements, $2nm + 2n + 1$ or $2nm + 2n$ multipliers ($2nm + 2n + 1$ is for 3-dimensional tracking; $2nm + 2n$ is for 2-dimensional tracking), $n + m$ adders, $2m - 2$ CORDIC's, and $n^2/m - n$ delay elements.

To obtain an intuitive understanding on the hardware requirements, the

hardware requirements for 3–dimensional tracking system with 9 state elements are presented in Table 5.1 (b). Processing elements in Tables 5.1 (a) and (b) are much simpler than a boundary processing element in the systolic array for the QR decomposition described in Section 3.2.2.

The number of clock cycles needed to implement each KF equation is summarized in Table 5.2 for 3–dimensional tracking. Here we assume that Equations (5.4), (5.5) and (5.6) are implemented using a broadcast processing structure, and that multiplication, division, CORDIC calculation, and addition are all performed within 1 cycle

In Table 5.2, Equation (5.4) is divided into two parts and Equation (5.5) is divided into three parts. Equation (5.4), which rotates the Kalman gain $K_o(k)$ for the LOS frame by the rotation matrix

$$\begin{bmatrix} F_1 & 0 & 0 \\ 0 & F_1 & 0 \\ 0 & 0 & F_1 \end{bmatrix}$$

where F_1 is defined in terms of $\sin\theta$ and $\cos\theta$, may be performed in two steps:

a) The calculation of $\sin\theta$ and $\cos\theta$ in $F_1(k)$ immediately after $\theta(k|k-1)$ is computed.

b) The rotation of the Kalman gain $K_o(k)$ by the rotation matrix $F_1(k)$.

The precalculation of $\sin\theta$ and $\cos\theta$ allows the rotation of the Kalman gain $K_o(k)$ to be performed immediately after $K_o(k)$ is available. It eliminates having to wait for the calculation of $\sin\theta$ and $\cos\theta$, after $K_o(k)$ is available.

Similarly, Equation (5.5) may be separated into the following three parts:

a) $Z(k|k-1) = h(\hat{X}(k|k-1))$

b) $\Delta Z = Z(k)-Z(k|k-1)$

c) $\hat{X}(k) = \hat{X}(k|k-1) + K(k)\ \Delta Z.$

The transformation of $\hat{X}(k|k-1)$ in Step (a) and the calculation of ΔZ in Step (b) are precomputed before K(k) is available.

In addition to the concurrent update of the state estimate error covariance matrices P(k)'s for each axis, some of the decoupled ECKF equations can be performed simultaneously. For example, after K_o is calculated, the processor for the LOS coordinates can continue computing $P_o(k)$ and $P_o(k+1|k)$ while the processor for the reference Cartesian coordinates calculates $\hat{X}(k)$ and $\hat{X}(k+1|k)$. Figure 5.16 shows how the equations can be performed in parallel according to their interdependence, and it shows how many clock cycles are needed to complete one iteration. We find that 16 clock cycles are needed for 3-dimensional tracking.

Figure 5.16 shows that Processor 2 and Processor 3 are left idle after computing Equations (5.4a) and (5.5a) respectively. Hence, if the implementation of Equations (5.4a) and (5.5a) takes longer than 2 clock cycles each, actually upto 5 clock cycles, the total computational time requirements would not increase. Equation (5.5a) is a 3-dimensional coordinates transformation which requires 2 CORDIC operations and 1 scaling operation, and Equation (5.4a) evaluates the rotation matrix $F_1(k)$ using 1 CORDIC operation and 1 multiplication. As a result, the increase in the required time for the implementation of a CORDIC operation from an assumed 1 cycle to 2 cycles does not affect the total

computational time required for an iteration of the ECKF. This means that the CORDIC operation is no longer needed to be performed within one clock cycle, as assumed before. It is now required to be implemented within 2 clock cycles.

Table 5.1 Hardware requirements for the simplified ECKF
(a) for m-dimensional tracking with n state elements
(b) for 3-dimensional tracking with 9 state elements

Table 5.1 (a)

Equation number	PE.	Mult.	Add.	CORDIC.	Delay.
5.1, 5.2, 5.3	$\frac{n^2}{m}+m$	2n	0	0	$\frac{n^2}{m}-n$
5.4*	0	nm+4	0	m-1	0
5.5, 5.6*	n	nm-3	m	m-1	0
Total*	$\frac{n^2}{m}+n+m$	2nm+2n+1	m	2m-2	$\frac{n^2}{m}-n$

PE. = Processing element
Mult. = Multiplier
Add. = Adder
Delay. = Delay element

* For 2-dimensional tracking, Equation (5.4), and Equations (5.5) and (5.6) both require nm multipliers, and the total number of required multipliers is 2nm+2n.

Table 5.1 (b)

Equation number	PE.	Mult.	Add.	CORDIC.	Delay.
5.1, 5.2, 5.3	30	18	0	0	18
5.4	0	31	0	2	0
5.5, 5.6	9	24	3	2	0
Total	39	73	3	4	18

Table 5.2 Computational time requirements of the parallel implementation of the simplified ESRCF for 3–dimensional tracking with 9 state elements

Table 5.2

Equation number	Time
5.1	5
5.2	6
5.3	5
5.4a	2
5.4b	1
5.5a	2
5.5b	1
5.5c	2
5.6	2

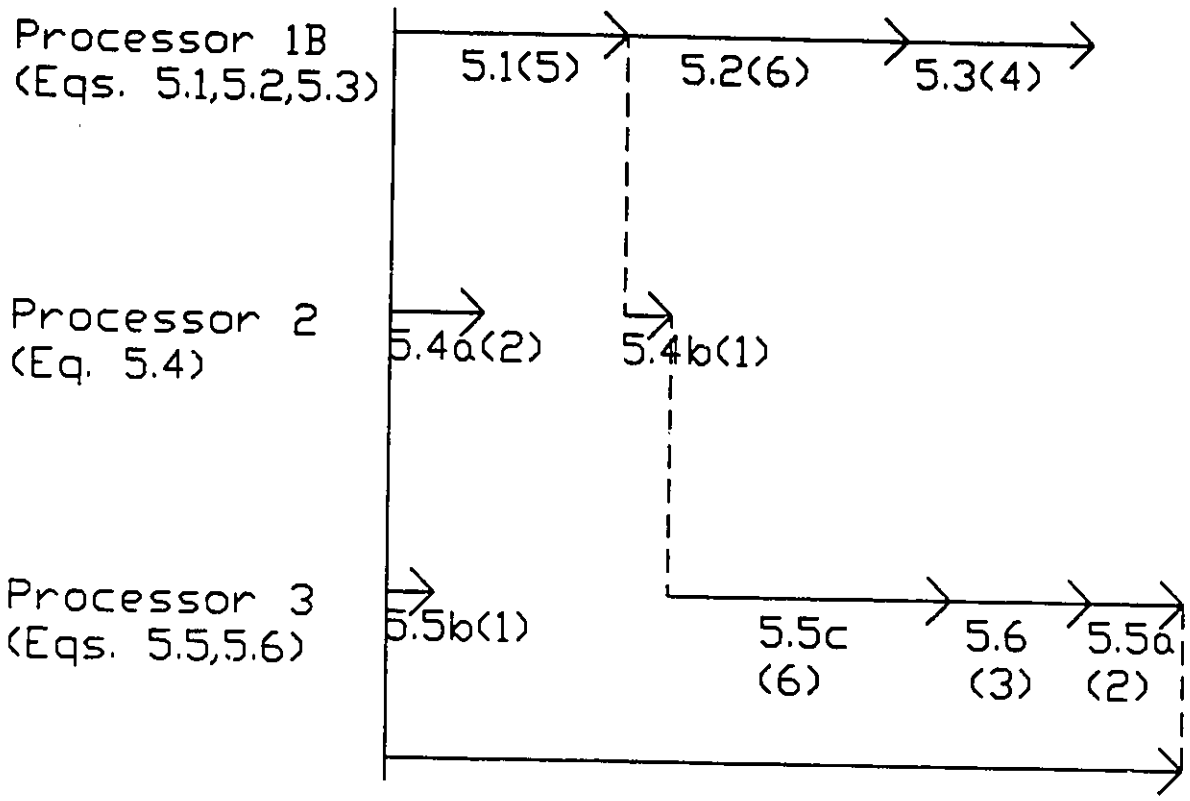


Figure 5.16 Parallel computation of the simplified ECKF (numbers in brackets indicates the number of clock cycles required for eacn step)

5.4 Comparisons

In Section 5.2, we have shown that the hardware and computational time requirements of a matrix–matrix multiplication, which is predominant in the KF equations, may be reduced through the use of the decoupling technique by a factor of m, where m is the tracking dimension. In addition to the decoupling technique, we have utilized special properties in the tracking KF to simplify the implementation of the KF.

In this section, we study how effective the use of the decoupling technique and special properties is in reducing hardware and computational time requirements by comparing the proposed architecture in this chapter with others. However, the proposed architecture is difficult to compare with other architectures in the literature. The main reason is that the proposed architecture in this chatper is the first parallel implementation of the extended KF, while other parallel architectures in the literature, including the ones reviewed in Chapter 3, implement the standard KF.

Although the extended KF and the standard KF are different, they use the same equations (5.1), (5.2), and (5.3) for the update of the state estimate error covariance matrix P(k) and for the calculation of the Kalman gain matrix K(k). The comparison of architectures for thesee equations can give an idea how effective the use of the decoupling technique and special properties is. Equations (5.1), (5.2), and (5.3) require more computation than the remaining equations.

We compare the proposed implementation of Equations (5.1), (5.2), and (5..3) with the architecture for the standard covariance KF, developed by Papadourakis and Taylor, and described in Section 3.3.1, since the proposed architecture in this chapter is for the extended covariance KF, not for the extended SRCF. Papdourakis and Taylor have simplified an implementation of the

standard covariance KF by reformulating the KF equations. Their implementation consists of n–by–n processing elements and requires more than 16n clock cycles to update P(K) and to calculate K(k), where n denotes the number of elements in X(k).

The proposed architecture in this chapter requires $\frac{n}{m}^2 + m$ processing elements and 16 clock cycles, where m denotes the dimension of tracking. A comparison of the two architectures confirms that the decoupling technique reduces the required number of processing elements by a factor of approximately m, as described in Section 5.2. The use of special properties does not affect the required number of processing elements, but it helps to reduce computational time. The application of special properties converts matrix–vector multiplications in the KF equations into vector–vector multiplications which require much less number of clock cycles to implement. The combined use of the decoupling technique and special properties is found to reduce the computational time requirements by a factor of greater than n.

A further comparison of the two architectures shows that the data flow in the proposed architecture is much simpler than that in the Papadourakis–Taylor architecture. This leads to simpler control logic requirements for the architecture developed in this chpater.

In conclusion, the parallel implementation of the extended covariance KF in this chapter is superior to the implementation by Padourakis and Taylor. This superiority clearly indicates the benefits of utilizing the decouping technique and special properties in the tracking KF.

5.5 Summary

In this chapter, we have developed the first parallel implementation of the

extended covariance Kalman filter for tracking applications. We have utilized the concepts of pipelining and parallel processing to minimize computational time and hardware requirements, bearing in mind the desirable properties of parallel architectures such as regularity, modularity, and local interconnection.

In designing the architectures of ECKF, we have made extensive use of the decoupling technique and special properties in matrices encountered in the tracking KF. The ECKF, simplified by the use of the decoupling technique and special properties in the tracking KF, are well suited for parallel architectures for the following reasons:

a) Computationally demanding time and measurement updates of the state estimate error covariance matrix P(k) for the ECKF are performed in parallel for each axis.
b) Computational requirements of the ECKF are reduced
c) The transformation of the Kalman gain matrix from the LOS to reference Cartesian coordinates, which compensates for the decoupling technique, is easy to implement, as shown in Section 5.1.2.

The reduction in computational requirements is achieved by the separation of a coupled P(k) into a number of smaller decoupled matrices, the elimination of an matrix inversion, and the simple evaluation of the measurement matrix $H_o(k)$ for the decoupled ECKF at each filtering time instant.

The sparse nature of the measurement equation H(k) reduces matrix–vector multiplications in Equations (5.1) and (5.2) to vector–vector multiplications, which are easier to implement than matrix–matrix multiplications. Furthermore, the use of the sparse, band nature of transition matrix $\phi(k)$ makes it

possible to multiply two n–by–n matrices in Equation (5.3), which normally takes an order of n steps on an architecure with n^2 multipliers and n^2–n delay elements, in an order of 1 steps. The letter n denotes the number of elements in the state vector X(k). In the proposed implementation, this simplified multiplication does not require any additional hardware except for a bidirectional internal bus connecting processing elements, as shown in Figure 5.5.

We have developed both pipeline and broadcast processing architectures for Equations (5.4), (5.5), and (5.6), utilizing the sparse, band nature of transition matrix $\phi(k)$. The pipeline processing architecture where the data are passed through processing elements one at a time is found to require a longer computational time, more delay elements, and fewer processing elements than the broadcast processing architecture where the data are broadcasted to all the necessary processing elements. The selection of an implementation should be based on design criteria. However, the broadcast processing implementation seems more suitable to achieve fast processing at a small cost of additional hardware.

The proposed implementation of Equation (5.6) in a broadcast processing manner with the exploitation of the special properties of $\phi(k)$ requires an order of 1 clock cycles on an order of n processing elements. However, if Equation (5.6) is implemented without using the properties of $\phi(k)$, the implementation of Equation (5.6) would require an order of n clock cycles on two orders of n processing elements. The use of the properties of $\phi(k)$ reduces the required number of processing elements and that of clock cycles by an order of magnitude.

The total hardware requirements for the implementation of the complete extended covariance KF with Equations (5.4), (5.5), and (5.6) being implemented in a broadcast processing manner are $n^2/m+n+m$ processing elements, 2nm+2n+1

multipliers, m adders, 2m–2 CORDIC's, $n^2/m-m$ delay elements; specifically, for 3–dimensional tracking, the hardware requirements are 39 processing elements, 73 multipliers, 3 adders, 4 CORDIC's, and 18 delay elements. The computational time requirement for each iteration is 16 clock cycles.

We have shown that the decoupling technique reduces the hardware and computational time requirements for time and measurement updates of P(k) by a factor of approximately m. A comparison of the architecture developed in this chapter with the Papadourakis–Taylor architecture shows that the combined use of the decoupling technique and special properties in the tracking KF reduces the hardware and computational time requirements for the time and measurement updates and calculation of the Kalman gain by factors of m and greater than n, respectively.

CHAPTER 6

PARALLEL IMPLEMENTATION OF THE EXTENDED SRCF

The computation of the Kalman filter equations with finite word-length arithmetic may lead to numerical problems due to cumulative roundoff or truncation errors. As described in Chapter 2, the KF has been reformulated in terms of the square root of the state estimate error covariance matrix to improve the numerical properties of the KF [4] [25]. This reformulated KF is referred to as the square root covariance filter (SRCF).

The SRCF has twice the effective precision of the standard KF, but it is more computationally demanding than the standard KF. When the improved numerical properties are needed and additional computational requirements are acceptable, the SRCF has been chosen over the standard covariance KF in various applications. Particularly, for a tracking application, the extended SRCF has been chosen over the extended covariance KF. However, as in the extended covariance KF, the higher computational demand of the extended SRCF has limited its use to some extent.

In Chapter 5, we have developed a parallel implementation of the extended covariance KF with the consideration of desirable characteristics for parallel architectures such as high degree of pipelining and parallelism. In this chapter, we develop a parallel architecture for the extended SRCF for tracking applications. There is more pressing need for parallel implementation of the

extended SRCF than that of the extended covariance KF, since the extended SRCF requires more computation than the extended covariance KF.

As in Chapter 5, we develop a parallel architecture for the simplified extended SRCF through the use of the decoupling technique and special properties of matrices in the tracking KF. This simplified extended SRCF requires much less computation than the standard extended SRCF. Furthermore, there is more inherent parallelism in the simplified extended SRCF than the standard extended SRCF.

The decoupled extended covariance Kalman filter and the decoupled extended SRCF differ in that the former updates the error covariance matrix $P_o(k)$, and the latter updates the square root of $P_o(k)$. However, they use the same equations to rotate the Kalman gain $K_o(k)$ from the LOS frame to the reference Cartesian frame, and to make a state estimation and prediction. As a result, in developing an architecture for the decoupled extended SRCF, we use processors developed in Chapter 5 for the equations common to both the decoupled extended CKF and the decoupled extended SRCF, and we develop a new processor for uncommon equations.

In Section 6.1, we describe the proposed architecture. In section 6.2, we study the effects of the decoupling technique on the parallel implementation of the QR decomposition to gain a better understanding on the benefits of the decoupling technique in a parallel architecture. In Section 6.3, we summarize the hardware and computational time requirements of the proposed architecture. In Section 6.4, we compare the proposed architecture with others. Finally, in Section 6.5, we present the summary of this chpater.

6.1 Architecture Description

Like the extended CKF, the extended SRCF equations (4.33–4.38) split into three parts in Section 4.4 by the decoupling technique. Correspondingly, the architecture for the decoupled extended SRCF equations may be split into 3 parts, as follows:

a) The processor for the LOS frame computes $S_o(k|k-1)$, $S_o(k)$ and $K_o(k)$, as shown by

$$\begin{bmatrix} S_o^T(k|k-1) \\ 0 \end{bmatrix} = Q_1 \begin{bmatrix} S_o^T(k-1) & \phi_o^T(k-1) \\ & U_o^T(k-1) \end{bmatrix} \tag{6.1}$$

$$\begin{bmatrix} F_o(k) & G_o(k) \\ 0 & S_o^T(k) \end{bmatrix} = Q_1 \begin{bmatrix} V^T(k) & 0 \\ S_o^T(k|k-1)H_o^T(k) & S_o^T(k|k-1) \end{bmatrix} \tag{6.2}$$

$$K_o(k) = G_o^T(k) \,/\, F_o^T(k) \tag{6.3}$$

b) The processor for the Coordinate transformation computes K(k) from $K_o(k)$ by using the Jacobian transformation:

$$K(k) = \begin{bmatrix} F_1 & 0 & 0 \\ 0 & F_1 & 0 \\ 0 & 0 & F_1 \end{bmatrix} K_o(k) \tag{6.4}$$

c) The processor for the reference Cartesian coordinate frame computes $\hat{X}(k)$ and $\hat{X}(k+1|k)$, as shown by

$$\hat{X}(k) = \hat{X}(k|k-1) + K(k)\ (\ Z(k) - h(\hat{X}(k|k-1))\ ) \tag{6.5}$$

$$\hat{X}(k+1|k) = \phi(k)\ \hat{X}(k) \tag{6.6}$$

The processor for the LOS frame, that for the Coordinate transformation of $K_o(k)$, and that for the reference Cartesian coordinates will be hereafter referred to as Processor 1, Processor 2, and Processor 3, respectively.

A block diagram of the proposed architecture is shown in Figure 6.1. It consists of 3 processors whose functions were described above, and it is very similar to the block diagram of Figure 5.1. However, differences will appear in implementing Processor 1, due to functional differences between Processor 1 for the ECKF and that for the ESRCF.

Processor 1 implements Equations (6.1) and (6.2) which require orthogonal triangularizations, and Equation (6.3) which requires a division of a vector by a scalar number. As a result, Processor 1 needs to perform two orthogonal upper triangularizations and one division of a vector by a scalar number. As discussed in Section 3.2.2, the orthogonal upper triangularization which is referred to as the QR decomposition may be performed by using various algorithms. However, the Givens rotation using the systolic array described in Section 3.2.2 is preferrable for the QR decomposition because of its simplicity and regularity,

Two QR decompositions can be implemented using one or two QR systolic arrays. However, sharing one systolic array for two QR decompositions is difficult, because the delay elements and multipliers necessary for Equation (6.1) have to be somehow bypassed in calculating Equation (6.2).

A triangular systolic array for the QR decomposition, described in Section 3.2.2, assumes that an input matrix to be orthogonally upper triangularized enters

the systolic array from the top row to the bottom row. We find that this systolic array can be easily modified to deal with an input matrix entering the systolic array in reverse order, from the bottom row to the top row. Boundary and internal cells of this modified systolic array have slightly different functionalities than those in Section 3.2.2. The functionalities of these boundary and internal cells are described in Figure 6.2.

Figures 6.3 and 6.4 depict a block diagram of Processor 1 using two QR systolic arrays represented by triangles; systolic arrays in Figure 6.3 assume that an input matrix enters the systolic array botton row first, whereas systolic arrays in Figure 6.4 assume the opposite direction of data flow of an input matrix. Since in Figure 6.3 the data enter the systolic array from the top and leave from the bottom, the bottom of one systolic array is connected to the top of the other, and the bottom of the latter systolic array is connected to the top of the former systolic array; the two systolic arrays in Figure 6.3 are thus connected by two unidirectional buses. In contrast to data flow in Figure 6.3, the data in Figure 6.4 enter the systolic array from the top and leave from the top again, so that only one bidirectional bus is required to connect the tops of the two systolic arrays.

An architecture, based on a unidirectional bus, requires longer interconnection than an architecture with a bidrectional bus, because the former needs two buses and the latter requires only one unidirectional bus. However, the unidirectional bus is usually easier to control than the bidirectional bus. It should be noted that an architecture, based on a unidirectional bus, requires a unidirectional interconnection among processing elements, and vice versa for an architecture based on a bidirectional bus. We will call a processor with a bidirectional bus Processor 1A, and a processor with a unidirectional bus Processor 1B hereafter.

We describe Processor 1A and Processor 1B in detail in Section 6.1.1 and 6.1.2 respcetively. We then describe the rest of the implementation for the decoupled ESRCF in Section 6.1.3.

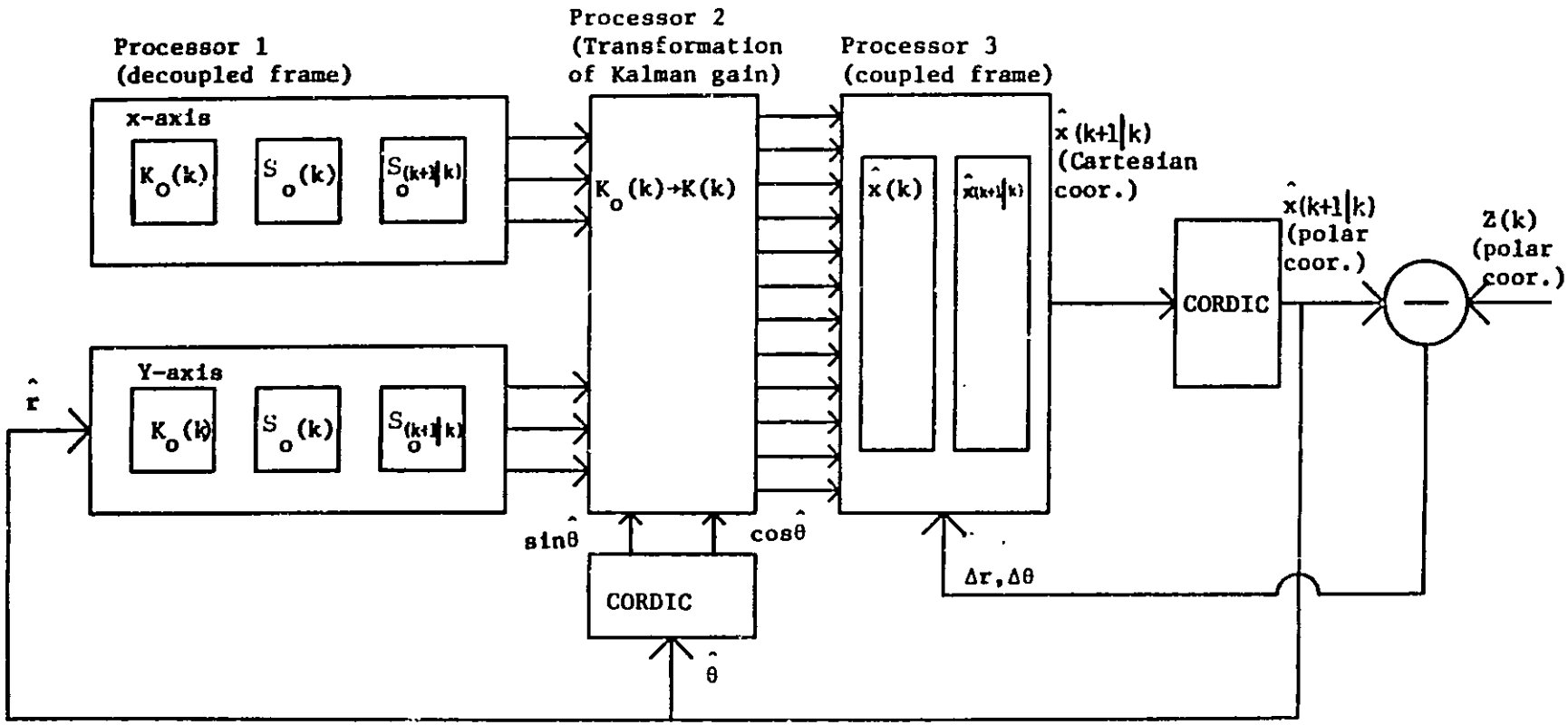
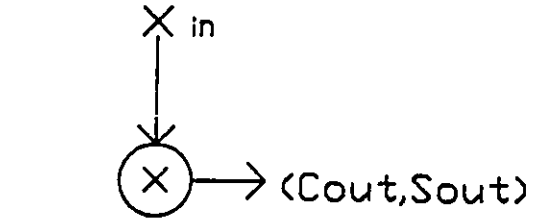


Figure 6.1 Block diagram of the ECKF



if x = 0 then

$c = 1$

$s = 0$

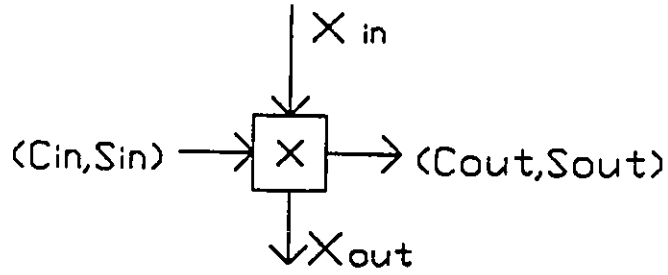
else

$$c_{out} = x_{in} / \sqrt{x^2 + x_{in}^2}$$

$$s_{out} = x / \sqrt{x^2 + x_{in}^2}$$

$$x = \sqrt{x^2 + x_{in}^2}$$

(a)



$$X_{out} = -S_{in} X + C_{in} X_{in}$$

$$X = C_{in} X + S_{in} X_{in}$$

(b)

Figure 6.2 Function of systolic array cells
a) boundary cell
b) internal cell

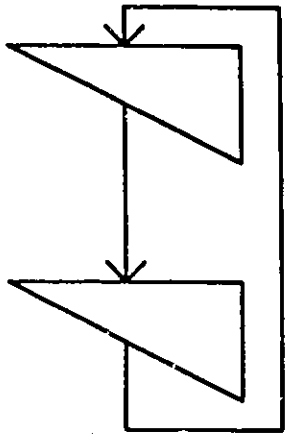


Figure 6.3 Block diagram of the processor for the LOS frame (An input matrix enters the systolic array bottom row first)

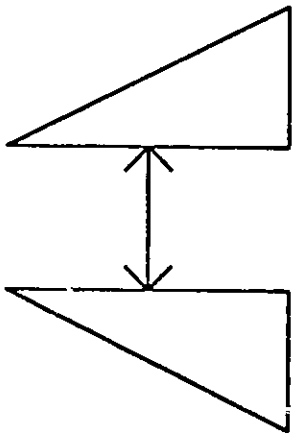


Figure 6.4 Block diagram of the processor for the LOS frame (An input matrix enters the systolic array top row first)

6.1.1 Processor 1A for the LOS frame

Processor 1A updates the square root $S_o(k)$ of the state estimate error covariance matrix $P_o(k)$ and calculates the Kalman gain $K_o(k)$ for the LOS frame by implementing Equations (6.1), (6.2), and (6.3). As shown in Figure 6.3, Processor 1A consists of two QR systolic arrays, where two systolic arrays are designed such that an input matrix enters bottom row first. We use a systolic array on the top for Equation (6.2), and one at the bottom for Equation (6.1) in Figure 6.3.

Due to similarities among the processing for the x–, y–, z–axes, the implementation for only the x–axis is described. We first consider the implementation of Equation (6.2), which is a measurement update equation of $S_o(k|k-1)$:

$$\begin{bmatrix} F_o(k) & G_o(k) \\ 0 & S_o^T(k) \end{bmatrix} = Q_1 \begin{bmatrix} V_o^T(k) & 0 \\ S_o^T(k|k-1)H_o^T(k) & S_o^T(k|k-1) \end{bmatrix} \tag{6.2}$$

It is expressed in detail for the x–axis below:

$$\begin{bmatrix} f_{11} & g_{11} & g_{12} & g_{13} \\ 0 & s_{11} & s_{21} & s_{31} \\ 0 & 0 & s_{22} & s_{32} \\ 0 & 0 & 0 & s_{33} \end{bmatrix} = Q_1 \begin{bmatrix} \sigma_r & 0 & 0 & 0 \\ \bar{s_{11}} & \bar{s_{11}} & \bar{s_{21}} & \bar{s_{31}} \\ 0 & 0 & \bar{s_{22}} & \bar{s_{32}} \\ 0 & 0 & 0 & \bar{s_{33}} \end{bmatrix}$$

Figure 6.5 shows an implementation of Equation (6.2) for the x–axis. This implementation is based on a QR systolic array with 10 processing elements. Figure 6.5 also shows the flow of an input data matrix; the input matrix enters

the systolic array bottom row first in a skewed manner. Note that in the input data stream nonzero elements are preceded by a number of zero elements, corresponding to the zero elements at the bottom left side of the input matrix. We may eliminate the first two rows of zeroes in the input data stream, if we ensure that the elements $\bar{s}_{22}$, and $\bar{s}_{33}$ would be treated as if the two rows of zeroes were not eliminated.

In the QR decomposition, a boundary cell in the systolic array, used in Processor 1A, generates 0 and 1 for $\cos\theta$ and $\sin\theta$ respectively and passes them to the next cell on the righthand side, when it receives 0 as input. Hence, the first two rows of zeroes can be eliminated, if the cells receiving $\bar{s}_{22}$ or $\bar{s}_{33}$ from the top somehow receive 0 and 1 from the left, when they recieve $\bar{s}_{22}$ and $\bar{s}_{33}$. This can be achieved by making internal cells generate 0 and 1 for $\cos\theta$ and $\sin\theta$, respectively, upon "reset" which takes place right before the input matrix enters the systolic array.

If we assume that at the end of the calculation of Equation (6.1) the matrix $S_o(k|k-1)$ is stored in the systolic array for Equation (6.1), as shown in Figure 6.6, then the data flow, shown in Figure 6.5, can be realized by connecting two systolic arrays for Equations (6.1) and (6.2), as shown in Figure 6.7. Note that in Figure 6.7 three delay elements, marked by dotted squares, are added before the systolic array for Equation (6.2). These delay elements are employed to feed the element $\bar{s}_{11}$ to the systolic array before the row (σ_r, $\bar{s}_{11}$, $\bar{s}_{22}$, $\bar{s}_{33}$) is fed into the systolic array. A multiplexer is also added to feed σ_r into a processing element at the top left corner.

Since the result of Equation (6.2) is stored in the systolic array as in Figure 6.8, we find that Equation (6.3):

$$K_o(k) = G_o^T(k)/F_o^T(k),$$

can be performed by reading out $G_o^T(k)$ and $F_o^T(k)$ from the systolic array and dividing $G_o^T(k)$ by $F_o^T(k)$. In other words, $K_o(k)$ is calculated by reading out the top row of the systolic array and dividing three right hand side elements of the row read by the left most element. This process which requires 3 dividers is shown in Figure 6.9.

We now consider the implementation of Equation (6.1):

$$\begin{bmatrix} S_o^T(k+1|k) \\ 0 \end{bmatrix} = Q_1 \begin{bmatrix} S_o^T(k)\,\phi^T(k) \\ U_o^T(k) \end{bmatrix} \tag{6.1}$$

It consists of the matrix–matrix multiplication of $S_o^T(k)$ and $\phi_o^T(k)$, followed by the QR decomposition. A matrix–matrix multiplication can be performed in various ways. We can use the band matrix property, as in Section 5.1.1. The multiplication of $S_o^T(k)$ and $\phi^T(k)$ is shown below in detail:

$$S_o^T(k)\phi^T(k) = \begin{bmatrix} s_{11} & s_{21} & s_{31} \\ 0 & s_{22} & s_{32} \\ 0 & 0 & s_{33} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ T & 1 & 0 \\ 0 & 1 & \rho \end{bmatrix}$$

$$= \begin{bmatrix} s_{11} + Ts_{21} & s_{21} + s_{31} & \rho s_{31} \\ Ts_{22} & s_{22} + s_{32} & \rho s_{32} \\ 0 & s_{33} & \rho s_{33} \end{bmatrix}$$

$$= \begin{bmatrix} b_{11} & b_{21} & b_{31} \\ b_{12} & b_{22} & b_{32} \\ b_{13} & b_{23} & b_{33} \end{bmatrix}$$

A column of the output matrix is a combination of two columns of $S_o^T(k)$. Hence, $S_o^T(k)$ which is stored in the systolic array at the end of Equation (6.2) can be multiplied by $\phi^T(k)$, as shown in Figure 6.10 (a). In Figure 6.10 (a), the QR systolic array is connected to three delay elements represented by dotted squares, and three arithmetic units represented by circles with a cross inside. The delay elements synchronize the elements of $S_o^T(k)$ for a matrix–matrix multiplication, and the multipliers perform a matrix–matrix multiplication. The functionality of an arithmetic unit is presented in Figure 6.10 (b). This implementation requires an order of n less number of clock cycles and processing elements than the conventional implementation for a matrix–matrix multiplication, as shown in Section 5.1.1. The latter n denotes the number of state elements in the state vector $\hat{X}(k)$.

After the multiplication of $S_o^T(k)$ and $\phi^T(k)$, the QR decomposition is performed on a matrix shown in detail below:

$$\begin{bmatrix} S_o^T(k) & \phi^T(k) \\ & U_o^T(k) \end{bmatrix} = \begin{bmatrix} b_{11} & b_{21} & b_{31} \\ b_{12} & b_{22} & b_{32} \\ b_{13} & b_{23} & b_{33} \\ u_{11} & u_{21} & u_{31} \\ 0 & u_{22} & u_{32} \\ 0 & 0 & u_{33} \end{bmatrix}$$

The elements of $S_o^T(k)\ \phi_o^T(k)$ are denoted by b_{ij}'s. This QR decomposition can be implemented, as shown in Fig 6.11 where not only $S_o^T(k)\phi^T(k)$ but also $U_o^T(k)$ are fed into the systolic array.

However, we can use the upper triangular shape of $U_o^T(k)$ to eliminate feeding $U_o^T(k)$ into the systolic array. When the QR decomposition is applied to it, an upper triangular matrix does not change, because it is already upper triangular. Hence, we can prestore $U_o^T(k)$ at the beginning of filtering and feed only the product of $S_o^T(k)$ and $\phi^T(k)$, as shown in Figure 6.12. It reduces the computational time by three clock cycles. Note that $U_o^T(k)$ is assumed constant, which is usually valid in typical tracking environments. However, if $U_o^T(k)$ is not constant, it has to be loaded at every filtering instant. This QR decomposition requires 6 processing elements.

At the end of the calculation of Equation (6.1), $S_o^T(k+1|k)$ is stored in the systolic array, as shown in Figure 6.6. The assumption made earlier in describing the implementation of Equation (6.2) is validated.

We can now develop a complete implementation of Processor 1 by combining architectures in Figures 6.7, 6.9, 6.10 and 6.12. Figure 6.13 shows the resulting implementation. The systolic array for the QR decomposition in Equation (6.2) is shown on the top of Figure 6.13, while the systolic array for Equation (6.1) is shown at the bottom. The implementation of Equation (6.3) is also shown besides the systolic array for Equation (6.2).

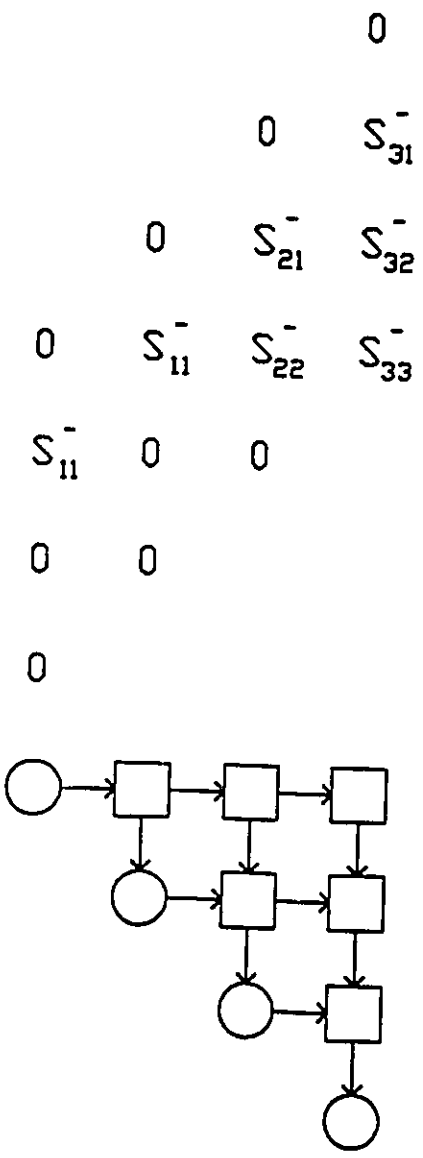


Figure 6.5 Implementation for Equation (6.2)

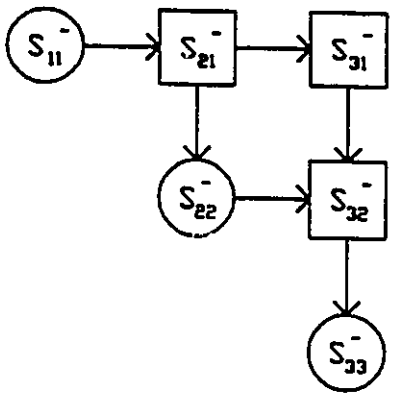


Figure 6.6 Contents of the systolic array for Equation (6.1) at the completion of Equation (6.1)

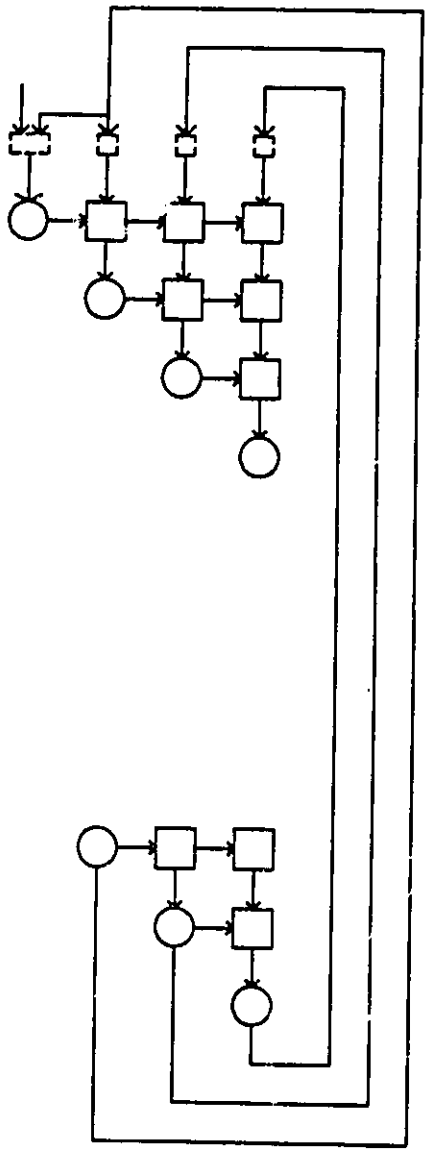


Figure 6.7 Connection of two systolic arrays for Equation (6.2)

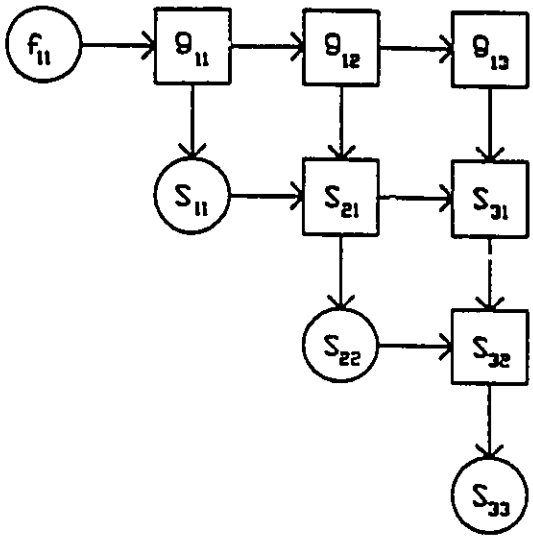


Figure 6.8 Contents of the systolic array for Equation (6.2) at the completion of Equation (6.2)

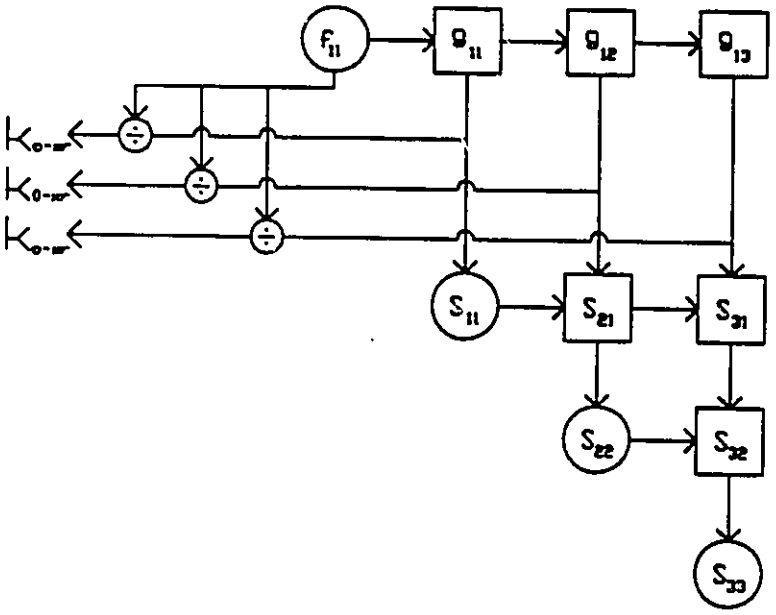
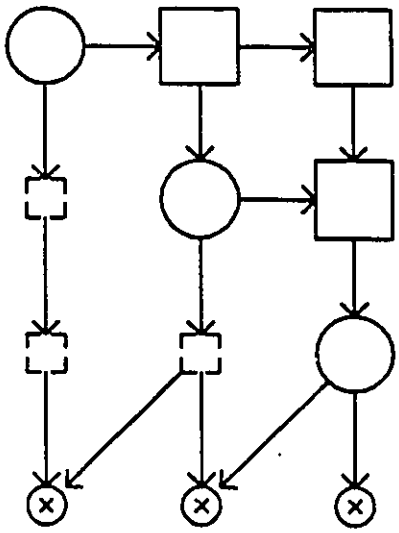
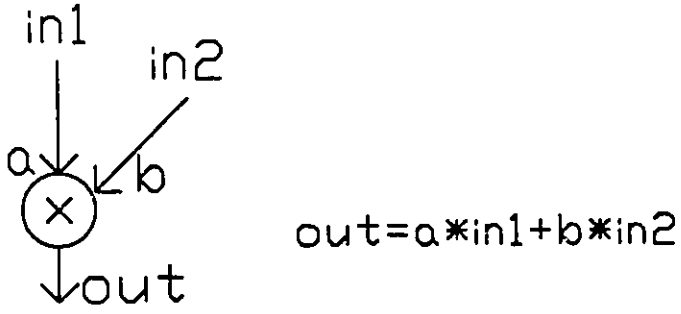


Figure 6.9 Calculation of $K_o(k)$, Equation (6.3)



(a)



(b)

Figure 6.10 Multiplication of $S_o^T(k)\phi^T(k)$

a) structure

b) function of a cell

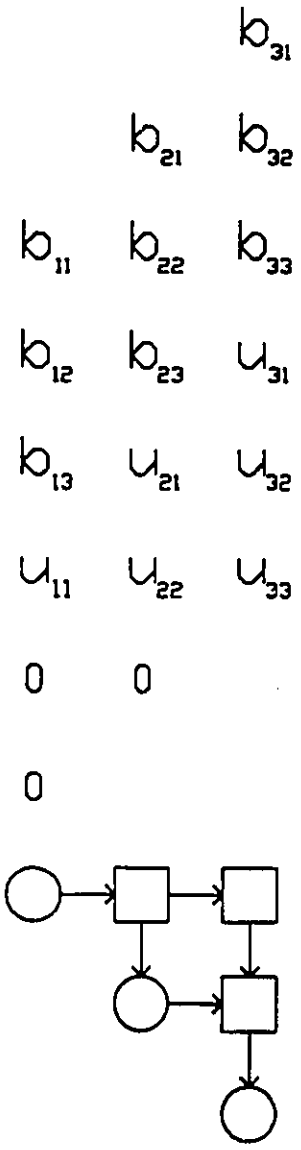


Figure 6.11 Implementation for Equation (6.1)

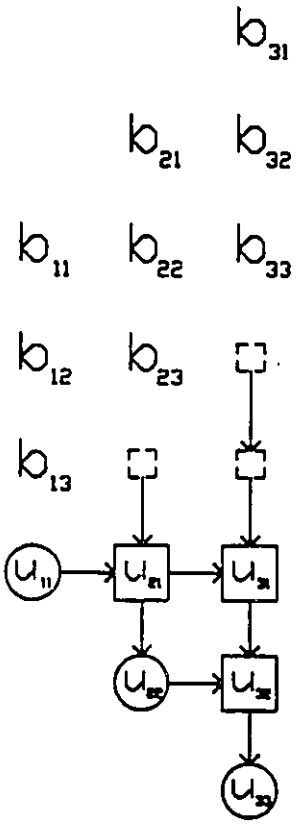


Figure 6.12 Simplified implementation for Equation (6.1)

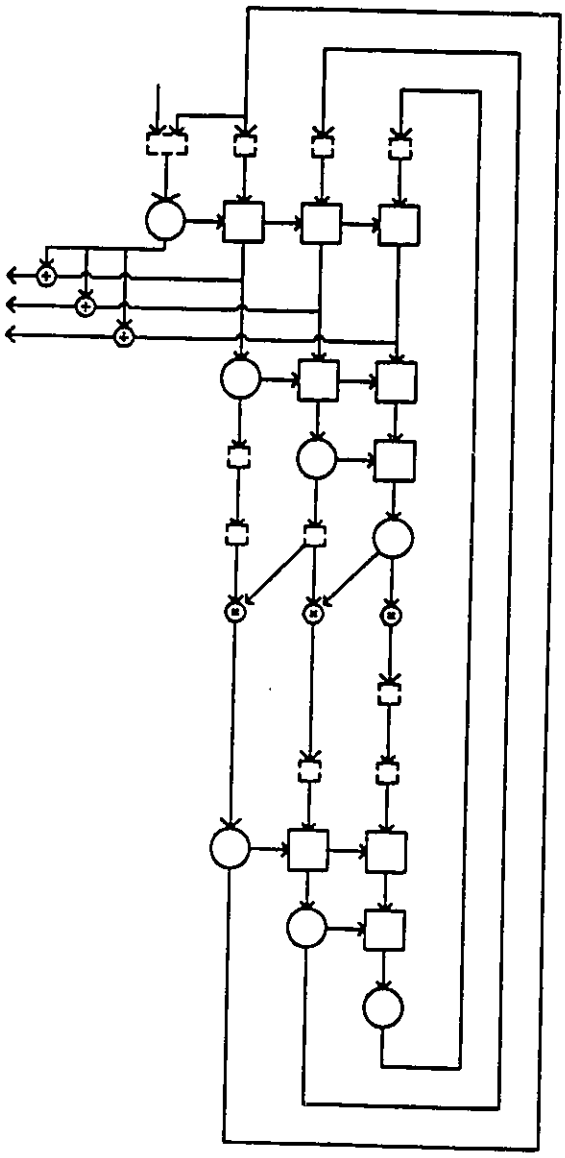


Figure 6.13 Implementation of Processor 1A for the LOS frame

6.1.2 Processor 1B for the LOS frame

Like Processor 1A, Processor 1B implements Equations (6.1), (6.2), and (6.3) to update the square root $S_o(k)$ of the state estimate error covariance matrix $P_o(k|k-1)$ and to calculate the Kalman gain $K_o(k)$ for the LOS frame. The difference between Processor 1A and Processor 1B is that two QR systolic arrays in Processor 1B assume that the top row of an input matrix enters the systolic array first, whereas those in Processor 1B assume the opposite. We use a systolic array on the top of Figure 6.4 for Equation (6.2), and one at the bottom for Equation (6.1).

The implementation of Equation (6.2) below is first considered:

$$\begin{bmatrix} F_o(k) & G_o(k) \\ 0 & S_o^T(k) \end{bmatrix} = Q_1 \begin{bmatrix} V^T(k) & 0 \\ S_o^T(k|k-1)H_o^T(k) & S_o^T(k|k-1) \end{bmatrix} \tag{6.2}$$

It is rewritten below in full for the x-axis:

$$\begin{bmatrix} f_{11} & g_{11} & g_{12} & g_{13} \\ 0 & s_{11} & s_{21} & s_{31} \\ 0 & 0 & s_{22} & s_{32} \\ 0 & 0 & 0 & s_{33} \end{bmatrix} = Q_1 \begin{bmatrix} \sigma_r & 0 & 0 & 0 \\ \bar{s}_{11} & \bar{s}_{11} & \bar{s}_{21} & \bar{s}_{31} \\ 0 & 0 & \bar{s}_{22} & \bar{s}_{32} \\ 0 & 0 & 0 & \bar{s}_{33} \end{bmatrix}$$

Figure 6.14 shows an implementation of Equation (6.2) with the flow of an input data matrix. This implementation which consists of 10 processing elements shows that, in contrast to the implementation in Figure 6.5, the elimination of any row of the input matrix from the input data stream is impossible, because of the reversed direction of the input data flow.

With the assumption that the systolic array for Equation (6.1) stores $S_o(k+1|k)$ at the end of Equation (6.1), as shown in Figure 6.15, the required data flow, illustrated in Figure 6.14, is realized when systolic arrays for Equations (6.1) and (6.2) are connected through 6 delay elements, as shown in Figure 6.16. The validity of the assumption regarding $S_o(k+1|k)$ will be shown later.

Since the systolic array for Equation (6.1) stores only an upper triangular portion of $S_o(k+1|k)$, as shown in Figure 6.15, we need to generate zeroes corresponding to the zero elements at the bottom left of the matrix on the righthand side of Equation (6.2). It can be accomplished by making boundary cells generate zeroes after they pass out their contents.

As in Processor 1A, Equation (6.3), the calculation of the Kalman gain $K_o(k)$ can be performed by reading out $F_o(k)$ and $G_o(k)$ stored in the systolic array for Equation (6.2) at the end of the calculation of Equation (6.2), and dividing $G_o(k)$ by $F_o(k)$. Its implementation which consists of 3 dividers is illustrated in Figure 6.17.

We now consider the implementation of Equation (6.1):

$$\begin{bmatrix} S_o^T(k+1|k) \\ 0 \end{bmatrix} = Q_1 \begin{bmatrix} S_o^T(k)\,\phi^T(k) \\ U_o^T(k) \end{bmatrix} \tag{6.1}$$

Equation (6.1) has two phases: a matrix–matrix multiplication and the QR decomposition. The matrix–matrix multiplication in Equation (6.1) can be performed by combining two columns of $S_o^T(k)$ for each column of the output matrix, as in Section 6.1.1. Figure 6.18 shows an implementation of the multiplication of $S_o^T(k)$ and $\phi^T(k)$. This implementation requires three arithmetic units whose function is decribed in Figure 6.10 (b), but it does not require any

delay elements, in contrast to the implementation in Figure 6.10 (a). The zero elements in $S_o^T(k)$, not stored in the systolic array, but needed in multiplying $S_o^T(k)$ and $\phi_o^T(k)$, can be generated by the boundary cells after the content of the boundary cells are passed out.

Figure 6.19 shows an implementation of the second phase of Equation (6.1), the QR decomposition. In Figure 6.19, the product of $S_o^T(k)$ and $\phi_o^T(k)$, represented by the b_{ij}'s, enters the systolic array before $U_o^T(k)$, because the systolic array is designed to receive first the bottom row of an input matrix to be triangularized. This means that $U_o^T(k)$ has to go through the QR decomposition after the product of $S_o^T(k)$ and $\phi_o^T(k)$ goes through, and that $U_o^T(k)$ cannot be prestored in the systolic array for Equation (6.1). However, $U_o^T(k)$ can be prestored in the systolic array for Equation (6.2), and be fetched after $S_o^T(k)$ is read from the systolic array. This means that $U_o^T(k)$ will go through the arithmetic units for the multiplication of $S_o^T(k)$ and $\phi(k)$, when $U_o^T(k)$ is fetched from the systolic array for Equation (6.2) to that for Equation (6.1). Hence, for $U_o^T(k)$ to go through the arithmetic units intact, the multiplication constants "a" and "b" for arithmetic units in Figure 6.10 (b) should be set to 1 and 0 respectively, while $U_o^T(k)$ goes through the arithmetic units.

Note that $S_o^T(k+1|k)$ is stored in the systolic array at the end of Equation (6.1), as shown in Figure 6.15. The assumption made earlier in describing the implementation of Equation (6.2) is validated.

After studying necessary structures for all the parts in Processor 1B, we can now present a complete implementation of Processor 1B by combining Figures 6.16, 6.17, 6.18, and 6.19. The resulting implementation is shown in Figure 6.20. The systolic array for Equation (6.2) is shown on the top of Figure 6.20, while that for Equation (6.1) is shown on the bottom. As mentioned earlier,

this architecture requires bidirectional interconnections between two systolic arrays and among processing elements in contrast to the architecture in Figure 6.13.

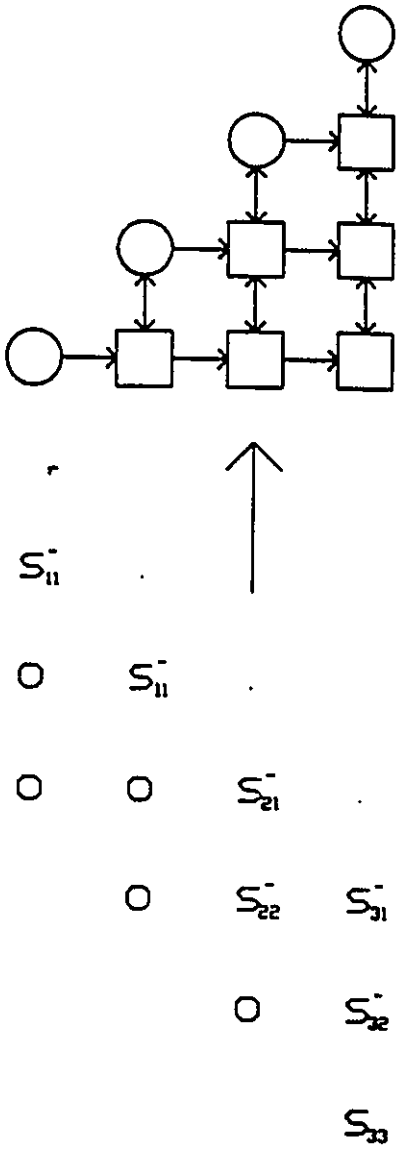


Figure 6.14 Implementation of Equation (6.2)

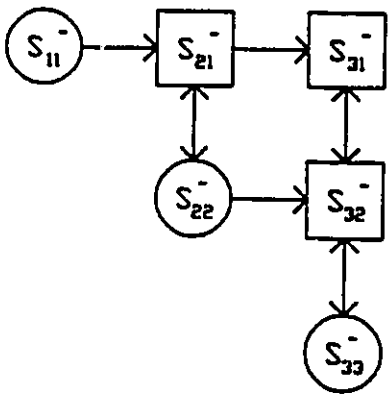


Figure 6.15 Systolic array for Equation (6.1) at the completion of Equation (6.1)

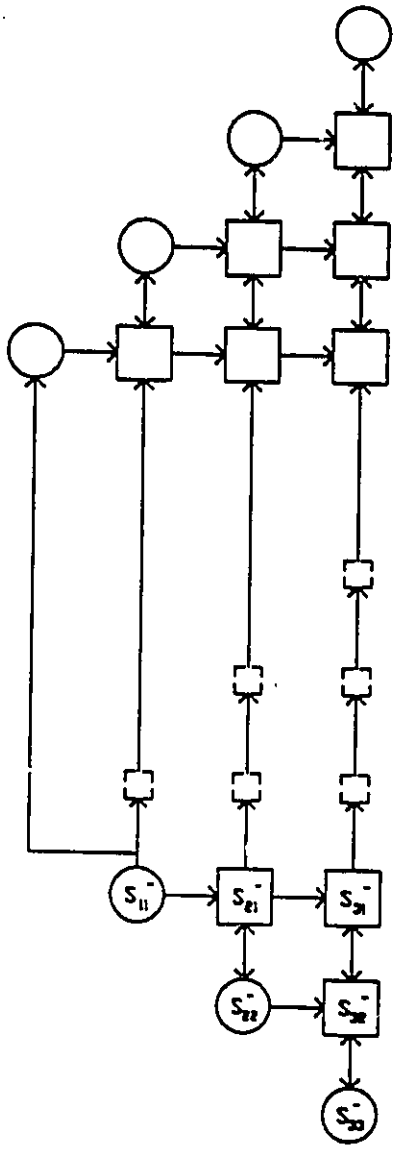


Figure 6.16 Connection of two systolic arrays for Equation (6.2)

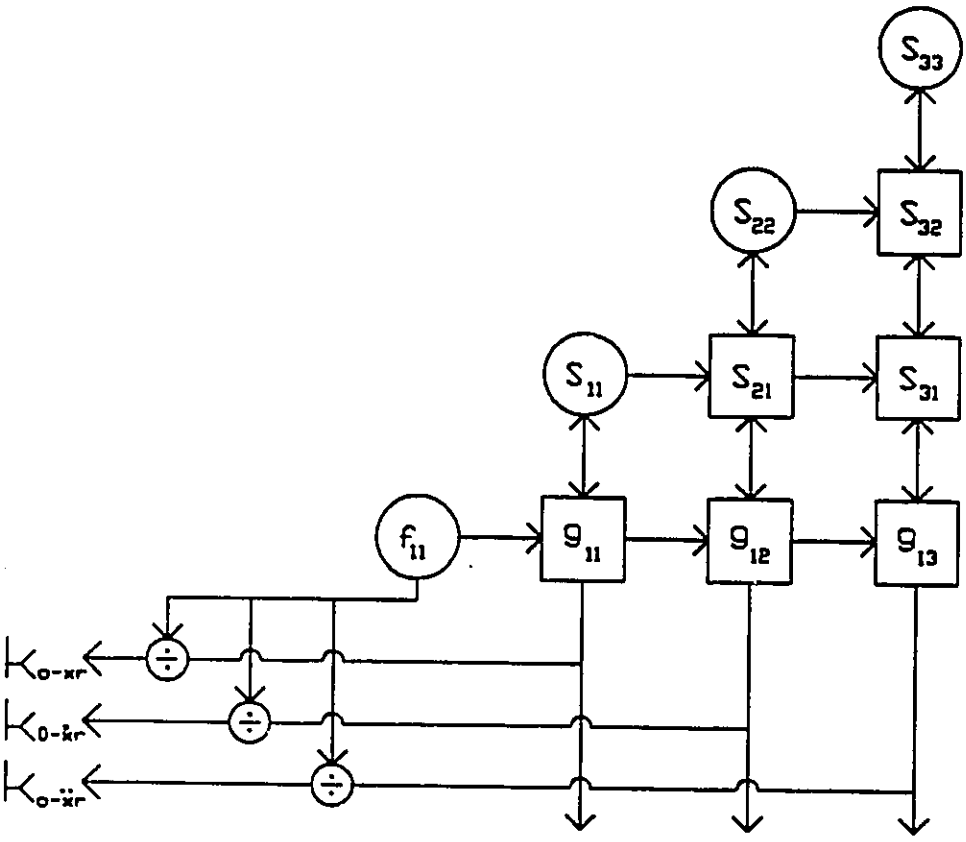


Figure 6.17 Calculation of $K_0(k)$, Equation (6.3)

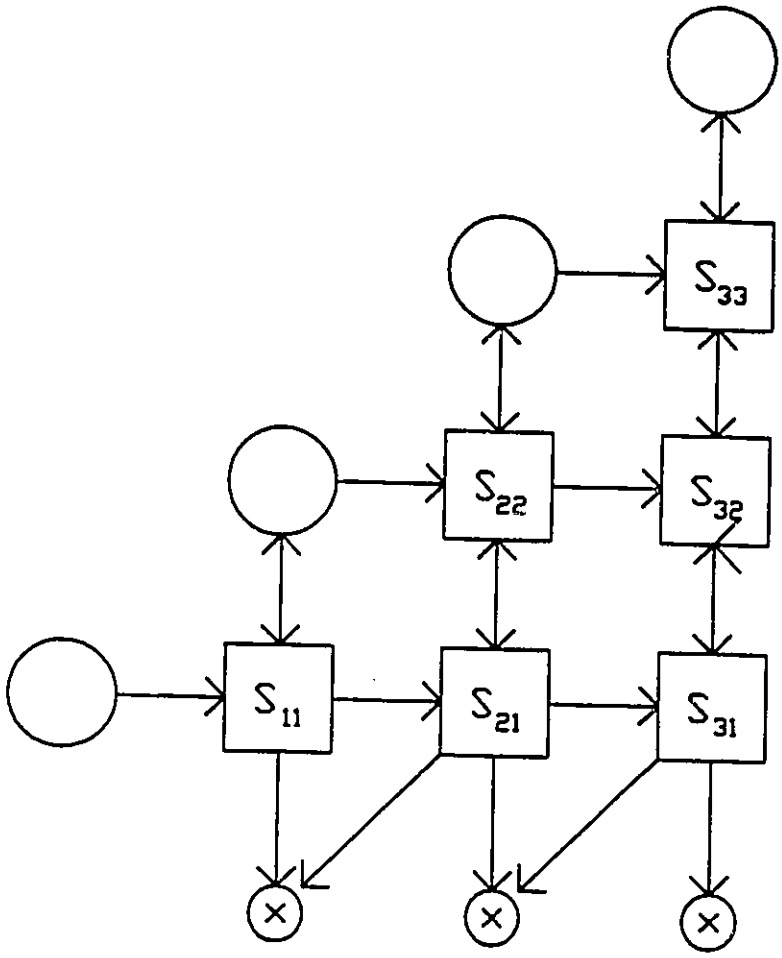


Figure 6.18 Multiplication of $S_o^T(k)$ and $\phi^T(k)$

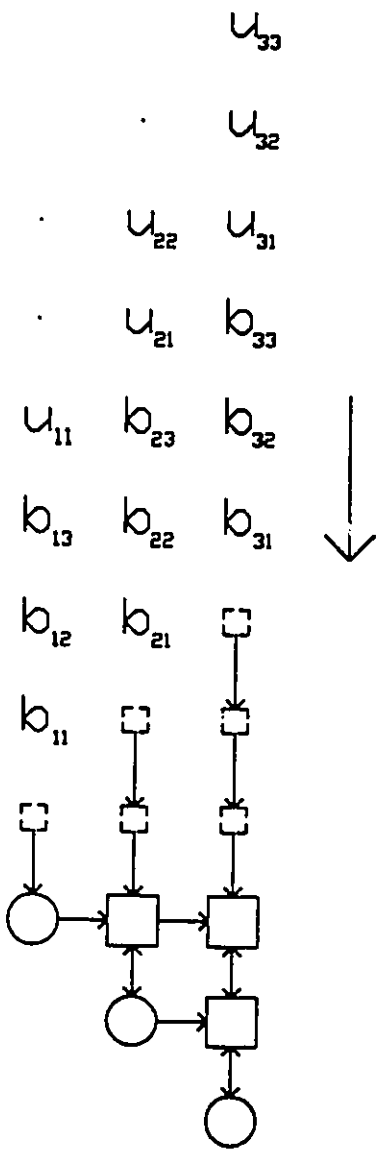


Figure 6.19 QR decomposition for Equation (6.1)

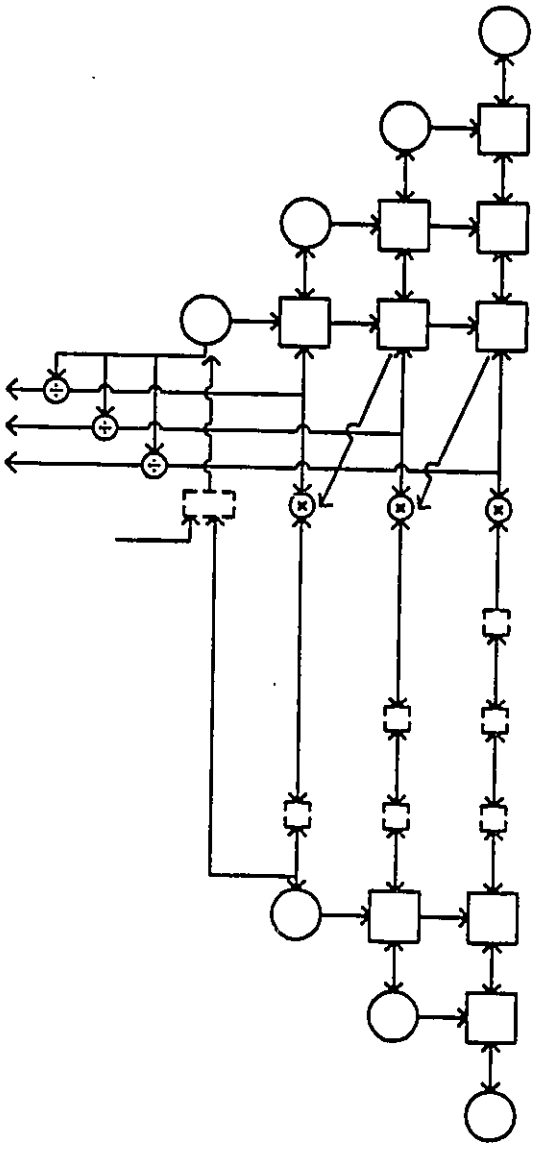


Figure 6.20 Implementation of Processor 1B for the LOS frame

6.1.3 Processors for the Coordinate Transformation of the Kalman Gain and for the Reference Cartesian Frame

The decoupled ESRCF uses the same equations as the decoupled ECKF to compute a state estimate, a prediction and the Kalman gain K(k) from $K_o(k)$. In other words, Equations (6.4), (6.5), and (6.6) for the decoupled ESRCF are identical to the corresponding equations (5.4), (5.5), and (5.6) for the decoupled ECKF.

In designing a parallel architecture for the extended SRCF, the same design principles should be followed as in designing a parallel architecture for the extended covariance KF. The architecture for the decoupled ECKF has been developed to reduce the required hardware and computational time, bearing in mind desirable characteristc for parallel architectures. Hence, the architecture for the decoupled ECKF equations (5.4), (5.5), and (5.6) should be employed to implement the decoupled ESRCF equations (6.4), (6.5), and (6.6).

However, there have been two types of implementation for the decoupled ECKF equations (5.4), (5.5), and (5.6): broadcast and pipeline processing architectures. For Equation (5.4) which transforms the Kalman gain from the LOS to reference Cartesian coordinates, Figure 5.6 shows a broadcast processing implementation, whereas Figure 5.7 shows a pipeline processing architecture. Similarly, for Equations (5.5) and (5.6), Figures 5.11 and 5.14 show broadcast and pipeline processing architectures respectively. As discussed in Section 5.1.2, the pipeline architecture requires a simpler data bus connection and fewer processing elements than the broadcast architecture. However, the former architecture needs to introduce a skew into data streams for the synchronization of incoming data streams, and requires more computation than the latter. The broadcast processing architectures in Figures 5.6 and 5.14 seem more suitable

than the pipeline processing architectures, because the former architectures are fast, easy to control, and do not require any time delay element. Yet, the selection of type of processing should be based on design criteria.

6.2 Benefits of the Decoupling Technique

In this section, we examine how the use of the decoupling technique reduces hardware and computational time requirements for the QR decomposition. The computationally demanding QR decomposition is a major part of the time and measurement updates of the square root $P^{1/2}(k)$ of the state estimate error matrix P(k).

We assume that the QR systolic array, described in Section 3.2.2, is used for the QR decomposition. The QR decomposition of a n–by–n matrix requires 3n–2 clock cycles on a n–by–n triangular array which consists of n(n+1)/2 processing elements: n boundary and n(n–1)/2 internal processing elements. The QR decomposition of m decoupled (n/m)–by–(n/m) matrices requires (3n/m)–2 clock cycles on m (n/m)–by–(n/m) triangular systolic arrays. Each systolic array has (n/m)(n/m+1) processing elements: (n/m) boundary cells and (n/m)(n/m–1) internal cells. The total number of processing elements in m systolic arrays is n(n/m+1). It consists of n boundary cells and n(n/m–1) internal cells.

A comparison of the requirements for the QR decomposition of a n–by–n array and those for m (n/m)–by–(n/m) arrays shows that the QR decomposition of m decoupled matrices requires approximately m times less processing elements and computational time than that of 1 n–by–n matrix. However, the numbers of boundary processing elements for the above two cases are the same.

The product of the reduction ratio for the number of processing elements and that for the computational time is m^2, and it is greater than the overall

computational reduction ratio which was found to be between m^2 and m in Section 4.4.1. This can be explained by the fact that the decoupling technique reduces only the number of internal cells, not that of computationally intensive boundary processing elements.

6.3 Requirements of Hardware and Computational Time

In this section, we study the requirements of hardware and computational time for the proposed parallel implementations of the extended SRCF. Tables 6.1 (a) and 6.1 (b), and 6.2 (a) and 6.2 (b) summarize hardware requirements for Processors 1A and 1B respectively. Tables 6.1 (a) and 6.2 (a) are for a general tracking environment where the dimension of tracking is denoted by m, and the number of state elements is n, whereas Tables 6.1 (b) and 6.2 (b) are for a 3-dimensional tracking with 9 state elements. In these tables, Equations (6.1) and (6.2) are broken into two parts to separate the requirements for a multiplication from those for the QR decomposition in each equation.

A comparison of Tables 6.1 (a) and 6.1 (b) shows that Processors 1A and 1B require the same number of arithmetic units except for delay elements. Processor 1A requires $\frac{n^2}{m}$ delay elements, whereas Processor 1B requires $(\frac{n^2}{2m} + \frac{n}{2})$ delay elements. For a typical tracking where n=9, m=3, the former architecture requires 27 delay elements, whereas the latter requires 18 delay elements. The reason Processor 1A requires more delay elements than Architecture 2 is twofold:

a) The delay elements in Processor 1B are shared both in the time and measurement updates due to the bidirectional bus.

b) The delay elements which synchronize $S_o^T(k-1)$ for the multiplication

$S_o^T(k-1)$ and $\phi_o^T(k-1)$ in Processor 1A are not necessary in Architecture 2.

Tables 6.3 (a) and (b), and 6.4 (a) and (b) present the total hardware requirements for the parallel implementation of the extended SRCF, using Processors 1A and 1B respectively. Tables 6.3 (a) and 6.4 (a) express requirements in terms of m and n, whereas Tables 6.3 (b) and 6.4 (b) are for a 3–dimensional tracking with 9 state elements. These tables assume that Equations (6.4), (6.5), and (6.6) are implemented using a broadcast processing structure, as shown in Figures 5.8 and 5.14. The implementations of the ESRCF using Processors 1A and 1B require the same number of arithmetic units except for delay elements, as expected.

We now determine computational time requirements for the implementation of Equations (6.1), (6.2), and (6.3) on Processors 1A and 1B. It may be carried out by counting the number of clock cycles it takes for the elements in the top row of a systolic array to go through the QR decomposition and to finally reach the bottom row of the other systolic array. This process is shown in Figures 6.21 and 6.22 for Equations (6.2) and (6.1) respectively. The numbers on the right hand side of Figure 6.21 indicate the number of clock cycles needed by the elements in the top row of the systolic array for the QR decomposition in Equation (6.1) to reach a particular row in the process of computing Equation (6.2). Figure 6.21 shows that it takes 7 cycles to complete Equation (6.2).

Similarly, the numbers in Figure 6.22 indicate the number of steps required by the elements in top row of the systolic array for Equation (6.2) to reach a particular row in the process of computing Equation (6.1). Figure 6.22 shows that Equation (6.1) requires 8 clock cycles to implememt.

Equation (6.3), which calculates the Kalman gain for the LOS frame using the results of Equation (6.2), can be performed while Equation (6.1) is executed after the completion of Equation (6.2). Hence, no additional clock cycle is required to implement Equation (6.3).

The numbers of clock cycles required to perform Equations (6.1), (6.2), and (6.3) on Processor 1B can be determined in the same way as those required to implement Equations (6.1), (6.2), and (6.3) on Processor 1A. They are shown in Figures 6.23 and 6.24. Theses figures show that the implementation of Equations (6.1) and (6.2) takes 10 cycles each. That is, Equations (6.1) and (6.2) take longer to implement on Processor 1A than on Processor 1B. This is due to the fact that in the implementation of Equations (6.1) and (6.2) using Processor 1A the first two rows of zeroes are eliminated in an input data stream for the measurement update, and that U(k), which takes 3 cycles to feed, is prestored in the systolic array for the time update.

Since the decoupled ESRCF equations (6.4), (6.5), and (6.6) are the same as the decoupled ECKF equations (5.4), (5.5), and (5.6) respectively, and they are implemented on the same architecture as the decoupled ECKF equations, the computational requirements for Equations (6.4), (6.5), and (6.6) are the same as those for Equations (5.4), (5.5), and (5.6). Hence, Table 5.2 which summarizes the numbers of cycles needed to implement Equations (5.4), (5.5), and (5.6) may be used for Equations (6.4), (6.5), and (6.6).

In implementing the complete decoupled ESRCF, parallelism can be further explored by implementing more than one independent operations simultaneously. For example, as in the parallel implementation for the ECKF, the processor for the LOS coordinates can continue computing $S_o(k)$ and $S_o(k+1|k)$ after calculating $K_o(k)$, while the processor for the reference Cartesian coordinates

calculates $\hat{X}(k)$ and $\hat{X}(k+1|k)$.

Figures 6.25 and 6.26 show how the decoupled ESRCF can be performed in parallel on Architectures 1 and 2. In these two figures, Equations (6.4) and (6.5) are assumed to be divided into two and three parts, respectively, in the same way as Equations (5.4) and (5.5) for the simplified ECKF are divided in Section 5.2. These divisions minimize computational time requirements by reducing the waiting period. Figures 6.25 and 6.26 indicate that for 3-dimensional tracking one iteration of the decoupled ESRCF requires 15 clock cycles on Architecture 1, and 20 clock cycles on Architecture 2.

Table 6.1 Hardware Requirements for Processor 1A
(a) for m–dimensional tracking with n state elements
(b) for 3–dimensional tracking wiht 9 state elements

Table 6.1 (a)

Equation number	Number of operations per iteration				
	PE.	Mult.	Div.	Mux.	Delay.
6.1.1	0	n	0	0	$\frac{n}{2}(\frac{n}{m}-1)$
6.1.2	$\frac{n}{2}(\frac{n}{m}+1)$	0	0	0	$\frac{n}{2}(\frac{n}{m}-1)$
6.2.1	0	0	m–1	0	0
6.2.2	$\frac{1}{2}(n+m)(\frac{n}{m}+2)$	0	0	m	n
6.3	0	0	n	0	0
Total	$\frac{n^2}{m}+2n+m$	n	n+m–1	m	$\frac{n^2}{m}$

Table 6.1 (b)

Equation number	Number of operation per iteration				
	PE.	Mult.	Div.	Mux.	Delay.
6.1.1	0	9	0	0	9
6.1.2	18	0	0	0	9
6.2.1	0	0	2	0	0
6.2.2	30	0	0	3	9
6.3	0	0	9	0	
Total	48	9	11	3	27

Table 6.2 Hardware Requirements for Processor 1B
(a) for m-dimensional tracking with n state elements
(b) for 3-dimensional tracking with 9 state elements

Table 6.2 (a)

Equation number	Number of operations per iteration PE.	Mult.	Div.	Mux.	Delay.
6.1.1	0	n	0	0	0
6.1.2	$\frac{n}{2}(\frac{n}{m}+1)$	0	0	0	$\frac{n}{2}(\frac{n}{m}+1)$
6.2.1	0	0	m−1	0	0
6.2.2	$\frac{1}{2}(n+m)(\frac{n}{m}+2)$	0	0	m	0
6.3	0	0	n	0	0
Total	$\frac{n^2}{m}+2n+m$	n	n+m−1	m	$\frac{n}{2}(\frac{n}{m}+1)$

Table 6.2 (b)

Equation number	Number of operation per iteration				
	PE.	Mult.	Div.	Mux.	Delay.
6.1.1	0	9	0	0	0
6.1.2	18	0	0	0	18
6.2.1	0	0	2	0	0
6.2.2	30	0	0	3	0
6.3	0	0	9	0	0
Total	48	9	11	3	18

Table 6.3 Total Hardware Requirements for the ESRCF (using Processor 1A)
(a) for m-dimensional tracking with n state elements
(b) for 3-dimensional tracking with 9 state elements

Table 6.3(a)

PE.	$\frac{n^2}{m}+2n+m$
Mul.	2nm+n+1(for 3-dimensional tracking)
	2nm+n (for 2-dimensional tracking)
Div.	n+m-1
Add.	n+m
Cordic	2m-2
Mux.	m
Delay.	$\frac{n^2}{m}$

Table 6.3(b)

PE.	48
Mul.	64
Div.	11
Add.	12
Cordic	4
Mux.	3
Delay.	27

Table 6.4 Total Hardware Requirements for the ESRCF (using Processor 1B)
(a) for m-dimensional tracking with n state elements
(b) for 3-dimensional tracking with 9 state elements

Table 6.4(a)

PE.	$\frac{n^2}{m}+2n+m$
Mul.	2nm+n+1(for 3-dimensional tracking) 2nm+n (for 2-dimensional tracking)
Div.	n+m-1
Add.	n+m
Cordic	2m-2
Mux.	m
Delay.	$\frac{n^2}{2m}+\frac{n}{2}$

Table 6.4(b)

PE.	48
Mul.	64
Div.	11
Add.	12
Cordic	4
Mux.	3
Delay.	18

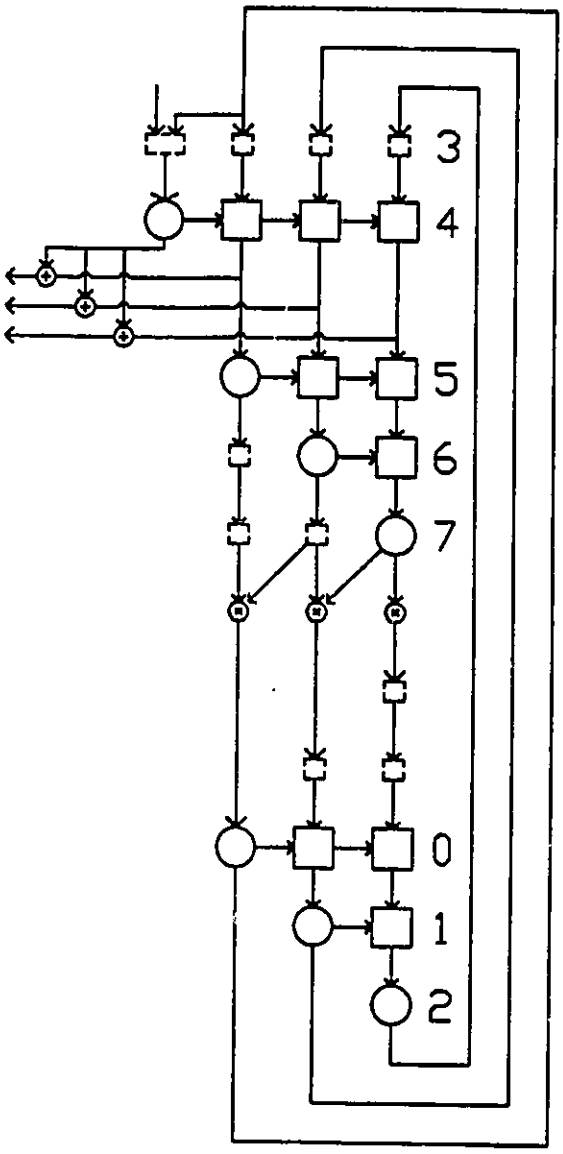


Figure 6.21 Computational time requirements for the measurement update, Equation (6.2), using Processor 1A

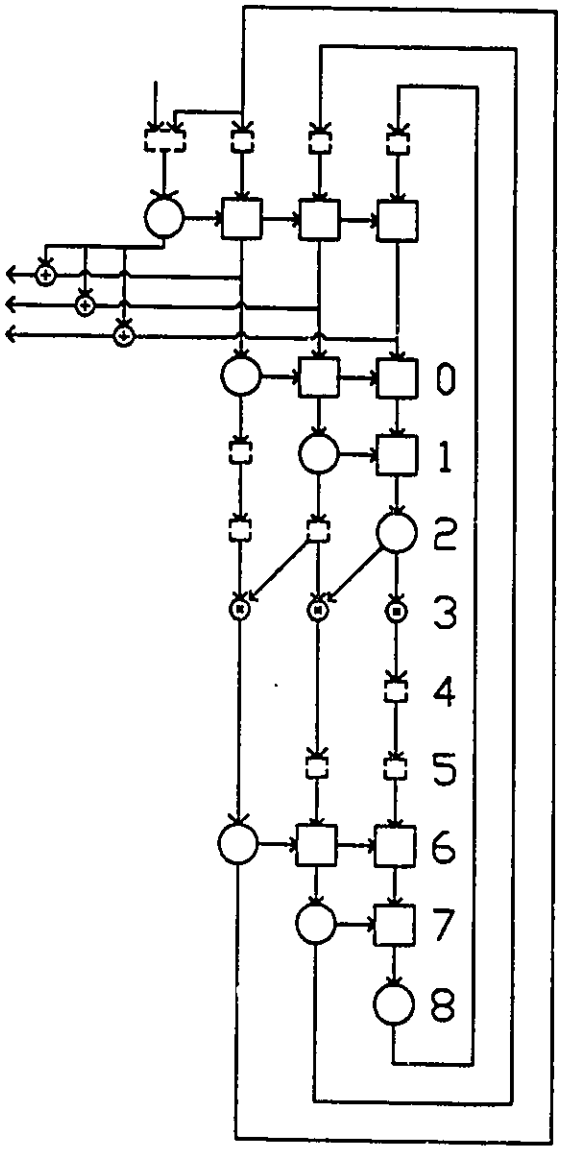


Figure 6.22 Computational time requirements for the time update, Equation (6.1), using Processor 1A

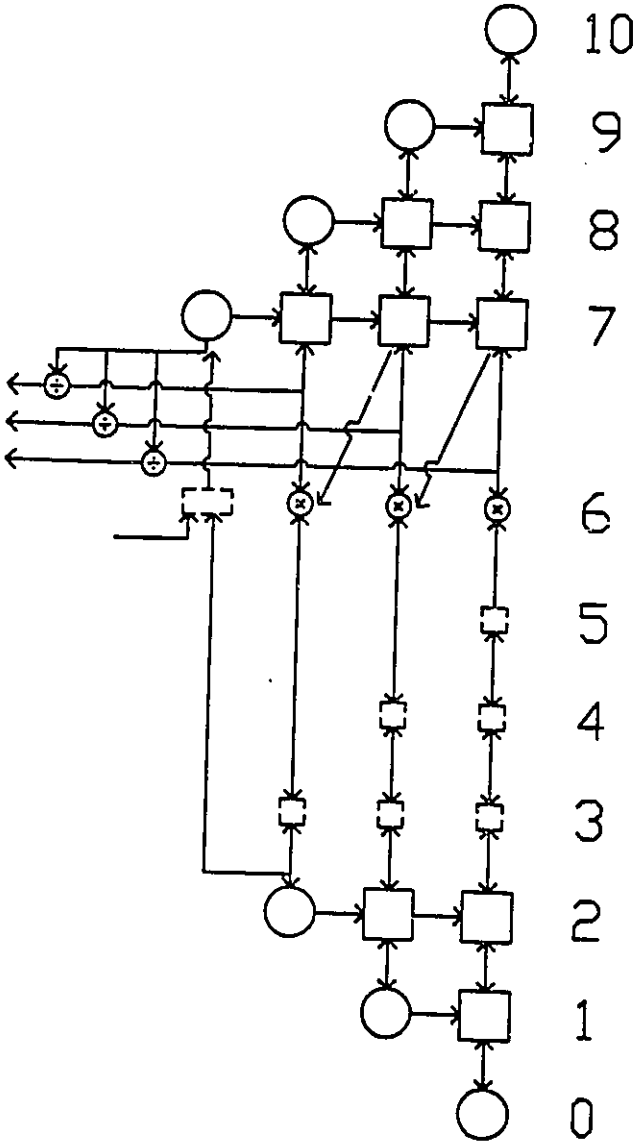


Figure 6.23 Computational time requirements for the measurement update, Equation (6.2), using Processor 1B

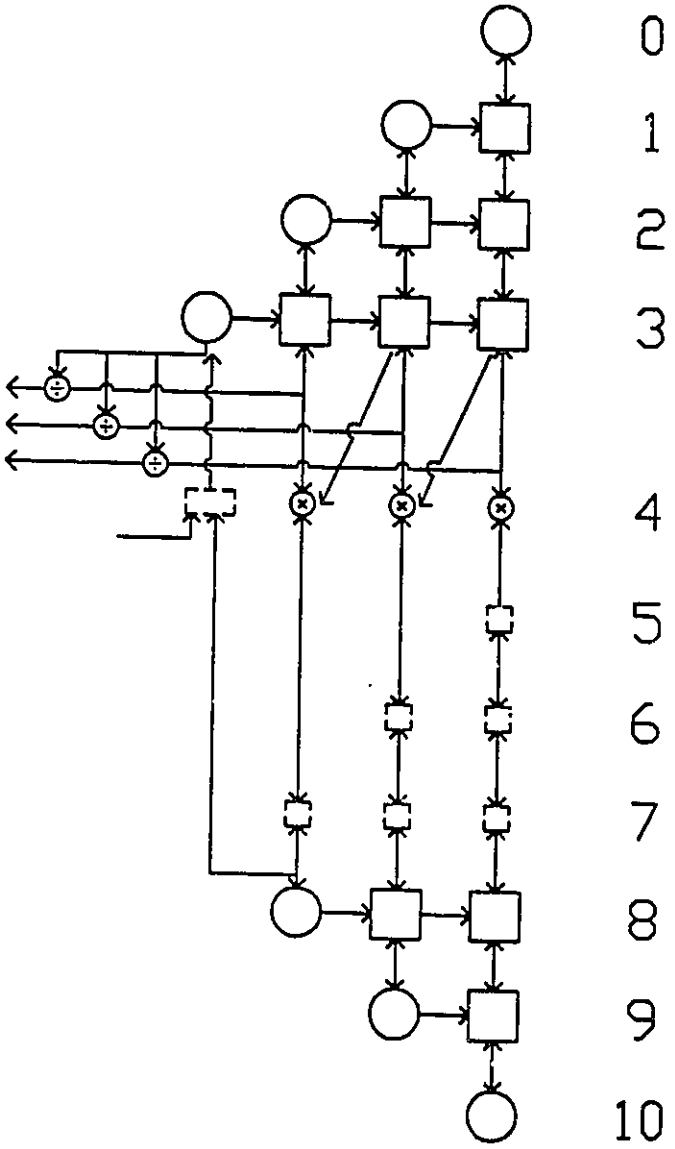


Figure 6.24 Computational time requirements for the time update, Equation (6.1), using Processor 1B

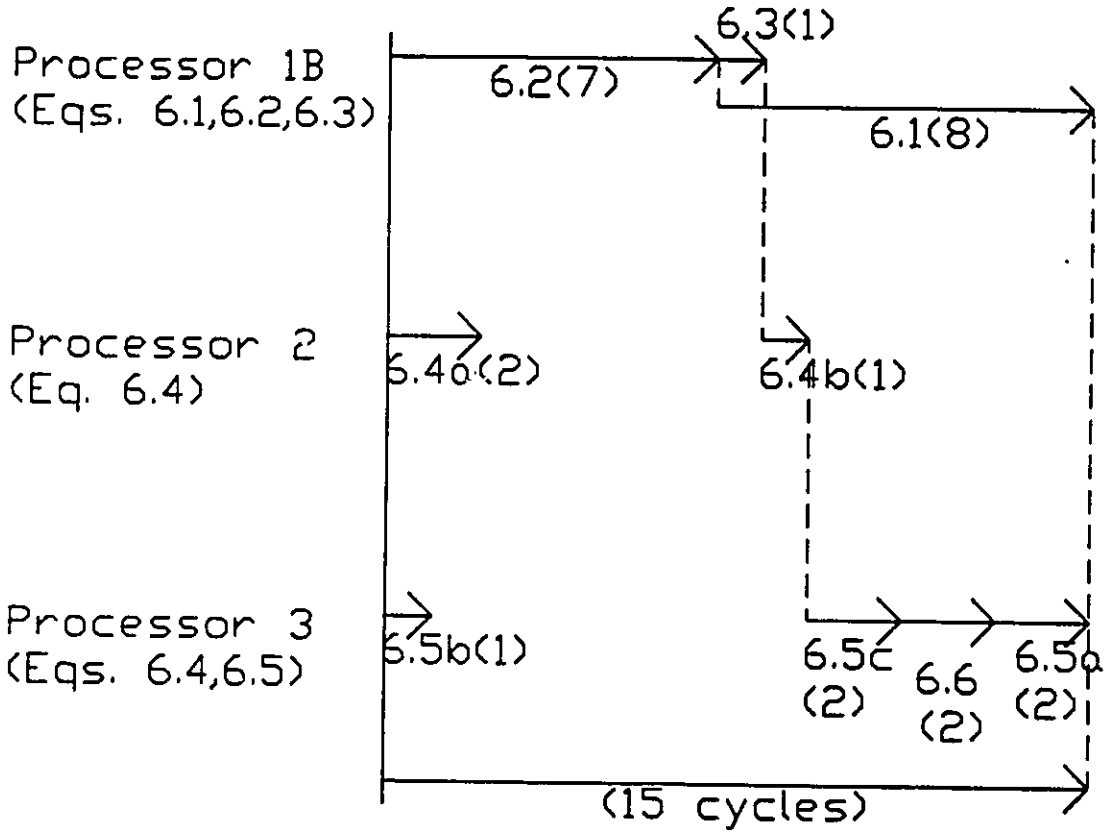


Figure 6.25 Parallel computation of the simplified ESRCF using Architecture 1 (numbers in brackets indicate the number of clock cycles required for each step)

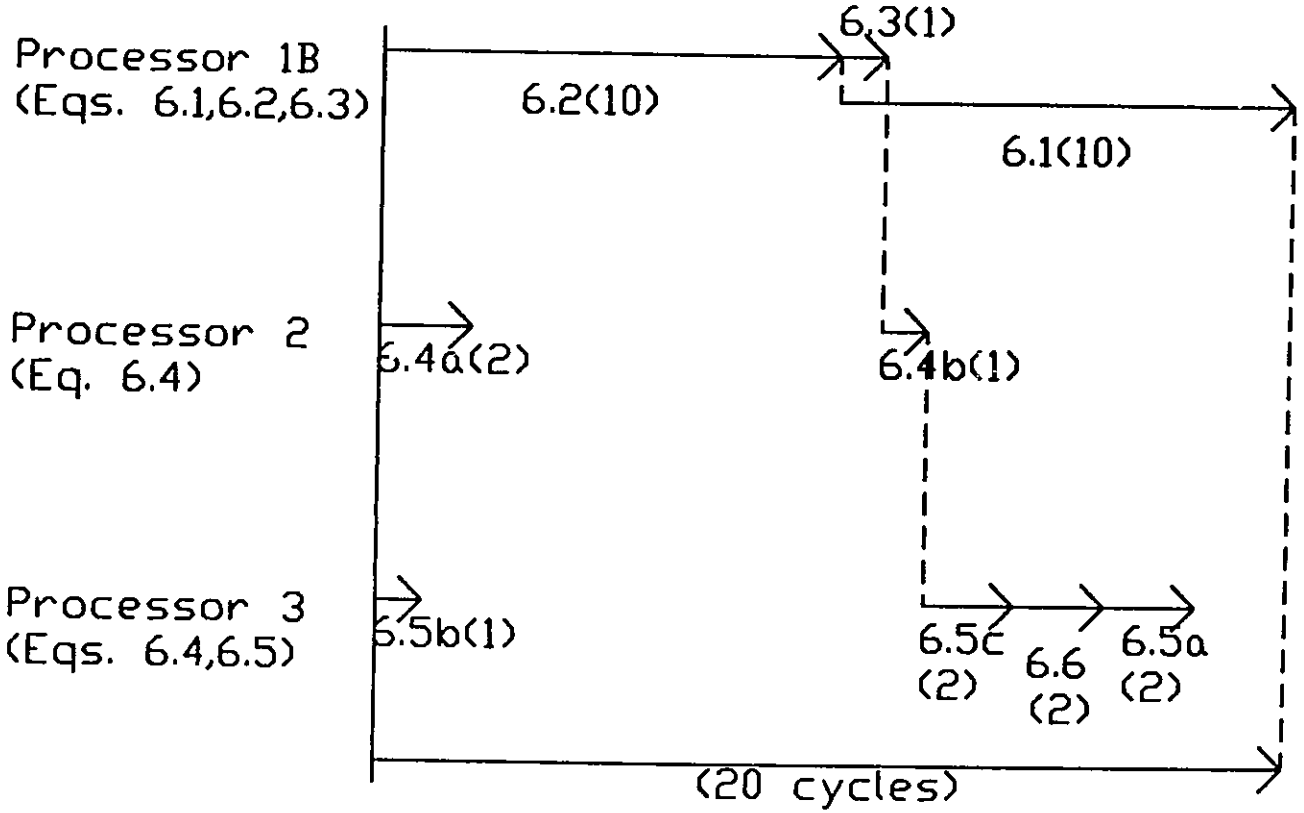


Figure 6.26 Parallel computation of the simplified ESRCF using Architecture 2 (numbers in brackets indicate the number of clock cycles required for each step)

6.4 Comparisons

The two architectures developed in this chapter are efficient parallel implementations of the extended SRCF. We now compare these architectures with others. Since the architectures so far published in the literature are for either the standard SRCF or the standard covariance KF, and the architectures developed in this chapter are for the ESRCF, it is possible to compare only the architectures for Equations (6.1), (6.2), and (6.3), which are common to both the ESRCF and SRCF. Although, this comparison is still limited, it can still give a good idea how efficient the architectures developed in this chapter are. Equations (6.1), (6.2), and (6.3) require more computation than the remaining equations.

We compare the proposed architectures in this chapter with the Sung–Hu architecture for the standard SRCF which has been reviewed in Section 3.3.2. Sung and Hu have explored parallelism by separating the KF equations into two loosely dependent groups. One of these two group performs measurement and time updates on the square root of the state estimate error covariance matrix, whereas the another group estimates state vectors $\hat{X}(k)$ and $\hat{X}(k|k-1)$. They have employed for the time update a n–by–n triangular systolic array. For the measurement update, they have employed a trapezoidal systolic array of m rows of processing elements with n+m processing elements on the top row, and n+1 elements at the bottom, utilizing the sparse nature of an input matrix. The letters m and n denote the tracking dimension and the number of state elements respectively. In Sung and Hu's implementation, the processing of Equations (6.1) and (6.2) requires 4n+m–1 clock cycles together.

The architectures developed in this chapter require m (n/m)–by–(n/m) triangular systolic arrays for the time update, m (n/m + 1)–by–(n/m + 1) triangular systolic arrays for the measurement update, and 15 or 20 clock cycles

for 3-dimensional tracking depending on the type of an implementation.

A comparison between the Sung-Hu architecture and the architectures developed in this chapter on the hardware requirements for the time update confirms that the decoupling technique reduces the number of processing elements by a factor of approximately m. However, for the measurement update, the Sung-Hu architecture requires nm + m(m+1)/2 processing elements, and the architectures proposed in this chapter require (n+m)(n/m + 1)/2 processing elements. These requirements are difficult to compare, for their relationship changes with m and n. Nevertheless, for 3-dimensional tracking with 9 state elements, The Sung-Hu architecture is found to require 33 processing elements, whereas the architectures developed in this chapter is found to require 30 processing elements. These requirments are comparable. This means that Sung and Hu's use of the sparse nature of an input matrix in the measurement update is as effective as the use of the decoupling technique in this chapter. For both the time and measurement updates, the architectures developed in this chapter require overall m/2 times less number of processing elements than the architecture of Sung and Hu for the time and measurement updates. The reduction factor of m/2 is obtained as an average of the reduction ratio for the time update and that for the measurement update. Specifically, for 3-dimensional tracking, the reduction ratio is 3/2.

For computational time requirements, Sung and Hu's architecture requires 4n+m-1 clock cycles, which corresponds to 38 clock cycles for 3-dimensional tracking with 9 state elements, whereas the architectures proposed in this chapter require 15 or 20 clock cycles depending on the type of implementation. The less computational time requirement for the architectures developed in this chapter is due to the simplification by the use of the decoupling technique, the sparse

nature of $H_o(k)$, and the sparse, band nature of $\phi(K)$. The decoupling technique allows m decoupled (n/m)-by-(n/m) covariance matrices to go through the QR decomposition in parallel. Since the required number of clock cycles for the QR decomposition is directly proportional to the size of an input matrix, the parallel QR decomposition of m small matrices requires less time than the QR decomposition of one big matrix.

In summary, in comparison to Sung and Hu's implementation of Equations (6.1) and (6.2), the use of the decoupling technique and special properties in tracking systems reduces hardware requirements by a factor of m/2 and computational time requirements by a factor of 2.5 (=38/15) or 1.9 (=38/20) for 3-dimensional tracking.

The extended SRCF is known to exhibit better numerical characteristics, and to be more computationally complex than the extended covariance KF [4] [25]. We compare the parallel implementations of these two filters. Table 6.5 summarizes the hardware requirements of the two parallel implementations on the basis of Tables 5.1 and 6.1. We find that the parallel architecture for the ESRCF requires overall slightly more arithmetic units than that for the ECKF: n more processing elements, n less multipliers, n+m+1 more dividers, m more adders, m more multiplexors, and n more delay elements.

In addition to the quantitative comparison, we compare the complexity of processing elements in both architectures. The processing elements in the architecture for the ECKF perform multiplications and pass data to neighboring processing elements, whereas some of the processing elements, boundary cells, in the architecture for the ESRCF perform square root operations. Since the square root operation is much more complex than a simple multiplication, the processing elements in the architecture for the ESRCF are more complex than those for the

ECKF.

The square root operation not only increases the complexity of a processing element implementing it, but also takes longer to implement it. The actual length of one clock cycle in the parallel implementation of the ECKF, which is defined as the longest time duration required to perform appropriate operations by any type of arithmetic unit, is longer than the actual length of one cycle in the parallel architecture for the ECKF. Hence, even though the parallel implementations of the ESRCF require either 15 or 20 clock cycles for one iteration, which is close to 16 clock cycles required by the parallel implementation of the ECKF, the parallel implementation of the ESRCF requires more time than that of the ECKF.

In summary, the parallel implementation of the ESRCF which is numerically superior to the ECKF requires overall slightly more arithmetic units than that for the ECKF. Processing elements in the implementation of the ESRCF are more complex than those for the ECKF, and the actual duration of a clock cycle in the implementation of the ESRCF is longer than that for the ECKF.

Table 6.5 Comparison of the Hardware Requirements of the ECKF and the ESRCF

	ECKF	ESRCF	ESRCF – ECKF
PE.	$\frac{n^2}{m}+n+m$	$\frac{n^2}{m}+2n+m$	n
Mul.	2nm+2n+1	2nm+n+1	–n
Div.	0	n+m–1	n+m–1
Add.	m	n+m	n
Cordic	2m–2	2m–2	0
Mux.	0	m	m
Delay.	$\frac{n^2}{m}-n$	$\frac{n^2}{m}$	m

6.5 Summary

In this chapter, we have developed the first parallel implementations of the extended SRCF for tracking applications. The ESRCF is numerically superior to the ECKF, but it is computationally more demanding than the ECKF. We have maximized the throughput rate of the architectures through the use of pipelining and parallel processing, bearing in mind the desirable characteristics of parallel architectures. We have employed for implementation the simplified ESRCF by the application of a decoupling technique and special properties in matrices in the tracking KF. The reason for choosing the simplified filter is that, as shown in Chapters 4 and 5, in the simplified ESRCF, the computationally demanding time and measurement updates of the state estimate error covariance matrix may be performed for each axis in parallel, and the computational requirements may be reduced significantly by the use of the sparse nature of $H_o(k)$ and the sparse, band nature of $\phi(k)$.

The simplified ESRCF and the simplifed ECKF use the same equations to transform the Kalman gain, and to make a state estimation and preditction. The architectures developed for the ECKF in Chapter 5 have been designed with the same design principles as the ESRCF. Hence, in this chpater, we have developed architectures for only uncommon equations and have used architectures developed in Chapter 5 for the common equations to both filters.

The uncommon equations, Equations (6.1), (6.2), and (6.3) require for each axis two QR decompositions and one division of a vector by a scalar number. We have found in Section 3.2.2 that the systolic array developed by H.T. Kung [15] is an ideal parallel implementation for the QR decomposition. This systolic array assumes that the input matrix enters it from the top row to the bottom row. We have found that the modifications on the functionalities of

processing elements allow the input matrix to enter the systolic array in the oppositie direction.

Based on these two types of systolic arrays, we have developed two architectures for Equations (6.1), (6.2), and (6.3). These two architectures consist of two QR systolic arrays for time and measurement updates, a number of delay elements to synchronize input data streams, and a number of multipliers for simplified matrix–matrix multiplications.

The architecture with H.T. Kung's systolic arrays requires two unidirectional buses to connect two systolic arrays, whereas the architecure with the modified systolic arrays requires only one bidirectional bus. The interconnection using two unidirectional buses is longer than that using one bidirectional bus, but the former interconnection is generally easier to control than the latter. Furthermore, the architecture with two unidirectional buses is found to require less clock cycles and more delay elements than the other architecture. Hence, the selection of an architecture should be based on design criteria such as the computational requirements, and implementational complexity.

We have shown that the use of the decoupling technique reduces the hardware and computational time requirements of the QR decomposition in the updates of the square root S(k) of the state estimate covariance matrix P(K) by a factor of approximately m, where m is the tracking dimension. This reduction is due to the separation of the updates of the coupled S(k) into the updates of m decoupled $S_o(k)$'s.

In the development of architectures for Equations (6.1), (6.2), and (6.3), the use of the sparse, band nature of the transition matrix $\phi(k)$ has simplified an implementation of a multiplication of $S_o^T(k)$ and $\phi^T(k)$ in Equation (6.1). The required number of clock cycles for this equation and that of processing elements

are reduced by an order of n { O(n) }. Similarly, the sparse nature of the measurement matrix $H_o(k)$ has reduced the number of elements to be nullified in the QR decomposition for Equation (6.2).

A comparison of the proposed architectures for a time update in this chapter with the Sung–Hu architecture confirms that the decoupling technique reduces the number of processing elements by a factor of m. However, for the measurement update, the proposed architecture and Sung–Hu architecture requires a comparable number of processing elements, because Sung–Hu's use of the sparse nature of an input matrix is as effective as the decoupling technique. For the time and measurement updates the proposed architecture requires overall m/2 times less number of processing elements than the Sung–Hu architecture.

A comparison of the computational time requirements between the proposed architecture and the Sung–Hu architecture shows that the former architecture requires 2.53 or 1.9 times less computational time than the latter for the updates of $S_o(k)$ and calculation of $K_o(k)$. This is due to the simplification by the combined use of the decoupling technique and special properties in the tracking KF. Although the application of the special properties does not contribute to a reduction in the number of processing elements, it decreases the computational time by simplifying matrix multiplications.

We have found that the proposed architecture for the ESRCF which is numerically superior to the ECKF requires slightly more arithmetic units than that for the ECKF. Processing elements in the architecture for the ESRCF are found to be more complex than those for the ECKF, and the actual length of a clock cycle in the implementation of the ESRCF is longer than that for the ECKF.

CHAPTER 7

CONCLUSIONS

The Kalman filter has been used in various applications such as communication, control, and target tracking. However, its high computational demand has limited its use to some extent. As digital integrated circuit technology becomes more accessible and cost effective, interest in the parallel implementation of the Kalman filter will continue to increase.

In this thesis, we have developed the first parallel implementations of the extended covariance Kalman filter and extended square root covariance filter for radar tracking applications. The extended covariance KF expresses the accuracy of its state estimates in the form of the state estimate error covariance matrix P(k), whereas the extended SRCF expresses the accuracy in the form of the square root $P^{1/2}(k)$ of P(k). The extended SRCF has better numerical properties than the extended covariance KF, but the former filter is more computationally demanding than the latter.

We have utilized the concepts of pipelining and parallel processing to minimize computational time and hardware requirements with the consideration of desirable properties of parallel architectures such as regularity, modularity, and local interconnection. When the pipeline processing architecture has a significant delay and requires a large number of delay elements, the parallel processing architecure with global interconnection has been employed.

We have made extensive use of a decoupling technique in designing the architectures of the ECKF and the ESRCF, after we have extended the use of the decoupling technique to the ESRCF.

The use of the decoupling technique reduces the computational requirements of the ECKF and those of the ESRCF in the following ways:

a) Eliminate the need for a matrix inversion in the ECKF.

b) Decouple the time and measurement updates of the state estimate error covariance matrix P(k) in the ECKF, and those of the square root $P^{1/2}(k)$ of P(k) in the ESRCF.

In the decoupled ECKF and ESRCF, the transformation of the Kalman gain matrix $K_o(k)$ from the line-of-sight (LOS) to reference Cartesian coordinates is required to compensate for the decoupling technique. We have found that the transformation of $K_o(k)$ requires a significantly small number of operations.

As a result of the decoupling technique, the number of operations for the time and measurement updates of P(k) in the ECKF is reduced by a factor of m^2, and the number of operations for $P^{1/2}(k)$ in the ESRCF is reduced by a factor of between m^2 and m, where m denotes the tracking dimension. The reason for these different reduction ratios is that the decoupling technique reduces an order of n^3 operations by a factor of m^2 and an order of n^2 operations by a factor of m. The letter n denotes the number of state elements in the state vector.

In the development of a parallel architecture, parallelism is exploited to reduce computational time requirements by allowing more than one operations to be performed in parallel. In the decoupled KF, the updates of m decoupled

$P_o(k)$'s and those of m $P_o^{1/2}(k)$'s may be performed for each axis in parallel. The use of the decoupling technique not only reduces hardware requirements but also computational time requirements.

We have shown that the parallel implementation of the updates of m decoupled $P_o(k)$'s require approximately m times less processing elements and clock cycles than that of the updates of one coupled P(k). The product of the reduction ratio for the number of processsing elements and that for the clock cycles is the same as the reduction ratio for the number of operations by the decoupling technique. This means that in the development of a parallel architecture the computational reduction ratio of m^2 is converted into the hardware requirement reduction ratio of m and the computational time reduction ratio of m.

Simlarly, we have found that in the parallel implementation of the updates of $P^{1/2}(k)$ the decoupling technique results in the reduction of the number of processing elements and that of the number of clock cycles by a factor of approximately m. The product of the reduction ratio for the number of processing elements and that for the number of clock cycles is m^2. This product is greater than the reduction ratio for the number of operations which has been found to be between m^2 and m. This can be explained by the fact that the number of only internal processing elements, not that of computationally intensive boundary processing elements, is reduced by the decoupling techinque.

We have found that the transformation of the Kalman gain, which accounts for the decoupling of the state estimate error covariance matrix P(k), is easy to implement.

The performance of the decoupled extended KF is generally comparable to that of the coupled extended KF in typical tracking situations, unless the

condition for the decoupling technique does not hold well. The required condition is that the orientation of a LOS frame does not vary significantly between filtering instants. We have shown that the performance of the decoupled ECKF is generally comparable to that of the conventional ECKF in two typical tracking examples of the Precise Radar Navigation system. Furthermore, we have found that the decoupled ECKF is more robust to errors in target modeling than the standard ECKF in the Precise Radar Navigation system, since its decoupled nature reduces the propagation of the effects of inaccurate modeling.

We have utilized properties of matrices in the tracking KF to reduce computational requirements and to simplify implementations. The sparse nature of the measurement matrix H(k) simplifies matrix–vector operations. For the extended covariance KF, the overall reduction ratio for the required number of multiplications and that of additions by the use of both the decoupling technique and special properties are found to be 24 and 13 respectively, for 3–dimensional tracking with 9 state elements. For the extended SRCF, the overall reduction ratios are 8.4 and 7.5 for the number of multiplications and additions respectively. One of the reasons why the reduction ratios for the ECKF are greater than those for the ESRCF is that the decoupling technique reduces the number of operations for the updates of P(k) in the ECKF by a factor of m^2 and that for the ESRCF by a factor of between m^2 and m. Another reason is that in the ECKF the computationally intensive matrix inversion is eliminated by the decoupling technique, whereas in the ESRCF the matrix inversion is already avoided by the use of forward substitution in the coupled ESRCF. The simplified matrix–vector operation by the use of the sparse nature of $H_0(k)$ is much easier to implement than a standard matrix–vector operation.

The use of the sparse, band nature of the transition matrix e(k) simplifies

matrix–matrix operations. The multiplication of two n–by–n matrices usually takes an order of n steps on an architecture with n^2 multipliers and n^2–n delay elements. However, the matrix–matrix multiplication, $\phi(k)P(k)\phi^T(k)$, in the ECKF has been implemented using the sparse, band nature of $\phi(k)$ without any additional hardware except for a bidirectional bus connecting processing elements. The implementation of the multiplication, $(P_o^{1/2}(k))^T\phi^T(k)$, in the ESRCF using the sparse, band nature of $\phi(k)$ requires an order of n multipliers and an order of n^2 delay elements, which are still much less than an order of m^2 multipliers and delay elements.

The parallel implementation of the ECKF and that of the ESRCF, developed in this thesis, consist of three processors:

a) Processor for the LOS coordinates.

b) Processor for the coordinate transformation.

c) Processor for the reference Cartesian coordinates.

Both the parallel architecture for the ECKF and that for the ESRCF use the same processors for the coordinate transformation and the reference Cartesian coordinates. In the implementation of the ECKF, the processor for the LOS frame for each axis consists of a (n/m)–by–(n/m) systolic array, and a number of mulipliers and delay elements, whereas in the implementation of the ESRCF the processor for the LOS frame consists of two QR systolic arrays, multipliers, and delay elements. The reason for using a 2–dimensional systolic array for the ECKF and two QR systolic arrays for the ESRCF is that for time and measurement updates the ECKF requires a number of matrix–matrix multiplications, whereas the ESRCF requires two QR decompositions.

We have developed two different architectures for the updates of $P_o^{1/2}(k)$ in the ESRCF. One of them is based on the QR systolic array developed by H.T. Kung [14]. This systolic array assumes that the input matrix enters the systolic array in an order from the top row to the bottom. Another architectur is based on the modified QR systolic array which assumes the input matrix to enter the systolic array in the opposite direction.

The architecture with H.T. Kung's systolic arrays requires two unidirectional buses, whereas the architecture with the modified systolic arrays requires only one bidirectional bus. The interconnection using two unidirectional buses is longer than that using one bidirectional bus, but the former interconnection is generally easier to control than the latter. Furthermore, the architecure with two unidirectional buses is found to require less clock cycles and more delay elements than the other architecture. The selection of an architecture should be based on design criteria such as the computational requirements, and implementational complexity.

The parallel implementation of the ECKF, developed in this thesis, requires $(n^2/m)+n+m$ processing elements, $2nm+2n+1$ mulitipliers, m adders, $2m-2$ CORDIC's, and $(n^2/m)-m$ delay elements. Specifically, for 3-dimensional tracking with 9 state elements, the hardware requirements are 39 processing elements, 73 multipliers, 3 adders, 4 CORDIC's, and 18 delay elements; the computational time requirement for each iteration is 16 clock cycles.

The proposed parallel implementation of the ESRCF, using H.T. Kung's systolic arrays, requires $(n^2/m)+2n+m$ processing elements, $2nm+n+1$ multipliers, $n+m-1$ dividers, $n+m$ adders, $2m-2$ CORDIC's, $n+m$ multiplexers, and n^2/m delay elements. For 3-dimensional tracking with 9 state elements, the requirements are 48 processing elements, 64 multipliers, 11 dividers, 12 adders, 4 CORDIC's, 12

multiplexers, 27 delay elements; this architecture takes 15 clock cycles to implement one iteration.

The parallel implementation of the ESRCF, using the modified systolic arrays, requires the same number of arithmetic units except for delay elements. It requires $(n^2/2m)+(n/2)$ delay elements, which is 18 elements for 3-dimensional tracking with 9 state elements. This architecture takes 20 clock cycles to implement one iteration.

A comparison of the proposed parallel implementation of the ECKF and those of the ESRCF shows that the former requires slightly less arithmetic units than the latter. Processing elements in the architecture for the ESRCF, especially boundary processing elements, are found to be more complex than those for the ECKF. The architecture for the ECKF and those of the ESRCF take a comparable number of clock cycles for an iteration of the KF. However, one clock cycle in the implementation of the ESRCF is longer than that for the ECKF. The selection of an implementation should be based on design criteria. The ESRCF which is numerically superior to the ECKF is implementationally more complex than the ECKF.

A comparsion of the proposed architecture for the ECKF and the Papadourakis-Taylor architecture [36] confirms that the decoupling technique reduces the number of processing elements for the time and measurement updates of P(k) by a factor of m. The proposed architecture requires greater than n times less computational cycles for each iteration than the Papadourakis-Taylor architecture. This is due to the combinded use of the decoupling technique and special properties in the tracking KF.

A comparison of the proposed architecture for the ESRCF and that of Sung-Hu [47] shows that the proposed architecture for the time update of $P^{1/2}(k)$

requires m times less number of processing elements than the Sung-Hu architecture. This is due to the simplification by the decoupling technique. However, for the measurement update, they require a comparable number of processing elements. This means that Sung-Hu's use of the sparse nature of an input matrix is as effective as the use of the decoupling technique. For 3-dimensional tracking with 9 state elements, the proposed architecture requires 2.53 or 1.53 times less computational time than Sung-Hu architecture for the updates of $S_o(k)$.

APPENDIX A

Cordinate Rotation Digital Computer

In this appendix, we discuss the Coordinate Rotation Digital Computer (CORDIC) algorithm. It was developed in 1956 by Volder to perform coordinate transformations and to compute trigonometric functions [49]. The CORDIC algorithm is based on an iterative scheme for vector rotations.

A vector X_i, $(x_i,y_i)^T$, having polar coordinates (r_i,θ_i) may be rotated to a new vector X_{i+1}, $(x_{i+1},y_{i+1})^T$, and scaled in magnitude as follows:

$$X_{i+1} = \begin{bmatrix} 1 & -\sigma_i\delta_i \\ \sigma_i\delta_i & 1 \end{bmatrix} X_i \qquad (A.1)$$

where the sign σ_i is either 1 or −1 based on the direction of rotation and δ_i is a decreasing sequence of arbitrary positive constants representing the amount of rotation at each iteration. The angular and radial components of X_{i+1}, θ_{i+1} and r_{i+1}, are related to the angular and radial components of X_i, θ_i and r_i, as follows:

$$\theta_{i+1} = \theta_i + \alpha_i \qquad (A.2)$$

$$r_{i+1} = r_i * k_i \qquad (A.3)$$

where

$$\alpha_i = \tan^{-1}\delta_i \tag{A.4}$$

$$k_i = (1 + \delta_i^2)^{1/2} \tag{A.5}$$

The n rotations of a vector X_o with angular and radial components θ_o and r_o result in a vector X_n with θ_n and r_n defined as follows:

$$\theta_n = \theta_o + \alpha \tag{A.6}$$

$$r_n = r_o * k \tag{A.7}$$

where α is a total rotation angle

$$\alpha = \sum_{i=0}^{n-1} \sigma_i\alpha_i = \sum_{i=0}^{n-1} \sigma_i \tan^{-1}(\delta_i) \tag{A.8}$$

and k is a known scale constant

$$K = \prod_{i=0}^{n-1} k_i = \prod_{i=0}^{n-1} (1 + \delta_i^2)^{1/2} \tag{A.9}$$

We introduce an auxiliary variable z and update it at each iteration as follows:

$$z_{i+1} = z_i + \sigma_i\alpha_i \tag{A.10}$$

After n iterations, z accumulates the sum of rotation angles:

$$z_n = z_o + \alpha \qquad \text{(A.11)}$$

To explain how the CORDIC alogrithm works, we now describe how to evaluate $\tan^{-1}(-y_o/x_o)$. The computation of $\tan^{-1}(-y_o/x_o)$ is needed in a coordinate transformation, as explained in Section 4.3. The value of $\tan^{-1}(-y_o/x_o)$ is the negative of a bearing angle of (x_o,y_o) in the polar coordinates. Hence, we can calculate the value of $\tan^{-1}(-y_o/x_o)$ by finding the amount of rotation necessary for (x_o,y_o) to lie on the y-axis.

The initial vector X_o, (x_o,y_o), is rotated through a sequence of angles α_i until $X_n = (x_n,0)$. The direction of rotation is decided at each step i such that (x_{i+1},y_{i+1}) is closer to the y-axis than (x_i,y_i). At the end of n rotations, since $z_o = 0$ and z_n accumulates the sum of rotation angles, z_n stores the value of $\tan^{-1}(-y_o/x_o)$,

The processing of Equation (A.1) seems to require mulitiplications. However, if δ_i is defined as the integral power of 2, the multiplication in Equation (A.1) can be replaced by a shift operation which is much simpler than a multiplication. The CORDIC algorithm with δ_i defined as 2^{-i+1} requires only additions and shift operations to perform coordinate transformations and trigonometric functions.

APPENDIX B

PRAN System Simulation Program

In this appendix, we describe a software package simulating the Precise Radar Aided Navigation (PRAN) system. In the PRAN system, a ship rotates its radar beam continuously at a constant speed as it moves along its path. The radar measures the location of a reflector in terms of range and bearing angle upon the detection of the reflector. These measurements are used to estimate the position of the ship with the a priori knowledge of the position of the reflector. The PRAN simulation software simulates the ship's movement along the path and the radar's scan of a beam. When the reflector is located under the radar's beam, the program reports the detection of the reflector and generates a set of mesurements in terms of the range and bearing angle of the reflector.

The PRAN system simulation software is designed to be flexible to handle various configurations of reflectors and a multitude of ship's dynamics. It receives as input the PRAN system configuration parameters such as the location of reflectors, a ship's path, and information on the ship's dynamics. This simulation package may be used in defining system parameters for the PRAN system such as the optimal location of reflectors and in evaluating the performance of tracking filters.

In Section B.1, we first describe how the ship's motion and the scan of a beam are simulated, and then we present the way to reduce the number of

sweeps of a beam. In Section B.2, we summarize the simulation program in the form of pseudo code.

B.1 Simulation Methods

B.1.1 Ship's motion

There are generally two types of ship's path: straight line and turn. In a straight line path, a ship either moves at a speed which fluctuates around the predefined constant speed or accelerates with randomness around the predefined constant acceleration.

The ship's motion along the straight line is simulated using the following mathematical models:

a) for a ship moving at a constant speed with fluctuation

$$X(k) = \begin{bmatrix} x(k) \\ \dot{x}(k) \\ \ddot{x}(k) \end{bmatrix} = \begin{bmatrix} 1 & T & T^2/2 \\ 0 & 1 & T \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x(k-1) \\ \dot{x}(k-1) \\ \ddot{x}(k-1) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u(k-1)$$

where

$$X(k) = \begin{bmatrix} x(k) \\ \dot{x}(k) \\ \ddot{x}(k) \end{bmatrix} = \begin{bmatrix} \text{position at time k} \\ \text{velocity at time k} \\ \text{acceleration at time k} \end{bmatrix}$$

T is a state update period, and u(k–1) is a white Gaussian noise sequence modelling the fluctuation of velocity.

b) for a ship moving at a constant acceleration with fluctuation

$$X(k) = \begin{bmatrix} x(k) \\ \dot{x}(k) \\ \ddot{x}(k) \end{bmatrix} = \begin{bmatrix} 1 & T & T^2/2 \\ 0 & 1 & T \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x(k-1) \\ \dot{x}(k-1) \\ \ddot{x}(k-1) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u(k-1)$$

where u(k–1) represents the fluctuation of acceleration.

Similarly, the turning motion may be modelled as follows:

$$x(k) = R \cos \dot{\theta}t$$

$$y(k) = R \sin \dot{\theta}t$$

where x(k) and y(k) denote the x– and y–positions in the frame of reference whose origin is at the center of turn, R denotes the turning radius, and $\dot{\theta}$ denotes a turning speed.

B.1.2 Representation of a beam pattern

In this section, we study the representation of a beam pattern and the simulation of the rotation of a beam. The beam used in the PRAN system is a fan beam. We assume that the elavation beamwidth of the beam used in the PRAN system is 180°. Although the beamwidth cannot be infinitesmal in reality, we assume that the beamwidth is infinitesmal for simplicity in representing a beam pattern. We will explain in the next section how to account for this assumption in simulation.

Figure B.1 shows a beam in the ship's frame of reference whose z–axis is defined as the vertical axis going through the center of a ship , the x–axis is defined as the line connecting from the stern of the ship to bow, and the y–axis is the line going through the starboard to port. This beam may be represented by two vectors, V_1 and V_2, as shown in Figure B.1.

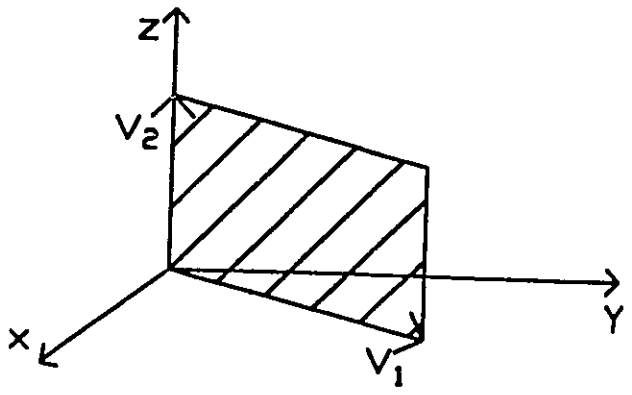


Figure B.1 Representation of a bean in the ship's frame of reference

The scan of a beam may be implemented in simulation by the rotation of the two vectors, V_1 and V_2, about the z-axis in the ship's frame of reference. Simliarly, the effects of pitching and rolling are represented by the rotation of a beam about the y- and x- axes respectively.

B.1.3 Estimation of detection time

It would be time consuming if we simulate all the sweeps and check at each sweep whether the reflector is located. Instead, if we can exactly predict a detection time, we would need to simulate a sweep only when the reflector is detected. However, it is not possible to predict an exact detection time because of the random nature of ship's motion and the effects of pitching and rolling.

Nevertheless, we have found the way to estimate the range of sweeps where the reflector is liekely located using the knowledge of ship's dynamics and radar characteristics. As a result, we need to look for a reflector only in the limited area. This eliminates the need to simulate all the unnecessary sweeps and consequently reduces the number of sweeps to be made.

There are usually more than one beam sweep at which the echo signal from a reflector has a significant amount of energy. The reason for this is that the beamwidth is not in reality infinitesmal. It is contrary to the assumption in the PRAN simulation program. The energy in each echo varies depending on the beam direction with respect to the reflector. However, the echo energy is maximal when the beam's boresight is closest to the reflector. For this reason, we define in the PRAN system simulation program a detection time as the instant at which the beam is closest to the reflector.

B.2 Pseudo Code

```
initialization
do i = 1 to total number of realizations
   initialize the position and velocity of the ship
   initialize the radar's viewing angle

   do j = 1 to total number of scans
        update the list of visible reflectors

        do k = 1 to number of visible reflectors
             detected = false

             while detected = false
             /* look for a reflector in the limited area */
                  update the position and velocity of the ship
                  generate the beam pattern
                  check if reflector is detected

                  if detected then
                       generate measurements (range, bearing angle)
                       detected = true

             update the position and velocity of the ship
             update the radar's viewing angle for the next reflector
```

REFERENCES

[1] Anderson, B. D.; B. M. Moore (1979) Optimal Filtering, Prentice-Hall, Englewood Cliffs, N.J.

[2] Baheti, R. (1986) "Efficient Approximation of Kalman Filter for Target Tracking", IEEE T-AES, vol. AES-22, no. 1, pp. 8-14.

[3] Benedict, T. R.; G. W. Bordner (1962) "Synthesis of an Optimal Set of Radar Track-While-Scan Smoothing Equations", IRE T-AC, July, 1962, pp. 27-32

[4] Bierman, G. (1977) Factorization Methods for Discrete Sequential Estimation, Academic Press, New York

[5] Brown, R. G. (1983) Introduction to Random Signal Analysis and Kalman Filtering, Wiley, 1983

[6] Chen, M. J.; K. Yao (1986) "On realization of least-squares estimation and Kalman filtering", Proc. 1st. Interanational workshop on systolic arrays, Oxford, pp. 161-170

[7] Cho, R.; S. Haykin; T. Greenlay (1986) "Polarimetric Radar For Precise Navigation: Study of an Experimental System", CRL Report No. TP7632E, McMaster University, Hamilton, Ontario

[8] Daum, F. E.; R. J. Fitzgerald (1983) "Decoupled Kalman Filters for Phased Array Radar Tracking", IEEE T-AC, vol. AC-28, pp. 269-283

[9] Fadeeva, D. K.; V. N. Fadeeva (1963) "Computational Methods for Linear Algebra", W. H. Freeman & Co., San Francisco

[10] Farina, A.; F. A. Studer (1984) Radar Data Processing, Research Studies Press LTD.

[11] Fisher, J. L.; D. P. Casasent; C. P. Neuman (1985) "A Factorized Extended Kalman Filter", Proc. SPIE 85, Real–Time Signal Processing, pp. 119–130

[12] Fitzgerald, R. J. (1971) "Divergence of the Kalman Filter", IEEE T–AC, vol. AC–16, pp. 736–747

[13] Gaston, F. M. F.; G. W. Irwin (1988) "A Systolic Squre Root Information Kalman Filter", Proc. IEEE International Conference on Systolic Arrays, 1988, pp. 643–652

[14] Gaston, F. M. F.; G. W. Irwin (1989) "VLSI architecture for square root covariance Kalman filtering", Proc. SPIE 89, Real–Time Signal Processing

[15] Gentleman, W. M.; H. T. Kung (1981) "Matrix triangularization by systolic arrays", Proc. SPIE 81, Real–Time Signal Processing, pp. 19–26

[16] Givens, W. (1958) "Computation of plane unitary rotations transforming a general matrix to triangular form", J. Soc. Industr. Appl. Math, vol. 6, pp. 26–50

[17] Golub, G. H.; C. F. Van Loan (1983) Matrix Computations, The John Hopkins University Press, Baltimore, MD

[18] Haykin, S. (1986) Adaptive Filter Theory, Prentice–Hall, Englewood Cliffs, NJ

[19] Haykin, S. (1989) [illegible] Filters, Macmillan Publising Co. New York, New York

[20] Hwang, K. (1984) Computer Architecture and Parallel Processing, McGraw–Hill, New York, New York

[21] Jover, J. M.; T. Kailath (1984) "A Parallel Architecture for Kalman Filter Measurement Update", IFAC 84 Hungary, pp. 1005–1009

[22] Kadela, T. F. ; Graham, J. H. (1984) "VLSI Architectures for optimal estimation", IEEE International Conference on Computer Design, 1984, pp. 364–369

[23] Kalman, R. E. (1960) "A new approach to linear filtering and prediction problems", Trans. ASME, (J. Basic Eng), vol. 82, pp. 35–50

[24] Kalman, R. E.; R. S. Bucy (1961) "New results in linear filtering and prediction problems", Trans. ASME (J. Basic Eng), vol. 83, pp. 95–108

[25] Kaminski, P. G.; A. E. Bryson; S. F. Schmidt (1971) "Discrete Square Root Filtering: A Survey of Current Techniques", IEEE T–AC, vol. AC–16, pp. 727–735

[26] Kung, H. T. (1982) "Why Systolic Architectures?", IEEE Computer. vol. 15, pp. 37–46

[27] Kung, H. T.; P. L. Lehman (1980) "Systolic (VLSI) Arrays for Relational Database Operations", Proc. ACM–Sigmod 1980 International Conference Management of Data, May. 1980, pp. 105–116

[28] Kung, H. T.; C. E. Leiserson (1978) "Systolic arrays (for VLSI)", Proc. Symposium on Sparse Matrix Computations and Their Applications, 1978, pp. 256–282

[29] Kung, S. Y. (1985) "VLSI Array Processors", IEEE ASSP magazine, vol. 2, no. 3, pp 4–22

[30] Kung, S. Y. (1988) VLSI Array Processors, Prentice Hall, New Jersey

[31] Kung, S. Y.; J. N. Hwang (1988) "An efficient Triarray Systolic Design for Real–Time Kalman Filtering", ICSASSP, 1988

[32] Lee, E. K. B.; S. Haykin (1988) "Parallel implementation of the tracking KF", ICASSP, 1989

[33] Macikunas, A.; S. Haykin; T. Greenlay (1984) "Trihedral Radar Reflector", Can. Patent applied for Nov. 22, 1984, U.S. patent applied for 1985, Oakville, Ontario

[34] Mead, C.; L. Conway (1980) Introduction to VLSI Systems, Addison-Wesley, Mass.

[35] Ohmuro, Takashi (1984) "A Decoupled Kalman Tracker using LOS Coordinates", Proc. 1984 International Symposium on Noise and Clutter Rejection in Radars and Imaging Sensors", 1984, pp. 451-455

[36] Paige C. C.; M. A. Saunders (1977) "Least Squares Estimation of Discrete Linear Dynamic Systems Using Orthogonal Transformations", SIAM J. Numer. Anal. vol. 14, no. 2, pp 180-193

[37] Papadourakis, G.; F. Taylor (1987) "Implementation of Kalman Filters using Systolic arrays", Proc. ICASSP 87, pp. 783-786

[38] Schlee, F. H.; C. J. Standish, N. F. Toda (1967) "Divergence in the Kalman Filter", AIAA J. vol. 5, pp. 1114-1120

[39] Schooler, C. C. (1975) "Optimal α-β Filters for Systems with Modeling Inaccuracies", IEEE T-AES, vol. AES-11, no. 6, pp. 1300-1306

[40] Sedra, A. S.; P. O. Bracket (1978) Filter Theory and Design: Active and Passive, Matrix Publishers Inc, Champaign, Illinois

[41] Singer, R. A. (1970) "Estimating Optimal Tracking Filter Performance for Manned Maneuvering Targets", IEEE T-AES, vol. AES-6, no. 4, pp. 473-483

[42] Singer, R. A.; K. W. Behnke (1971) "Real-Time Tracking Filter

Evaluation and Selection for Tracking Applications", IEEE T-AES, vol. AES-7, no. 1, pp. 100-110

[43] Skolnik, M. I. (1982) Introduction to Radar Systems, McGraw-Hill, New York

[44] Sorenson, H. W. (1985) Kalman filtering: Theory and Application, IEEE press, New York

[45] Stone, H. S. (1980) Introduction to Computer Architecture, Science Research Associates Inc.

[46] Storer, J. E. (1957) Passive Network Synthesis, McGraw-Hill Book Co. New York, New York

[47] Strang, G. (1980) Linear Algebra and Its Applications, second edition, Academic Press, New York

[48] Sung, T.; Y. Hu (1987) "Parallel VLSI Implementation of The Kalman Filter", IEEE T-AES, vol. AES-23, no. 2, pp. 215-224

[49] Volder, E.J. (1959) "The CORDIC Trigonometric Computinting Technique", IRE Trans. on Electronic Computers

[50] Walther, J. S. (1971) "A unified algorithm for elementary functions", Proc. AFIPS conference 1971

[51] White, D. M. (1976) "Simulation of Automatic Detection and Track-While-Scan Systems for Unstabilized Shipboard Radars", Ph.D Thesis, George Washington University, Washington, D.C.

[52] Yeh, H (1988) "Systolic Implementation on Kalman Filters", IEEE T-ASSP, vol. ASSP-36, no. 9, pp. 1514-1517