Numerical Implementation of Minimum-Energy Filters

Author: Conor Horgan

Supervisors: Jochen Trumpf and Mohammad Zamani

Course: ENGN2706 Engineering Research and Development Project (Methods)

Submission: 16th November 2012

Acknowledgements

I would like to thank Jochen Trumpf and Mohammad Zamani for their continued support and guidance throughout the duration of this project.

Abstract

This report is concerned with the numerical implementation of a recently proposed attitude estimation filter, the Geometric Approximate Minimum Energy (GAME) filter and the evaluation of its performance relative to the Multiplicative Extended Kalman filter (MEKF), which is the industry standard attitude estimation filter. Evaluation and comparison of the two filtering systems consisted of a MATLAB code suite which evaluated the performance of each filter for the same parameters. The solutions of the gain equations for each filter were determined using two numerical algorithms: Choi's method and a simple Euler method. Due to numerical difficulties and time constraints, the GAME filter could not be implemented using Choi's method. Instead, a close approximation to the GAME filter, the SO3 filter, was implemented using Choi's method. The results suggest the superior performance of the GAME filter at high noise levels, with further research required to determine the performance at lower noise levels.

Table of Contents

I. Glossary and Notation

GAME Filter = Geometric Approximate Minimum Energy Filter

EKF = Extended Kalman Filter

MEKF = Multiplicative Extended Kalman Filter

RDE = Riccati Differential Equation

ODE = Ordinary Differential Equation

ARE = Algebraic Riccati Equation

BDF = Backwards Differentiation Formulas

UAV = Unmanned Aerial Vehicle

Backwards Difference Operator

$\nabla X(t_i)$ denotes the backward difference defined below for the base and recursive cases.

$$\nabla X(t_i) = X(t_i) - X(t_{i-1})$$
$$\nabla^j X(t_i) = \nabla\left(\nabla^{j-1} X(t_i)\right)$$

Lower Index Operator

For $\Omega = [a,b,c]^T \in \mathbb{R}^3$, the lower index operator $(.)_\times : \mathbb{R}^3 \to SO(3)$ yields the skew symmetric matrix

$$\Omega_\times = \begin{bmatrix} 0 & -c & b \\ c & 0 & -a \\ -b & a & 0 \end{bmatrix}$$

Symmetric Projector

The symmetric projector $\mathbb{P}_s$ is defined by

$$\mathbb{P}_s(M) := \frac{1}{2}\left(M + M^T\right)$$

II. Aims and Contributions

This report is concerned with the numerical integration of minimum energy filters and their implementation and simulation in MATLAB. In particular, the gain equation of a newly developed filter, the geometric approximate minimum-energy (GAME) filter [1], is modelled as a particular stiff Riccati differential matrix equation (RDE). By using a novel method developed by Chiu H. Choi [2] to iteratively determine the solutions of RDEs, the gain equation of the GAME filter is solved numerically.

In order to determine the performance of the GAME filter relative to other filters used in industry applications, the process of solving the GAME filter gain equation is implemented in MATLAB. This involves extending a previously developed MATLAB code suite which simulates the GAME filtering system such that the solution of the filter's gain equation can be numerically determined. In order to develop an informative simulation process, two different filters are simulated and each is implemented using two different methods of solution for the gain equation. In this way, the relative performance of each filter and method of implementation can be compared and evaluated.

The two filters simulated in this project are the GAME filter and the multiplicative extended Kalman filter (MEKF) [3]. The GAME filter, as the focus of this project is a newly developed filter which is expected to show high performance results. In order to assess the performance of this filter for given situations, it is compared to the MEKF, the industry standard for filtering systems used in spacecraft and satellite applications [1]. It is expected that the results of these simulations will show the relatively greater performance of the GAME filter in terms of its error convergence at a range of noise levels and system parameters.

For further and more comprehensive analysis, each of the two filtering systems is implemented using two different methods. The first method is a simple Euler method, a basic numerical integration technique [4]. The second method is Choi's method, which is another numerical integration technique developed for this particular type of Riccati equation [2].

Following the implementation of these two filtering systems with each of the two gain equation solution algorithms, it was determined that the GAME filter outperformed the MEKF in terms of error convergence for a given set of simulations. This confirmed expectations of the GAME filter's superiority and presents a strong and valid argument for

the future use of the GAME filter in unmanned aerial vehicle (UAV) applications. This report allows an extended assessment of the performance of the GAME filtering system for UAV applications, paving the way for more substantial research in this field.

In the near future, the author of this report and the developers of the GAME filtering system, hope to extend this code suite in Python for the comparison of a greater number of filtering systems with a standardized set of variables and noise levels. It is expected that this suite will provide an excellent tool for the analysis and evaluation of filtering systems yet to be developed.

III. Introduction

Attitude estimation is the process of estimating the orientation of an object or vehicle in three dimensional space based on measurements such as remote observations of celestial bodies or reference points. Attitude estimation is important in the context of the control of vehicles such as spacecraft, satellites and unmanned aerial vehicles (UAV) as it is required in order for onboard computers to determine their optimal trajectory. Attitude estimation involves the use of sensors such as gyroscopes and magnetometers to measure data which is then used to determine the attitude of a system. The signals obtained from these sensors invariably contain noise and must be filtered such that the true signal can be determined. As such, the development of filtering systems for the purposes of attitude estimation has been an integral step in the advancement of aerial vehicles.

The field of filtering for attitude estimation is a complex and diverse field which has evolved with ever-developing technology and applications. The algorithms that define attitude filtering systems such as the Extended Kalman Filter (EKF) [5,6] are used to model and predict complex systems in three dimensional space and as such are difficult to solve. Due to the complexity of these equations, analytical solution cannot be found. In order to solve these filtering problems, various forms of numerical integration must be employed to determine accurate solutions to the equations that define the filtering systems.

This report details the numerical implementation of Choi's solution [2] to the stiff Riccati differential equation in the context of the gain equation of the Geometric Approximate Minimum Energy (GAME) filter, a recently developed near-optimal filtering system. The solution of the GAME filter is then simulated using Choi's method and compared against a

similar solution derived using a simple Euler method. The GAME filter is then evaluated against the MEKF, the industry standard filter, which is solved using Choi's method and Euler's method, respectively, as well as the same initial state variables and noise levels.

The report is organized as follows. Section I introduces the project specific terms and notation used. Section II highlights the aims of the report as well as the final result. Section III serves as an introduction that details the significance of filtering systems in the context of attitude estimation and underlines the purpose and reason for research into the GAME filtering system and its potential applications. Section IV serves as an introduction to the field of numerical integration and its development with respect to Riccati equations. In addition, section IV also serves to contextualize numerical integration with respect to attitude estimation filtering. Section V details the experimental procedure used for this research and presents the results of the research work and discusses their significance as well as inherent limitations and flaws. Finally, in section VI, the conclusion of the report is given, highlighting the key contributions of the research presented in this report and outlining the future research that follows on from these discoveries.

IV. Literature Survey and Theoretical Background

The field of numerical integration of ordinary differential equations (ODEs) has arisen due to the fact that many such equations cannot be solved analytically and instead require that the solutions be estimated to a high degree of accuracy [7]. These sorts of ordinary differential equations arise in many different fields when various mathematical modelling techniques are employed to describe various phenomena arising in fields such as chemistry, physics, economics, engineering and biology [4]. Several methods have evolved for solving ODEs, with applications to different sorts of ODEs consisting of initial value problems (IVPs) and boundary value problems (BVPs) [7].

This literature survey will consider some of the general solution algorithms for ODEs and describe the basic principles in order to contextualize the concept of numerical integration. Following this, methods for the solution of Riccati differential equations (RDEs) will be evaluated and compared, leading to an explanation of the chosen algorithm used in

determining the solution to the gain equation of the GAME filter with which this report is concerned.

Numerical Integration of ODEs

The simplest methods for the numerical integration of ODEs are a family of methods known as the Euler methods. [8] These consist of the forward Euler method and the backward Euler method. Euler methods are used to develop a solution to the general differential equation

$$y'(t) = f(t, y(t)) \qquad y(t_0) = y_0$$

Where y is the function and t represents time.

The forward Euler method is a simple, explicit method, derived using a Taylor expansion at t_{n-1} [4], which uses successive tangent lines to approximate the true solution of a curve. By rearranging the finite difference approximation, where h is the time step

$$y'(t) \approx \frac{y(t+h) - y(t)}{h}$$

and applying the Taylor expansion in a recursive way, this results in the equation for the forward Euler method [7].

$$y_{n+1} = y_n + hf(x_n, y_n)$$

The implicit backward Euler method is similar to the explicit forward Euler method, except instead of applying the original finite difference approximation [7], one applies the analogous

$$y'(t) \approx \frac{y(t) - y(t-h)}{h}$$

and computes the Taylor expansion centred around t_n [4], the backward Euler method is derived.

$$y_{n+1} = y_n + hf\left(t_{n+1}, y_{n+1}\right)$$

Whilst conceptually simple, the Euler methods are not very accurate and hence unsuitable for use with many different ODEs [7]. As such several other methods of solving ODEs have been developed [4]. Of these, the most popular single-step method is the Runge-Kutta fourth-order method [9], which is a generalization of the previous Euler methods that replaces the slope

function f with the weighted slope average in the range $x_n \le x \le x_{n+1}$ [7]. The equation of the fourth-order Runge-Kutta method is

$$y_{n+1} = y_n + h\left(w_1 k_1 + w_2 k_2 + w_3 k_3 + w_4 k_4\right)$$

With the parameters

$$\begin{aligned} k_1 &= f\left(x_n, y_n\right) \\ k_2 &= f\left(x_n + \alpha_1 h, y_n + \beta_1 h k_1\right) \\ k_3 &= f\left(x_n + \alpha_2 h, y_n + \beta_2 h k_1 + \beta_3 h k_2\right) \\ k_4 &= f\left(x_n + \alpha_2 h, y_n + \beta_4 h k_1 + \beta_5 h k_2 + \beta_6 h k_3\right) \end{aligned}$$

such that the formula for the Runge-Kutta agrees with a Taylor polynomial of degree four. The most commonly used set of parameters yield the following result [7]:

$$\begin{aligned} y_{n+1} &= y_n + \frac{h}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right) \\ k_1 &= f\left(x_n, y_n\right) \\ k_2 &= f\left(x_n + 0.5h, y_n + 0.5h k_1\right) \\ k_3 &= f\left(x_n + 0.5h, y_n + 0.5h k_2\right) \\ k_4 &= f\left(x_n + h, y_n + h k_3\right) \end{aligned}$$

In addition to the usual single step methods such as the Euler methods and the Runge-Kutta method, multistep methods such as the Adams-Moulton method are used to solve ODEs. These methods use several previously calculated values of $y_n, y_{n-1}, \ldots$ to calculate the value of y_{n+1} and work to reduce the local truncation error associated with numerical integration [10].

While these numerical integration methods are useful in many different ODE problems, the magnitude of their truncation errors leads to poor performance when used with many types of ODEs, particular stiff ODEs. The Riccati differential equation (RDE), in particular, is a difficult to solve ODE, for which many relatively simple numerical integration techniques are unsuitable [11].

The general form of the RDE [2] is

$$\dot{X}(t) = Q(t) + X(t)A(t) + B(t)X(t) - X(t)R(t)X(t) \qquad X(t_0) = X_0 \qquad t_0 \le t \le T$$

Where $A(t) \in \mathbb{R}^{n\times n}$, $B(t) \in \mathbb{R}^{m\times m}$, $Q(t) \in \mathbb{R}^{m\times n}$, $R(t) \in \mathbb{R}^{n\times m}$, $X(t) \in \mathbb{R}^{m\times n}$ and $t \in \mathbb{R}$. This has been widely studied in the past 30 years [2,12,13,14,15,16,17,18,19,20] due to the numerous applications the equation has in fields such as optimal control and robust stabilization [11,15] which require numerically accurate and efficient algorithms for solving stiff RDEs. This extensive study of the solution of Riccati equations has led to five main classes of methods for solving such equations [21,2]. The five approaches considered here are vectorization, linearization, Chandrasekhar's method, superposition and matrix methods.

Vectorization

The first approach considered for the solution of differential Riccati equations is the vectorization approach which consists of unrolling the matrices in the RDE into vectors and then integrating the resulting system of n^2 differential equations [2,21]. This approach, whilst relatively simple when compared to other methods of solution, suffers from the fact that it requires a nonlinear system with n^2 unknowns to be solved at each time step. This means that if the Riccati equation is stiff, the cost of applying standard backwards differentiation formulas (BDF), both in terms of time and space, is very high [2]. As such, given the size, complexity and stiffness of the Riccati equation considered for the solution of the GAME filter, the vectorization approach is unsuitable for this application.

Linearization

The linearization class of methods is based on the transformation of a matrix quadratic equation into a system of linear first-order differential equations. One must consider the above differential Riccati equation with a different initial condition.

$$\dot{X}(t) = Q(t) + X(t)A(t) + B(t)X(t) - X(t)R(t)X(t) \qquad X(t_0) = V_0U_0^{-1}$$

The transformation of the quadratic RDE into a system of linear first-order matrix differential equations yields [13]

$$\frac{d}{dt}\begin{bmatrix} U(t) \\ V(t) \end{bmatrix} = \begin{bmatrix} -A(t) & R(t) \\ Q(t) & A(t)^T \end{bmatrix}\begin{bmatrix} U(t) \\ V(t) \end{bmatrix} \qquad t_0 \le t \le T$$
$$\begin{bmatrix} U(t_0) \\ V(t_0) \end{bmatrix} = \begin{bmatrix} U_0 \\ V_0 \end{bmatrix}$$

where $U(t) \in \mathbb{R}^{n\times n}$ and $V(t) \in \mathbb{R}^{n\times n}$ for some invertible $U_0 \in \mathbb{R}^{n\times n}$ and $V_0 \in \mathbb{R}^{n\times n}$. Given that the solution to the Riccati equation exists, then the solution to the above equation gives [13]

$$X(t) = V(t)U^{-1}(t) \qquad t_0 \le t \le T$$

While this solution method works well for some variations of the Riccati equations and several solution algorithms are based on this general form of solution, the algorithms do not handle the differential equations well when they are stiff. This results in algorithms that are not very accurate for stiff RDEs and hence this class of methods is unsuitable for applications to the GAME filter, for which the equation that defines the gain of the filter is stiff.

Chandrasekhar's Method

A third class of methods, of which the most well-known is the Chandrasekhar method, involves the transformation of RDEs into two coupled systems of nonlinear differential equations [2]. This class of methods is most applicable to time-invariant RDEs [21] and is most efficient when the number of controllers and observers is small [2]. The two coupled systems of equations that together make up the Chandrasekhar system are [14,21,2]

$$\dot{L} = \left(K^T G^T - A^T\right)L, \qquad L(0) = L_0 \in \mathbb{R}^{n\times l}$$
$$\dot{K} = -G^T L L^T, \qquad K(0) = G^T X_0 \in \mathbb{R}^{m\times n}$$

where $Q(t) \equiv CC^T, R(t) \equiv GG^T$. Through the construction of the Chandrasekhar system, the solution of the RDE can be obtained. While this method has been adapted for linear, time-varying distributed systems [22], the method is somewhat unstable due to issues relating to the numerical integration [21]. As such, this method too, is unsuitable for applications to the solution of the GAME filter gain equation.

Superposition Methods

In previous work by Harnard [23], a set of superposition principles governing matrix Riccati equations was derived. Based on these superposition principles, another class of methods for the solution of RDEs has been developed [18,20]. According to Choi, the general solution of

an RDE can be expressed as a nonlinear combination of at most five independent solutions [2]. In order for these methods to work, they require integration of the RDE several times with different initial conditions before the superposition formulas, which are themselves computationally complex, can be applied [21]. Given the high computational complexity associated with these methods, they too are unsuitable for application to the GAME filter.

Matrix Methods

Relatively recently, the class of matrix methods developed for the solution of differential Riccati equations has been studied [2,21,24]. These RDE solution methods use the standard ODE numerical algorithms detailed earlier in this paper and apply them to RDEs using matrix-valued algorithms [21]. Of these matrix based methods, the most relevant for application to the gain equation of the GAME filter is Choi's method. This solution algorithm employs implicit backward differentiation formulas (BDFs) to yield a solution to the RDE through the transformation of the RDE into an algebraic Riccati equation (ARE) which can be solved in each time step [2]. Such implicit methods are well-suited to solving stiff equations such as the GAME filter gain equation which is the focus of this paper. As Choi's method is the chosen algorithm for this particular research, its basis and structure will be detailed in the following paragraphs.

Choi's Method

As described previously, Choi's method [2] uses matrix-valued algorithms based on standard numerical algorithms for ODEs and applies them to RDEs. Choi considers the matrix Riccati differential equation of the form

$$\dot{X}(t) = Q(t) + X(t)A(t) + B(t)X(t) - X(t)R(t)X(t) \qquad X(t_0) = X_0$$

where $A(t) \in \mathbb{R}^{n\times n}$, $B(t) \in \mathbb{R}^{m\times m}$, $Q(t) \in \mathbb{R}^{m\times n}$, $R(t) \in \mathbb{R}^{n\times m}$, $X(t) \in \mathbb{R}^{m\times n}$ and $t \in \mathbb{R}$. By using the assumption that $t_0 \le t \le T$, Choi constructs a Newton-backward difference interpolating polynomial for a typical entry x_{ij}, of $X(t)$. In doing so, he derives another equation to approximate the original RDE.

$$\frac{1}{h}\sum_{v=1}^{r}\frac{1}{v}\nabla^{v}X_{k+1} = Q(t_{k+1}) + X_{k+1}A(t_{k+1}) + B(t_{k+1})X_{k+1} - X_{k+1}R(t_{k+1})X_{k+1}$$

$$X_i = X(t_i); \qquad \text{for } i = 0,1,\ldots,r; \qquad r < k+1$$

Using this equation, and expanding the finite difference operators, Choi develops the following algebraic Riccati equation (ARE).

$$\bar{Q}_{k+1,r} + X_{k+1}\bar{A}_{k+1,r} + \bar{B}_{k+1,r}X_{k+1} - X_{k+1}\bar{R}_{k+1,r}X_{k+1} = 0$$

where

$$\begin{aligned}
\bar{A}_{k+1,r} &= -\left(\frac{1}{2}\sum_{i=1}^{r}\frac{1}{i}\right)I_n + hA\left(t_{k+1}\right)\\
\bar{B}_{k+1,r} &= -\left(\frac{1}{2}\sum_{i=1}^{r}\frac{1}{i}\right)I_m + hB\left(t_{k+1}\right)\\
\bar{Q}_{k+1,r} &= \sum_{i=1}^{r}\frac{(-1)^{i-1}}{i}\binom{r}{i}X_{k-i+1} + hQ\left(t_{k+1}\right)\\
\bar{R}_{k+1,r} &= hR\left(t_{k+1}\right)
\end{aligned}$$

By considering the ARE in the given time interval, Choi then uses Newton's method to develop a good estimate of the unknown solution X_{k+1}. Choi shows that if at the *i*th iteration, the solution X_{k+1} is given by

$$X_{k+1} = X_{k+1,(i)} + \delta X_{k+1}$$

where $X_{k+1(i)}$ is the *i*th approximant to X_{k+1} and δX_{k+1} is the correction matrix, then the ARE can be rearranged by neglecting higher order terms and eliminating δX_{k+1} resulting in the equation.

$$X_{k+1,(i+1)}\left(\bar{A}_{k+1,r} - \bar{R}_{k+1,r}X_{k+1(i)}\right) + \left(\bar{B}_{k+1,r} - X_{k+1(i)}\bar{R}_{k+1,r}\right)X_{k+1,(i+1)} + \bar{Q}_{k+1,r} + X_{k+1(i)}\bar{R}_{k+1,r}X_{k+1(i)} = 0$$

This equation takes the form of the standard Sylvester equation $AX + XB + C = 0$. By solving this Sylvester equation at each time step, the new approximant $X_{k+1,(i+1)}$ is obtained and thus the original RDE is numerically solved. In order to iteratively solve this Sylvester equation at each time step, Choi employs an algorithm known as the Hessenburg-Schur method [25]. This algorithm is similar to the Bartels-Stewart algorithm [26], which was previously the algorithm of choice for Sylvester equations. The significant difference between these two algorithms is in the definition of the matrices used to compute transformations of the matrices *A*, *B* and *C*. This difference results in the Hessenburg-Schur method being between 30 and 70 percent faster than the Bartels-Stewart algorithm [25].

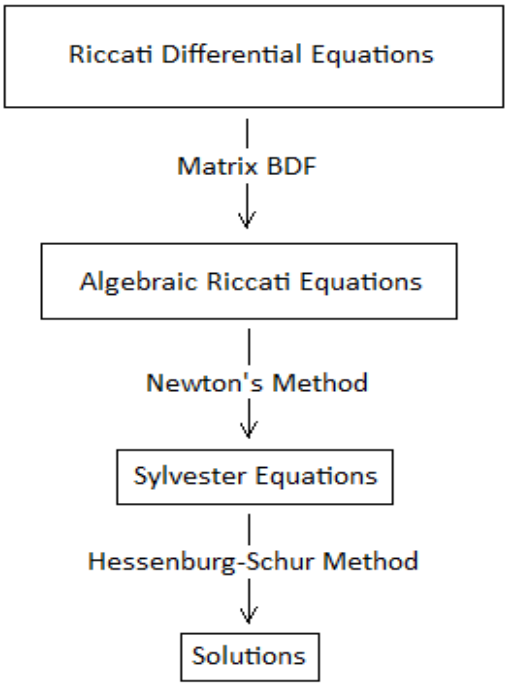

Figure 1- A Schematic of Choi's Algorithm for the solution of Riccati equations [2]

Choi's method represents a fast and efficient method of solving a Riccati differential equation. It is suitable for applications involving stiff Riccati equations and as such is a suitable algorithm to employ when solving the gain equation of the GAME filter. A schematic overview of Choi's method is shown in the figure 1.

The GAME Filter

In order to contextualize the development and solution of the Riccati differential equation, this section of the literature survey details the development of the GAME filter, and in particular, the gain equation that takes the form of a Riccati equation.

The GAME filter is a recently developed filter, having been derived in the last four years [27]. The filter has been developed in a deterministic framework and aims to have a high level of performance relative to the MEKF, the industry standard filter [1]. The GAME filter has been developed following significant recent work in the field of geometric non-linear observers [28,29,30,31] which present many advantages over the traditional EKF derived filters. The implementation of stochastic methods such as the EKF derivatives presents many difficulties because of their linearization approach which does not respect the SO(3) geometric structure of the state space, a problem not faced by the geometric observers [1,32].

The attitude kinematics and measurements for the GAME filter are as follows [27]

$$\dot{X}(t) = X(t)\Omega_{\times}(t), \qquad X(0) = X_0$$
$$u(t) = \Omega(t) + Bv(t)$$
$$y_i(t) = X(t)^T \dot{y}_i + D_i w_i(t), \qquad i = 1,...,n$$

Where X is the attitude rotation matrix and Ω is the angular velocity. The signal u denotes the body-fixed frame measured angular velocity input and the signal v denotes the input measurement error. The vectors $\dot{y}_i$ are known reference vectors with y_i as their measurements and w_i as the measurement errors. The matrices B and D are known coefficients for the measurement errors with their associated metrics $Q := BB^T$ and $R_i := D_i D_i^T$.

The equation that defines the GAME filter, as shown in [27], is given by

$$\dot{\hat{X}} = \hat{X}(u - Pl)_{\times}, \qquad \dot{\hat{X}}(0) = I$$

where the innovation term l is

$$l = \sum_i \left(R_i^{-1}\left(\hat{y}_i - y_i\right)\right) \times \hat{y}_i = 2\text{vex}\sum_i \mathbb{P}_a\left(\hat{y}_i\left(\hat{y}_i - y_i\right)^T R_i^{-1}\right)$$

and $\hat{y} := \hat{X}^T \dot{y}$.

The gain equation of the filter, the Riccati equation that is the basis of this report, is given by

$$\dot{P} = Q + \mathbb{P}_s\left(P\left(2u - Pl\right)_{\times}\right) - PSP + PAP$$
$$P(0) = \left(trace\left(K_0^{-1}\right)I - K_0^{-1}\right)^{-1}$$

The variables in this gain equation are defined as follows

$$S := \sum_i \left(\hat{y}_i\right)_{\times}^T R_i^{-1}\left(\hat{y}_i\right)_{\times}$$
$$A := trace\left(C\right)I - C$$
$$C := \sum_i \mathbb{P}_s\left(R_i^{-1}\left(\hat{y} - y_i\right)\hat{y}_i^T\right)$$

As previously stated, the gain equation of the GAME filter shown above is a Riccati differential equation. This therefore requires solution by a suitable algorithm, which for the purposes of this research, is Choi's method. The findings of this report discuss the implementation of this equation and its solution using Choi's method. The results are evaluated and compared to the MEKF gain equation implementation. This is detailed in the results and discussion section below.

V. Results and Discussion

The purpose of this report is to implement Choi's method of numerically solving a Riccati differential equation for the solution of the gain equation of the GAME filter and compare this to a solution of the GAME filter gain equation using a simple Euler method. In addition, to evaluate the performance of the GAME filter, the same implementations will be used to solve the gain equation of the MEKF and compare the performance of each filter in terms of their error convergence.

The gain equation of the GAME filter, as stated above, is given by

$$\dot{P} = Q + \mathbb{P}_s\left(P\left(2u - Pl\right)_\times\right) - PSP + PAP$$

The gain filter of the MEKF has not been formally discussed in this report but has been shown in [32] to be

$$\dot{P} = Q + P\frac{1}{2}u_\times - \frac{1}{2}u_\times P - PSP$$

with the same parameters as for the GAME filter. Both the gain equation for the GAME filter and the gain equation for the MEKF are Riccati differential equations of the form

$$\dot{X}(t) = Q(t) + X(t)A(t) + B(t)X(t) - X(t)R(t)X(t)$$

This allows both filter gain equations to be solved and compared using Choi's method through a MATLAB implementation.

MATLAB Implementation

In order to implement Choi's method for both the MEKF and the GAME filter using MATLAB, a code suite had to first be generated to simulate the measurements that are used by the filters as well as the noise levels of those measurements. An implementation was also required to define the filter equations which in turn rely on the gain equation to appropriately filter the measurement data. The initial implementation of this code suite was completed by Mohammad Zamani, a co-developer of the GAME filter [1,27].

For the purposes of this report, the code suite was extended by the author, to simulate each filter in an equivalent manner suitable for comparison and evaluation. The following section of the report details the important aspects of the implementation.

The MATLAB code suite inherited from previous work contained the variables and functions required to implement the GAME filter, solving the gain equation using the simple Euler method. The first part of extending this code required duplicating the initialization of variables, a variable for each filter type and for each separate implementation. This resulted in four variable names for each parameter named as follows.

```
q_B_SO3;
q_B_MEKF;
q_C_SO3;
q_C_MEKF;
```

The *'B'* name variable refers to the implementation of the particular filter using the Euler method whereas the *'C'* name variable refers to the implementation using Choi's method of implementation for the gain equation. The *SO3* parameter indicates the GAME filter (which is formulated directly on the special orthogonal group SO(3)) and the *MEKF* parameter indicates the MEKF filter.

Using this naming system throughout the MATLAB code suite, four independent sets of parameters were developed that are exactly equal in terms of initial state measurements and noise levels. This system, while increasing the computational time and complexity, allowed for each filter to be independently tested, ensuring that no one variable is having an effect on another filter. This also allows for a fair evaluation and comparison of the different filters and their implementation methods.

The MATLAB code suite is arranged in cell mode, to allow for the efficient computation of blocks of code. This helps to reduce the computational time, as the initialization values can be pre-calculated. Initial variables such as the *time-step*, *state X*, *initial state angle*, *angle of rotation*, *process noise*, and *measurement noise* were initialized and defined for each of the different filters and implementations.

The next, significant code block is the time loop in which the filter equations and the gain equations, including all relevant parameters, are calculated. The calculation of the filter equations and the filter gain equations is a complex map, which relies on the calculation of each parameter in the previous time step. Figure 2 below details the calculations and the dependencies of each parameter on other parameters in the time loop.

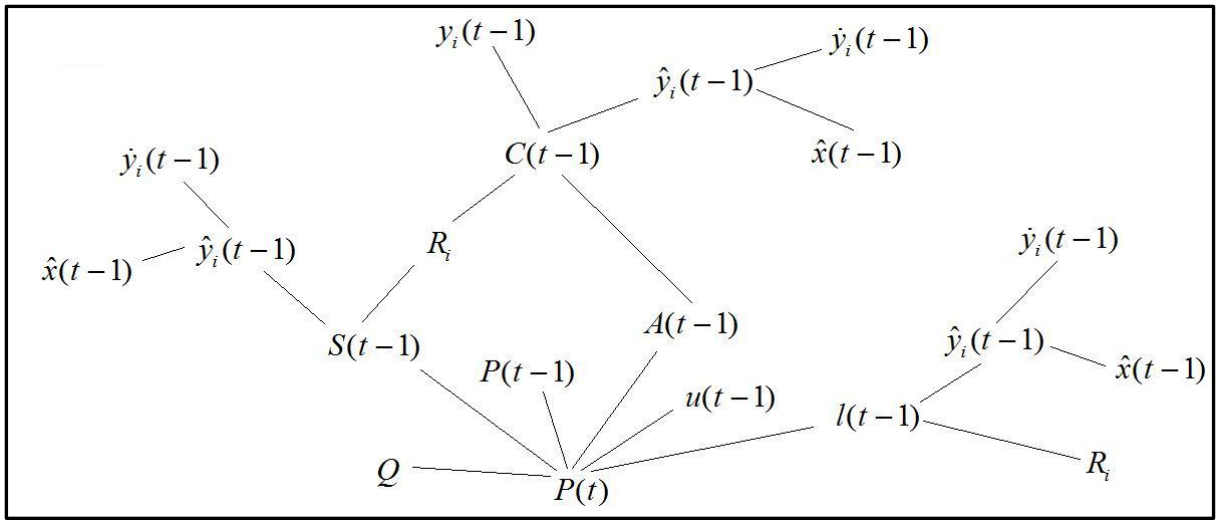

Figure 2- Parameter Dependency Map of GAME Filter Gain Equation

In order to calculate the values of each parameter for each time step, several user-defined functions needed to be implemented. These user-defined functions include *LowerIndexOperator* which calculates the lower index operator of a column vector as defined in the glossary and notation section and *SymmetricProjector* which similarly calculates the symmetric projection of a matrix as defined in the glossary and notation section. In addition, several user-defined functions were included in the code suite, which were used to calculate transformations of variables.

The extension of the MATLAB code suite to include an implementation of Choi's algorithm for the solution of Riccati differential equations required several functions to be written to compute various parameters. The functions *Choi_ABar*, *Choi_BBar*, *Choi_QBar* and *Choi_Rbar* were written to calculate the variables $\bar{A}_{k+1,r}$, $\bar{B}_{k+1,r}$, $\bar{Q}_{k+1,r}$ and $\bar{R}_{k+1,r}$ as defined in the '*Choi's Method*' subsection of section IV of this report. As an example, the function *Choi_QBar* is shown in figure 3below. The full code suit is reproduced in Appendix A.

```
function Y = Choi_QBar(Q, h, X, r)
sumQ = 0;
for i = 1:r
    if ((size(X,3) - i + 1) <1)
        sumQ = sumQ + (((-1)^(i-1))/i) * nchoosek(r,i) * X(:,:,1);
    else
    sumQ = sumQ + (((-1)^(i-1))/i) * nchoosek(r,i) * X(:,:,end-i+1);
    end
end
Y = sumQ + (h * Q);
end
```

Figure 3- The User-Defined MATLAB Function Choi_QBar

Using these functions at each time step, one derives the algebraic Riccati equation outlined in Choi's paper.

$$\bar{Q}_{k+1,r} + X_{k+1}\bar{A}_{k+1,r} + \bar{B}_{k+1,r}X_{k+1} - X_{k+1}\bar{R}_{k+1,r}X_{k+1} = 0$$

Choi's solution, which is outlined in [2], next uses Newton's method to derive a Sylvester equation based on the ARE above. The process of using Newton's method and the approximation made to arrive at the Sylvester equation are outlined in Choi's paper. In addition Choi presents the Sylvester equation for the reader. As such, in the implementation of Choi's Method in MATLAB, only the Sylvester equation derived from the ARE must be solved. In order to solve this equation, a user-defined MATLAB function *Choi_SylvesterSolver*, is used. This function, shown below, uses a MATLAB defined function *sylv* to solve the Sylvester equation. This is one of MATLAB's two Sylvester equation solver functions, the other being *lyap*. The function *sylv* was chosen because it implements the Hessenberg-Schur algorithm which Choi also employs, rather than the Bartels-Stewart algorithm, which is used in the *lyap* function.

```
function Y = Choi_SylvesterSolver(Q, A, B, R, h, X, r, Choi_NewtonIterations)


for i = 0:Choi_NewtonIterations

    A_Sylvester = Choi_BBar(B,X,h,r) - (X * Choi_RBar(R,h));
    B_Sylvester = Choi_ABar(A,X,h,r) - (Choi_RBar(R,h) * X);
    C_Sylvester = (Choi_QBar(Q,h,X,r)) +(X * Choi_RBar(R,h) * X);

    Y = sylv(A_Sylvester, B_Sylvester, C_Sylvester);

        X = Y;

end
end
```

Figure 4- User-Defined MATLAB Function Choi_SylvesterSolver

This user-defined MATLAB function is then called in the time-loop of the MATLAB code suite to solver the gain equations of the MEKF and the GAME filter. The MATLAB implementation of each of the gain equations using the two different solution algorithms is shown in the figure below.

```
Ps_B_SO3(:,:,i+1) = Ps_B_SO3(:,:,i) + dt * ((1 * projsym(Ps_B_SO3(:,:,i)...
    * LowerIndexOperator(2 * u_B_SO3-(Ps_B_SO3(:,:,i) * inos_B_SO3)))) + ...
    Q_B_SO3 + Ps_B_SO3(:,:,i) * (quads_B_SO3) * Ps_B_SO3(:,:,i));

Ps_B_MEKF(:,:,i+1) = Ps_B_MEKF(:,:,i) + dt *(Q_B_MEKF + (Ps_B_MEKF(:,:,i)...
    *0.5*LowerIndexOperator(u_B_MEKF)) - (0.5*LowerIndexOperator(u_B_MEKF)...
    *Ps_B_MEKF(:,:,i)) + (Ps_B_MEKF(:,:,i)* (quads_B_MEKF) * Ps_B_MEKF(:,:,i)));


Ps_C_SO3(:,:,i+1) = Choi_SylvesterSolver(Q_C_SO3, 0.5 * ...
    LowerIndexOperator(u_C_SO3), -0.5 * LowerIndexOperator(u_C_SO3), ...
    quads_C_SO3, dt, Ps_C_SO3(:,:,i), r, Choi_NewtonIterations);

Ps_C_MEKF(:,:,i+1) = Choi_SylvesterSolver(Q_C_MEKF, 0.5 * ...
    LowerIndexOperator(u_C_MEKF), -0.5 * LowerIndexOperator(u_C_MEKF),...
    quads_C_MEKF, dt, Ps_C_MEKF(:,:,i), r, Choi_NewtonIterations);
```

Figure 5- MATLAB Implementation of Filter Gain Equations

Results

The final implementation of the filter gain equations required the designation of two important parameters. Firstly the parameter *r*, as defined in Choi's paper, represents the number of iterations of the backwards difference operator used in determining the ARE. For the purposes of testing, a value of 6 was initially chosen, but it was later determined that a

value between 1 and 3 was best. The second parameter that needed to be defined was the number of iterations used for Newton's method. The variable name given to this parameter was *Choi_NewtonIterations*. For much of the testing phase, this value remained at 5.

For initial implementations, the noise levels for both the *process noise* and the *measurement noise* were set to high levels. This is because a filtering system works better when there is a greater amount of noise as compared with a system that has very little noise because the equations that define the filter can more easily determine what is relevant and what is not. The equations that define filtering systems are not designed for the case where the input signal has no noise, and as such, for signals with low levels of noise, implementing a filtering system is difficult. High levels of noise are indicative of the noise levels on UAV systems whereas low noise levels are reflective of the noise levels in spacecraft and aerospace applications. The initial value of the *process noise* was pi/4 and the initial value for the *measurement noise* was pi/3. These values are reflective of UAV noise levels as determined for simulation purposes in [1].

Before detailing the results of the implementations of each of these filtering systems in MATLAB, a note must be made about the GAME filter and the implementation of its gain equation using Choi's method. The symmetric projector term in the equation

$$\mathbb{P}_s\left(P\left(2u-Pl\right)_\times\right)$$

introduces a complex term which has both quadratic and linear terms in one. For the solution of this equation using Choi's method to be accurate, this term must be separated into its constituent quadratic and linear parts in order to assemble a Riccati equation analogous to the RDE considered by Choi's method. As previously indicated, this complex disassembly of the term has not been included in this report due to time constraints and as such the implementation of the GAME filter using Choi's method is not possible. In its place, a filter that closely approximates the GAME filter is implemented using Choi's method for the purposes of comparison with the MEKF. This filter is termed the SO3 filter due to it formulation on the special orthogonal group SO(3).

Instead, to limit the scope of the report, the analysis focuses on the comparison between the error convergence of both MEKF and the GAME filter, when implemented using the simple Euler method. In addition, the performances of Euler's method and Choi's method are

compared in terms of the MEKF, in order to establish the superiority of one method over another in terms of the error convergence of the filter.

For the purposes of implementation of the two filtering systems using Choi's method, the GAME filter is modelled as an MEKF with an adjusted quadratic term. This provides a reasonable estimate of what the derived gain equation of the GAME filter might be and is used purely for visualization purposes.

The initial evaluation of the two filtering systems had the following parameters. The results of this simulation are shown below.

$$r = 1$$

$$Choi\_NewtonIterations = 5$$

$$Process\ Noise = \text{pi}/4$$

$$Measurement\ Noise = \text{pi}/3$$

The results of the MATLAB implementation are displayed in a graphical form, which allows for a convenient method of analysis. There are two MATLAB plots generated for each simulation, with each containing six subplots. The first plot details the results of the simulation for the MEKF using both Choi's method and the simple Euler method. The second plot details the results of the GAME filter using both Choi's method and the simple Euler method. There are three subplots for each method of solution. The first shows the rotation angle of the current state and the second shows the rotation angle of the filter estimate. These two subplots together give an indication of the level of accuracy of the filter in determining the rotation angle of the system state. The third and most important subplot depicts the estimation error comparison between the previous two subplots, that is, it looks at the difference in the actual rotation angle and the rotation angle estimate of the filter.

Figure 6 is significant for comparing the effectiveness of Choi's method for solving the gain equation of the MEKF as compared to the simple Euler method. As the parameters used for each implementation are the same, the only difference between the two methods is the method of solving the gain equation of the filter. At the high level of noise used for this simulation, there is a significant difference in the performance of the MEKF using the two different algorithms. The graphs of the error convergence show a much faster error convergence for the MEKF implementing Choi's algorithm than for the MEKF which implements the simple Euler method.

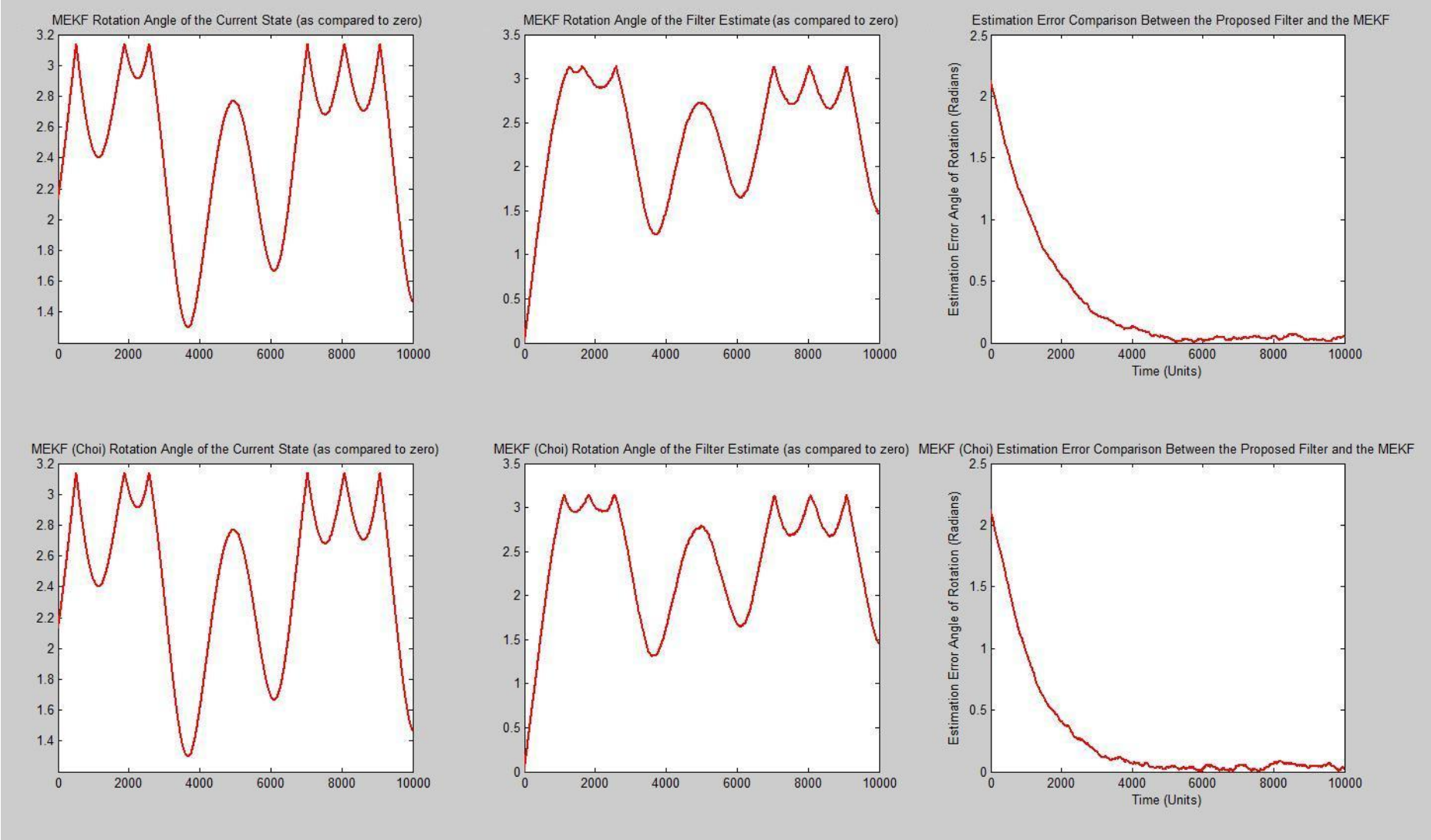

Figure 6- Graphical View of Results of MEKF Simulation

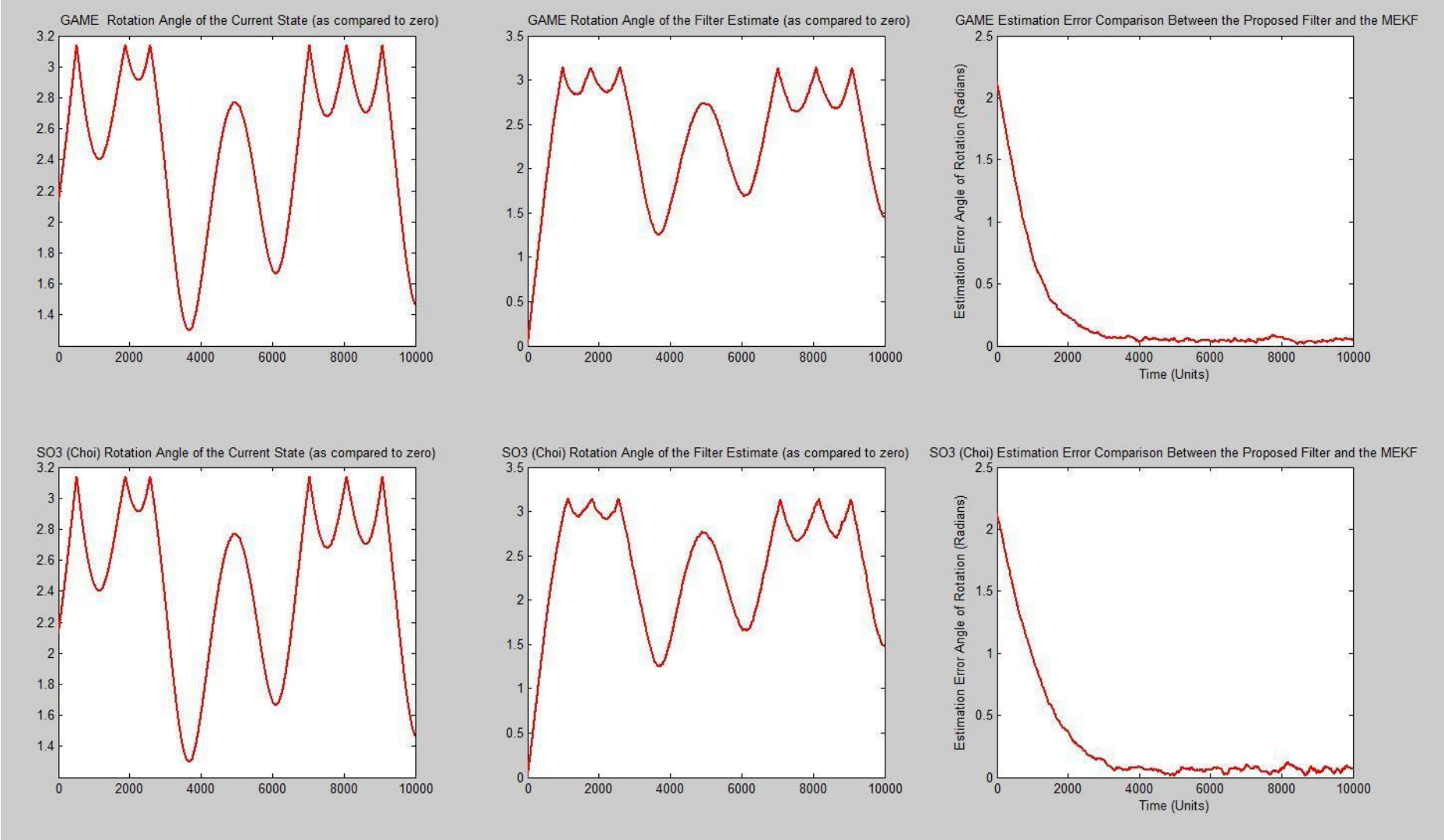

Figure 7- Graphical View of Results of GAME Filter Simulation

While it was anticipated that Choi's implementation would result in a slightly faster error convergence compared to an implementation employing the simple Euler method, the results show that Choi's algorithm far outperforms the simple Euler algorithm. This is an unexpected result given the nature of each of the two algorithms. Both algorithms are methods of numerical integration that are used to approximate the solution of the gain equation at each time step. The fact that each algorithm quickly converges to a level that is close to zero indicates that the algorithm is working as expected and is returning an accurate approximation.

Given the much greater performance of Choi's algorithm, this could suggest that there is an unknown aspect of the particular variables in the MEKF gain equation that results in Choi's method being far superior to the Euler method. Alternatively, this unusual degree of performance could suggest an error in the implementation of Choi's method as will be further discussed later in the report.

An alternative theory that describes the unexpected results obtained from the simulation of the two filtering systems relates to the nature of the numerical solution of Choi's method. As detailed throughout this report, the purpose of a filtering equation is to model and predict the state equation of the system and the purpose of the gain equation is to in turn determine the solution to the filtering equation. As such, one can consider two levels of numerical estimation that are used to determine the state of the system. A numerical algorithm can be expected to oscillate above and below the true solution of an equation, without ever perfectly fitting the equation. The significant superior performance of Choi's method relative to the Euler method could suggest that through the iteration process, as the filter equation pushes the filter equation closer to the true state equation, the movement of the filter equation towards the state equation accumulates. This could explain the much faster convergence of the filtering equations when implemented using Choi's method and requires further analysis to determine the true answer.

The next interesting result gained from the initial simulation is the comparison between the MEKF performance and the GAME filter performance. While these two filters cannot be directly compared based on their implementation using Choi's algorithm as described previously, both filters can be correctly implemented and compared at each a range of noise levels.

The simulation results for the error convergence of each of the two filters using the Euler method show a clear superiority of the GAME filter over the MEKF. The gradient of the initial part of the graph is far steeper for the GAME filter than for the MEKF which indicates its much faster error convergence. This is an expected and encouraging result that suggests the greater performance of the GAME filter over the MEKF for the given parameters and noise levels. This result agrees with the results of analysis in previous literature [1] which states that "the proposed filter [GAME] outperforms the industry standard filter, the MEKF, in simulations by showing faster error convergence." It is also suggested that this improved performance is due to the additional terms in the Riccati equation which provide more information about the system being measured, which in turn improves the accuracy and error convergence of the method of implementation.

While the performance of the GAME filter under these initial parameters has been shown by the simulation to be greater than that of the MEKF, the parameters chosen were relatively 'easy'. This means that the high level of noise ensures that the filtering systems will operate as expected. In order to prove the superior performance of the GAME filter with respect to the MEKF, both must be tested with parameters at different initial states and different noise levels.

The determination of a noise level designed to vigorously test the performance of the GAME filter is garnered from [33]. The parameters chosen in this paper were used to evaluate the performance of five different major filters in the field of attitude estimation at very low levels of noise. The results of this analysis had shown the lack of performance of each of the filter types as well as a good level of performance at slightly higher noise levels. As such, this low level of noise is employed to determine the high level of performance of the SO3 filter, which is a close approximation of the GAME filter.

The parameters determined from the literature for use in the testing of the filtering system are.

$$\textit{Process Noise} = 2.6875 \times 10^{-7}$$

$$\textit{Measurement Noise} = pi/180$$

The graphical results for the MEKF, the GAME filter and the SO3 filter using the above values for the noise level parameters as well as values of $r = 1$ and *Choi_NewtonIterations* =

5, are shown in figures 8 and 9 below. As expected these parameter values have resulted in some interesting results.

The performance of the MEKF is clearly shown to be very poor for the given low noise level parameters, as determined through simulation in previous literature [33]. Using the simple Euler implementation, it can be seen that the filter equation 'blew up'. This occurred because the Euler method used to numerically solve for the gain of the filter diverged very quickly from the true value due to the inaccuracy of the method at these parameters. This lack of performance is entirely expected given the previous simulation results of the MEKF at these levels as well as the unsuitability of the Euler method for determining the solution of a Riccati differential equation.

The MEKF results using Choi's implementation are themselves quite informative about the success of the method of implementation. At the beginning of the simulation, there is a clear divergence between the filter estimate and the true state. The jagged graph shows the high inaccuracy and oscillation of the filtering estimates around a value of pi. However, after 6000 time units, the filter eventually converges on the true state of the system. This is shown by the graph of the filter estimate rotation angle in the second subplot beginning to correlate with the rotation angle of the state. In the third subplot too, the graph begins to converge towards zero.

There are many interesting aspects of these results that must be discussed. Firstly, part of the simulation process involves the introduction of random variables to simulate noise. This results in different simulation results with every test. In previous simulations using Choi's method for the MEKF, as the filter eventually converges to zero, it 'blows up' once more. This is indicative of possibly two scenarios. Firstly, the MEKF, as shown by previous simulations, is unable to perform at these low levels of noise. If this were the case, it would suggest the high performance of the Choi algorithm despite the low levels of noise and the suboptimal filtering system. Alternatively, these results suggest an issue in the implementation of Choi's method. The second possibility arises due to the oscillations of the graph of error convergence around pi, suggesting that the solution of the Riccati equation is wrapping around on itself due to the parameters chosen in the implementation of Choi's method.

The implementation of the GAME filter shows similar results to that of the MEKF. Using the Euler method implementation, the filter also 'blows up' due to the divergence of the filter

estimate from the true state. Also similar to the MEKF, the implementation of the GAME filter using Choi's method also shows eventual convergence to some degree. However, as previously stated, the implementation of the GAME filter using Choi's method is not a true implementation of the GAME filter due to some mathematical complexities and must be further analysed before results can be obtained.

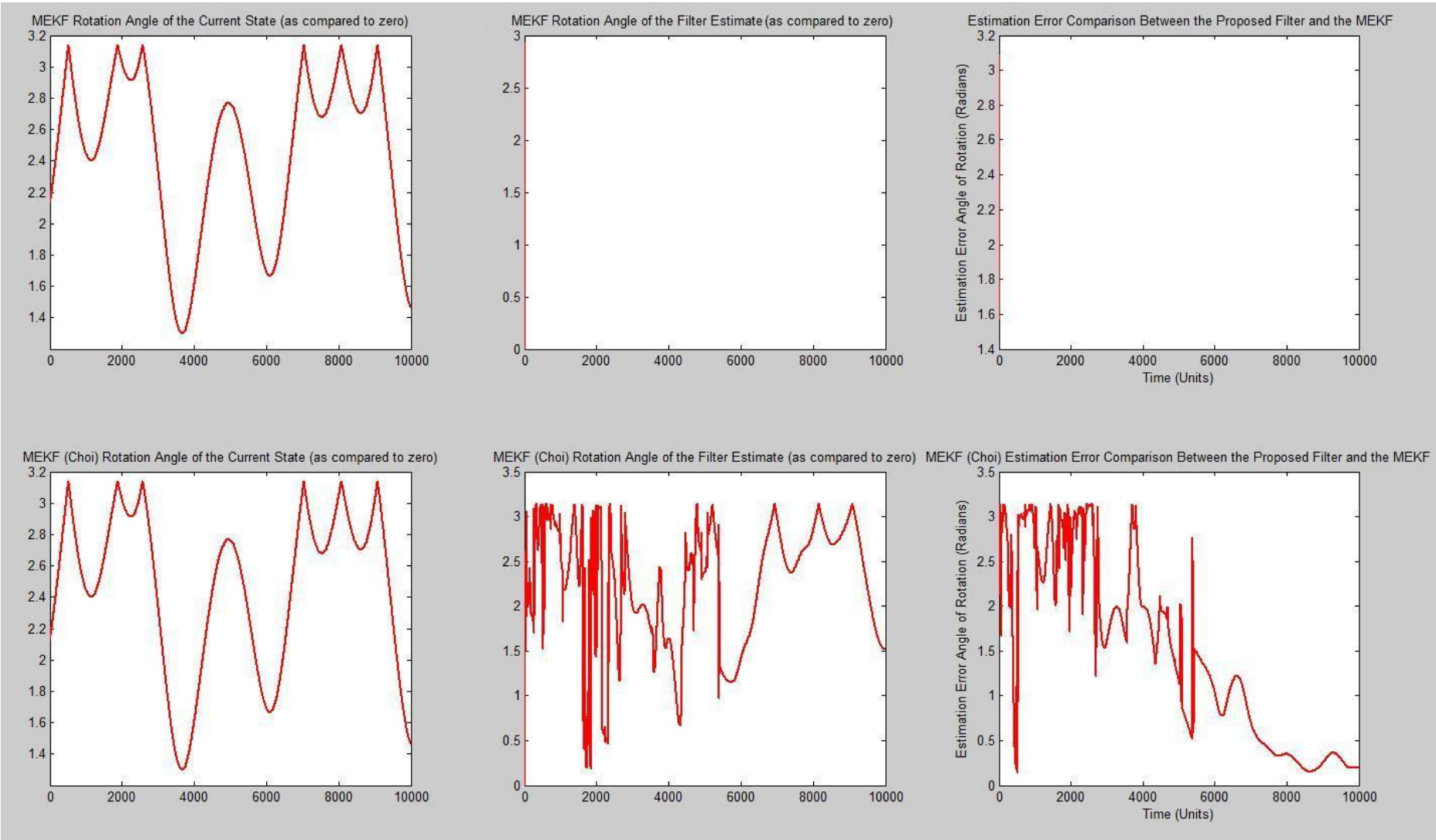

Figure 8- Graphical View of Results of MEKF Simulation with Low Noise Levels

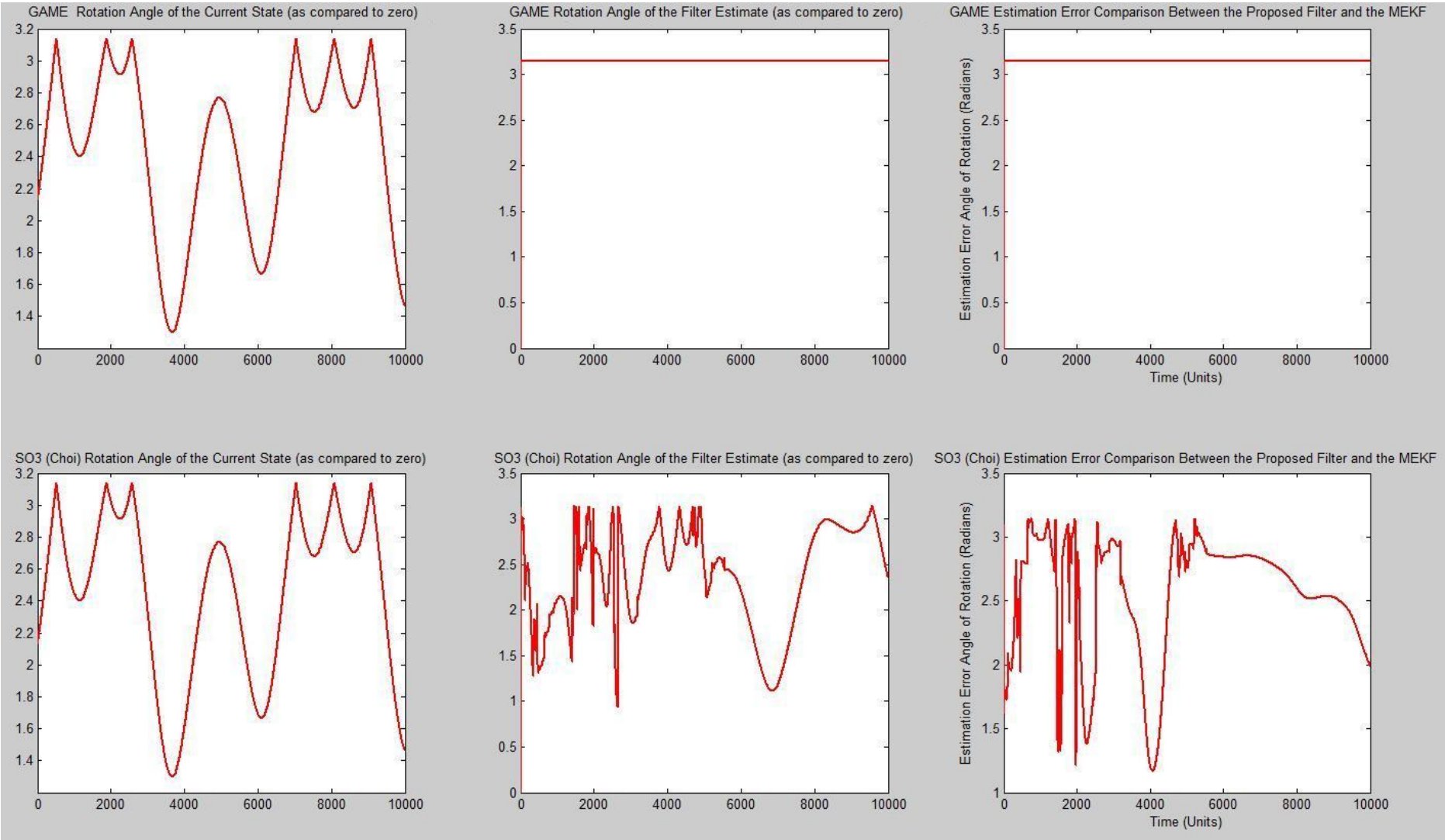

Figure 9- Graphical View of Results of GAME Filter and SO3 Filter with Low Noise Levels

Discussion

The analysis of the filtering systems considered in this report present some interesting results. While much of the analysis has promise in that it appears to agree with expected results, there are also many factors which must be further analysed in order to make comprehensive conclusions as to the nature of the filtering systems and their comparative performances.

The performance of Choi's implementation has been shown to be significantly better in giving an accurate numerical integration of the filter gain equations for both the MEKF and the GAME filter than the simple Euler method implementation at high noise levels. While this result can be stated conclusively for the MEKF at high noise level parameters, due to the difficulties implementing this method for the GAME filter, the same conclusion cannot be reached for the implementation at low noise levels. Instead the performance of the SO3 filter was compared to the performance of the MEKF at low noise levels using Choi's implementation. However, due to issues with the implementation of Choi's method, results are inconclusive. For a more comprehensive analysis of the performance of Choi's implementation, the gain equation for the GAME filter must be rearranged into a suitable form for Choi's method and the simulation results analysed.

The performance analysis and comparison of the MEKF and the GAME filter at the high noise levels using the simple Euler implementation showed that the GAME filter far outperformed the MEKF, which agreed with the results of previous research. However, results at lower noise levels are somewhat inconclusive. Testing at intermediate noise levels, that is, noise levels between the designated high and low values, showed faster error convergence for the MEKF than the GAME filter in some cases. This result suggests some limitation in the performance of Choi's implementation that could be linked to the chosen parameter values for r and *Choi_NewtonIterations*. Further analysis is required to determine if changes in the value of the parameters can extract a better performance from the GAME filter. An alternative explanation lies in the actual MATLAB implementation of the filter or Choi's algorithm. Before and conclusions can be made, the source of this error must be determined.

The results of the analysis completed to date suggest two significant issues in the implementation of Choi's method for solving the gain equations of the two filtering systems. The first potential issue is an incorrect implementation of the algorithm in MATLAB. Implementation of these

systems requires a number of complex parameters to be defined and calculated at each time step. The implementation of the filtering systems in particular, is quite sensitive, with the assigned sign values of particular variables determining the success or failure of the MATLAB implementation. This highlights the sensitivity of the methods being implemented and the potential for small implementation errors which could potential explain the reduced performance of the GAME filter and of Choi's method at lower noise levels.

A second significant problem the results have unearthed is the possibility that the method of solving the Riccati differential equations could be extremely sensitive. This observation has arisen due to the large variations in filter performance with relatively small changes in the noise level parameters discovered through the testing process. This could potentially have significant drawbacks for future implementation of either Choi's method or the GAME filter given the sensitivity could lie in either process. This is an extremely undesirable characteristic of a filtering system which experiences a range of noises levels and measurement data in applications.

In addition to the issues relating to the implementation of the GAME filter, further research is required in order to determine the performance of the GAME filter in comparison to other commonly used filters in industry applications. Such research would require the implementation of other filters using the same parameters as the GAME filter in order to assess its relative performance.

Regardless of the source of error in the implementation of the GAME filtering system in the MATLAB simulation, a great deal of further analysis is required before any significant conclusions as to the performance of either Choi's method or the GAME filter can be drawn.

VI. Conclusion

The purpose of this report has been to analyse, evaluate and compare two filtering systems, the GAME filter and the MEKF. In order to complete this analysis, each filtering system was implemented using two numerical integration techniques, a simple Euler method and Choi's method. The implementation of each system was completed in MATLAB, with common initial parameters used to comparatively measure the performance of each.

Initial results have indicated the greater performance of the GAME filter over the MEKF at high noise levels, as well as the greater performance of Choi's method over the simple Euler method at high noise levels. However, due to difficulties in the implementation of Choi's method for the GAME filter and unexpected sensitivities, results for each system at low noise levels are inconclusive.

The results do however indicate the likelihood of the GAME filter outperforming the MEKF at lower noise levels if the implementation of the system is correctly completed. This is because of the trend towards convergence of the filtering equations before they appear to 'blow up' upon reaching a level that is close to zero. This activity suggests an error in implementation that can potentially be improved.

The results of this report do show significant promise for the future implementation of the GAME filtering system and UAV systems given its greater performance in terms of error convergence than the industry standard filter, the MEKF [1]. If the implementation of the GAME filter is corrected, it is likely that it will be implemented on future industry applications and thus could play a significant role in the future filtering for attitude estimation of autonomous aerial vehicle systems.

It is clear that while positive results can be taken from the findings of this report, much works needs to be done before any concrete conclusions can be drawn. Firstly, the source of error in the implementation of the GAME filtering system using Choi's method must be determined. This will require significant testing and the mathematical formulas evaluated and ensuring that the results produced are as expected. In addition, further research into the effect of the iteration parameters on the performance of the filtering implementation must be conducted.

Once the implementation of the GAME filtering system and Choi's method is perfected, research must be conducted into the performance of both the MEKF and the GAME filter at low noise levels using Choi's method. This research will give a better picture of the relative performance of the GAME filter and the MEKF as well as the feasibility of an implementation using Choi's method. Depending on the results of this analysis, further research could then be conducted regarding the performance of the GAME filter and in particular, its performance compared to other filtering systems commonly used in industry applications.

References

[1] M Zamani, J Trumpf, and R Mahony, "Minimum-Energy Filtering for Attitude Estimation: A Geometric Correction to the Multiplicative Extended Kalman Filter," *DRAFT PAPER*, July 2012.

[2] Chiu H. Choi and Alan J. Laub, "Efficient Matrix-Valued Algorithms for Solving Stiff Riccati Differential Equations," *IEEE Transactions on Automatic Control*, vol. 35, no. 7, pp. 770-776, July 1990.

[3] F Markley, "Attitude Error Representations for Kalman Filtering," *Journal of Guidance Control and Dynamics*, vol. 26, no. 2, pp. 311-317, 2003.

[4] Uri M. Ascher and Linda R. Petzold, *Computer Methods for Ordinary Differential Equations and Differential Algebraic Equations*. USA: Society for Industrial and Applied Mathematics, 1998.

[5] J L Crassidis, F L Markley, and Y Cheng, "Nonlinear Attitude Filtering Methods," *Journal of Guidance Control and Dynamics*, vol. 30, no. 1, pp. 12-28, January 2007.

[6] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *Journal of Basic Engineering, Transactions of the ASME*, vol. 82, no. 1, pp. 35-45, 1960.

[7] Dennis G. Zill, *A First Course in Differential Equations with Modeling Applications*, 9th ed., Charlie Van Wagner, Ed. Belmont, USA: Brooks/Cole, 2009.

[8] Philip J Davis and Philip Rabinowitz, *Methods of Numerical Integration*. New York, USA: Academic Press, 1975.

[9] John Charles Butcher, *The Numerical Analysis of Ordinary Differential Equations: Runge-Kutta and General Linear Methods*.: Wiley-Interscience, 1987.

[10] C. W. Gear, "The Numerical Integration of Ordinary Differential Equations," *Mathematics of Computation*, vol. 21, pp. 146-156, 1967.

[11] Sergio Bittanti, Alan J. Laub, and Jan C. Willems, *The Riccati Equation*.: Springer-Verlag, 1991.

[12] E. J. Davidson and M. C. Maki, "The Numerical Solution of the Matrix Riccati Differential Equation," *IEEE Transactions on Automatic Control*, vol. AC-18, pp. 71-73, 1973.

[13] C. S. Kenney and R. B. Leipnik, "Numerical Integration of the Differential Matrix Riccati Equation," *IEEE Transactions on Automatic Control*, vol. AC-30, pp. 962-970, 1985.

[14] D. G. Lainiotis, "Generalized Chandasekhar Algorithms," *IEEE Transaction on Automatic Control*, vol. AC-21, pp. 728-732, 1976.

[15] D. G. Lainiotis, "Partitioned Riccati Solutions and Integration-free Doubling Algorithms," *IEEE Transactions on Automatic Control*, vol. AC-21, pp. 677-689, 1976.

[16] Alan J. Laub, "Schur techniques for Riccati Differential Equations," *Feedback Control of Linear and Nonlinear Systems*, pp. 165-1741, 1982.

[17] R. B. Leipnik, "A Canonical Form and Solution for the Matrix Riccati Differential Equation," *Bulletin - Australian Mathematical Society*, vol. 26, pp. 355-361, 1985.

[18] D. W. Rand and P. Winternitz, "Nonlinear Superposition Principles: A new Numerical Method for Solving Matrix Riccati Equations," *Computer Physics Communications*, vol. 33, pp. 305-328, 1984.

[19] W. T. Reid, *Riccati Differential Equations*.: New York: Academic, 1972.

[20] M. Sorine and P. Winternitz, "Superposition Laws for Solutions of Differential Matrix Riccati Equations Arising in Control Theory," *IEEE Transactions on Automatic Control*, vol. AC-30, pp. 266-272, 1985.

[21] Peter Benner and Mena Hermann, "BDF Methods for Large-Scale Differential Riccati Equations," *Proceedings of Mathematical Theory of Network and Systems*, vol. B. De Moor, B. Motmans, J. Willems, P. Van Dooren, and V. Blondel eds, 2004.

[22] John S. Baras and Demetrios G. Lainiotis, "Chandrasekhar Algorithms for Linear Time Varying Distributed Systems," *Information Sciences*, vol. 17, pp. 153-167, 1979.

[23] J. Harnad, P. Winternitz, and R. L. Anderson, "Superposition Principles for Matrix Riccati Equations," *Journal of Mathematical Physics*, vol. 24, pp. 1062-1072, 1983.

[24] Luca Dieci, "Numerical Integration of the Differential Riccati Equation and Some Related Issues," *SIAM Journal on Numerical Analysis*, vol. 29, no. 3, pp. 781-815, 1992.

[25] G. H. Golub, S. Nash, and C. Van Loan, "A Hessenburg-Schur Method for the Problem AX+XB=C," *IEEE Transactions on Automatic Control*, vol. AC-24, pp. 909-913, December 1979.

[26] R. H. Bartels and G. W. Stewart, "A Solution of the Equation AX+XB=C," *Communications of the ACM*, vol. 15, no. 9, pp. 820-826, 1972.

[27] Mohammad Zamani, "Deterministic Attitude and Pose Filtering, an Embedded Lie Groups Approach," Australian National University, Canberra, PhD Thesis [Draft Copy - as yet unpublished] 2012.

[28] S Bonnabel, P Martin, and P Rouchon, "Non-linear Symmetry-Preserving Observers on Lie Groups,"

IEEE Transactions on Automatic Control, vol. 54, no. 7, pp. 1709-1713, 2009.

[29] S Bonnabel, P Matin, and P Rouchon, "Symmetry-Preserving Observers," *IEEE Transactions on Automatic Control*, vol. 53, no. 11, pp. 2514-2526, 2008.

[30] P Coote, J Trumpf, R Mahony, and J C Willems, "Near-Optimal Deterministic Filtering on the Unit Circle," *Proceedings of the 48th IEEE Conference on Decision and Control*, pp. 5490-5495, 2009.

[31] R Mahony, T Hamel, and J.-M. Pflimlin, "Nonlinear Complementary Filters on the Special Orthogonal Group," *IEEE Transactions on Automatic Control*, vol. 53, no. 5, pp. 1203-1218, 2008.

[32] S. Bonnabel, P. Martin, and E. Salaun, "Invariant Extended Kalman Filter: Theory and Application to a Velocity-Aided Attitude Estimation Problem," *Joint 48th IEEE Conference on Decision and Control and 28th Chinese Control Conference*, pp. 1297-1304, December 2009.

[33] Xiaojun Tang, Zhenbao Liu, and Jiasheng Zhang, "Square-root Quaternion Cubature Kalman Filtering for Spacecraft Attitude Estimation," *Acta Astronauica*, vol. 76, pp. 84-94, February 2012.

Appendix A

```
% Author: Conor Horgan
% Supervisors: Jochen Trumpf, Mohammad Zamani, Robert Mahony
% Script: Choi_ABar.m
% Date: 29/09/2012

% The mathematics and formulas detailed in this script are based on
% derivations from the following paper:

% Title:     Efficient Matrix-Valued Algorithms for Solving Stiff Riccati
% Differential Equations
% Authors:   Chiu H. Choi, Alan J. Laub
% Journal:   IEEE Transactions on Automatic Control, vol.35, no.7, 1990


% This function comptues the ABar value, as defined in equation 3.9 of
% Chui H. Choi's paper. The ABar value is calculated as part of the process
% of transforming the Riccati Differential Equation (RDE) (eq 1.1) into an
% Algebraic Riccati Equation (ARE) (eq 3.9).

% Input Variables:
% A: The matrix A(t) of the RDE defined in eq 3.1
% h: The step size of the RDE defined in eq 3.2
% r: The number of iterations used for the backwards difference operator to
% construct the ARE in eq 3.9

% Output Variable:
% ABar: The matrix ABar of the ARE as defined in eq 3.9

% Function Description:
% This function computes a summation from 1 to the value of r and ceates a
% matrix of this value by multiplying it by the identity matrix of correct
% size. This matrix is then added to the original matrix A(t) multiplied by
% the step size, h. The returned value is ABar.

function ABar = Choi_ABar(A, h, r)
sumA = 0;
for i = 1:r
    sumA = sumA + 1/i;
end
ABar = (-1 * (0.5 * sumA) * eye(3)) + (h * A);
end
```

```
% Author: Conor Horgan
% Supervisors: Jochen Trumpf, Mohammad Zamani, Robert Mahony
% Script: Choi_BBar.m
% Date: 29/09/2012

% The mathematics and formulas detailed in this script are based on
% derivations from the following paper:

% Title:     Efficient Matrix-Valued Algorithms for Solving Stiff Riccati
% Differential Equations
% Authors:   Chiu H. Choi, Alan J. Laub
% Journal:   IEEE Transactions on Automatic Control, vol.35, no.7, 1990


% This function comptues the BBar value, as defined in equation 3.9 of
% Chui H. Choi's paper. The BBar value is calculated as part of the process
% of transforming the Riccati Differential Equation (RDE) (eq 1.1) into an
% Algebraic Riccati Equation (ARE) (eq 3.9).

% Input Variables:
% B: The matrix B(t) of the RDE defined in eq 3.1
% h: The step size of the RDE defined in eq 3.2
% r: The number of iterations used for the backwards difference operator to
% construct the ARE in eq 3.9

% Output Variable:
% BBar: The matrix BBar of the ARE as defined in eq 3.9

% Function Description:
% This function computes a summation from 1 to the value of r and ceates a
% matrix of this value by multiplying it by the identity matrix of correct
% size. This matrix is then added to the original matrix B(t) multiplied by
% the step size, h. The returned value is BBar.

function BBar = Choi_BBar(B, h, r)
sumB = 0;
for i = 1:r
    sumB = sumB + 1/i;
end
BBar = (-1 * (0.5 * sumB) * eye(3)) + (h * B);
end
```

```
% Author: Conor Horgan
% Supervisors: Jochen Trumpf, Mohammad Zamani, Robert Mahony
% Script: Choi_QBar.m
% Date: 29/09/2012

% The mathematics and formulas detailed in this script are based on
% derivations from the following paper:

% Title:     Efficient Matrix-Valued Algorithms for Solving Stiff Riccati
% Differential Equations
% Authors:   Chiu H. Choi, Alan J. Laub
% Journal:   IEEE Transactions on Automatic Control, vol.35, no.7, 1990


% This function comptues the QBar value, as defined in equation 3.9 of
% Chui H. Choi's paper. The QBar value is calculated as part of the process
% of transforming the Riccati Differential Equation (RDE) (eq 1.1) into an
% Algebraic Riccati Equation (ARE) (eq 3.9).

% Input Variables:
% Q: The matrix Q(t) of the RDE defined in eq 3.1
% h: The step size of the RDE defined in eq 3.2
% r: The number of iterations used for the backwards difference operator to
% construct the ARE in eq 3.9

% Output Variable:
% QBar: The matrix QBar of the ARE as defined in eq 3.9

% Function Description:
% This function computes a summation from 1 to the value of r which is
% then multiplied by the binomial coefficient of (r,i). The resulting value
% is then multiplied by a previously calculated value of the matrix X(t)
% computed according to the index, i. The resultant matrix is added to the
% Q(t) matrix multiplied by the time step, h. The resulting value is QBar.

function QBar = Choi_QBar(Q, h, X, r)
sumQ = 0;
for i = 1:r
    if ((size(X,3) - i + 1) <1)
        sumQ = sumQ + (((-1)^(i-1))/i) * nchoosek(r,i) * X(:,:,1);
    else
    sumQ = sumQ + (((-1)^(i-1))/i) * nchoosek(r,i) * X(:,:,end-i+1);
    end
end
QBar = sumQ + (h * Q);
end
```

```
% Author: Conor Horgan
% Supervisors: Jochen Trumpf, Mohammad Zamani, Robert Mahony
% Script: Choi_RBar.m
% Date: 29/09/2012

% The mathematics and formulas detailed in this script are based on
% derivations from the following paper:

% Title:     Efficient Matrix-Valued Algorithms for Solving Stiff Riccati
% Differential Equations
% Authors:   Chiu H. Choi, Alan J. Laub
% Journal:   IEEE Transactions on Automatic Control, vol.35, no.7, 1990


% This function comptues the RBar value, as defined in equation 3.9 of
% Chui H. Choi's paper. The RBar value is calculated as part of the process
% of transforming the Riccati Differential Equation (RDE) (eq 1.1) into an
% Algebraic Riccati Equation (ARE) (eq 3.9).

% Input Variables:
% R: The matrix R(t) of the RDE defined in eq 3.1
% h: The step size of the RDE defined in eq 3.2

% Output Variable:
% RBar: The matrix RBar of the ARE as defined in eq 3.9

% Function Description:
% This function multiplies the matrix R(t) by the step size, h. The
% returned value is RBar.

function RBar = Choi_RBar(R, h)
RBar = h*R;
end
```

```
% Author: Conor Horgan
% Supervisors: Jochen Trumpf, Mohammad Zamani, Robert Mahony
% Script: Choi_SylvesterSolver.m
% Date: 29/09/2012

% The mathematics and formulas detailed in this script are based on
% derivations from the following paper:

% Title:     Efficient Matrix-Valued Algorithms for Solving Stiff Riccati
% Differential Equations
% Authors:   Chiu H. Choi, Alan J. Laub
% Journal:   IEEE Transactions on Automatic Control, vol.35, no.7, 1990


% This function finds the solution the equation AX + XB = C using the
% Hessenberg-Schur method for solving Sylvester equations, where the
% values of A,B and C are derived in Choi's paper.

% Input Variables:
% Q: The matrix Q(t) of the RDE defined in eq 3.1
% A: The matrix A(t) of the RDE defined in eq 3.1
% B: The matrix B(t) of the RDE defined in eq 3.1
% R: The matrix R(t) of the RDE defined in eq 3.1
% h: The step size of the RDE defined in eq 3.2
% X: The matrix X(t) of the RDE defined in eq 3.1
% r: The number of iterations used for the backwards difference operator to
% construct the ARE in eq 3.9
% Choi_NewtonIterations: The number of iterations used in solving the
% Sylvester equation ate each time step

% Output Variable:
% QBar: The matrix QBar of the ARE as defined in eq 3.9

% Function Description:
% This function determines the value of the matrices A,B and C used in the
% Sylvester equation as derived in Choi's paper. The Sylvester equation is
% the solved using the MATLAB function 'sylv' which solves the Sylevester
% equation using the Hessenberg-Schur method. This returns the value of X
% for the next time step.

function Y = Choi_SylvesterSolver(Q, A, B, R, h, X, r, Choi_NewtonIterations)

for i = 0:Choi_NewtonIterations

    A_Sylvester = Choi_BBar(B,h,r) - (X * Choi_RBar(R,h));
    B_Sylvester = Choi_ABar(A,h,r) - (Choi_RBar(R,h) * X);
    C_Sylvester = (Choi_QBar(Q,h,X,r)) +(X * Choi_RBar(R,h) * X);

    Y = sylv(A_Sylvester, B_Sylvester, C_Sylvester);
    X = Y;
end

end
```

```
% Author: Conor Horgan
% Supervisors: Jochen Trumpf, Mohammad Zamani, Robert Mahony
% Script: LowerIndexOperator.m
% Date: 26/09/2012

% The mathematics and formulas detailed in this script, are based on
% derivations from the following paper:

% Title:     Minimum-Energy Filtering for Attitude Estiamtion: A Geometric
% Correction to the Multiplicative Extended Kalman Filter
% Authors:   Mohamad Zamani, Jochen Trumpf and Robert Mahony
% Journal:   DRAFT PAPER

% Input Variable:
% X: A 3x1 vector

% Output Variable:
% Y: A skew symmetric matrix of the vector X

% Function Description:
% This function computes the lower index operator of vector X as defined on
% page 4 of the above paper.

function Y = LowerIndexOperator(X)

if (size(X) == [3 1])
    Y =[ 0 -X(3) X(2);
        X(3) 0 -X(1);
        -X(2) X(1) 0;];
else
    error('Error:DimensionError', 'Input matrix X must be of size (3x1)')
end
end
```

```
% Authors: Mohammad Zamani, Conor Horgan
% Supervisors: Jochen Trumpf, Mohammad Zamani, Robert Mahony
% Script: Simulator.m
% Date: 30/09/2012

% The mathematics and formulas detailed in this script, for the
% construction and solution of the GAME are based on derivations from the
% following paper and thesis:

% Paper-
% Title:     Minimum-Energy Filtering for Attitude Estiamtion: A Geometric
% Correction to the Multiplicative Extended Kalman Filter
% Authors:   Mohamad Zamani, Jochen Trumpf and Robert Mahony
% Journal:   DRAFT PAPER

% Thesis-
% Title:     Deterministic Attitude and Pose Filtering, an Embedded Lie
% Groups Approach
% Author:   Mohamad Zamani
% Institution:   DRAFT THESIS, Australian National University

% This script is designed to simulate the filter and gain equations, 4.3,
% 4.4, 4.5 and 4.6 on page 28, chapter 4, of Behzad's Thesis draft copy.
% The code is based on a code suite written by Mohammad Zamani and has been
% extended for the simulation of the SO3 and MEKF filters implemented using
% an Euler method and Choi's method for solving the gain equations of both
% filters.

% The functions and parameters in this script are all calculated four times
% for the two filtering systems and the two implementation methods for the
% solution of the filter gain equations. The naming convention used is as
% follows:

% _B_SO3:   The GAME filter implemented with Euler method
% _B_MEKF:  The MEKF implemented with Euler method
% _C_SO3:   The GAME filter implemented with Choi's method
% _C_MEKF:  The MEKF implemented with Choi's method


clc
clear all
close all


% Time Variables
dt = 0.001;          % Time Step
Tf = 1;              % Maximum Time
tf = Tf/dt;
t = 0:dt:Tf-dt;

% Iterations Variables
Choi_NewtonIterations = 5;  % Newton Equation Iterations
r = 1;                      % Backwards Difference Operator Iterations

warning('off', 'all');


% Input Frequency
```

```
inpfreq = 2;

% Noise Level and Eigenvalue Control
%*************************************************************************
eigenvalues = 'OFF';            % Set to 'ON' to calculate eigenvalues
noise = 'LOW';                  % Set to 'HIGH' or 'LOW'

eigen_B_SO3(:,:,1) = [0;0;0];
eigen_B_MEKF(:,:,1) = [0;0;0];

eigen_C_SO3(:,:,1) = [0;0;0];
eigen_C_MEKF(:,:,1) = [0;0;0];

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

%% System variables


% State X represented with a normalized quaternion 4-vector q
q_B_SO3 = zeros(4,tf);
q_B_MEKF = zeros(4,tf);

q_C_SO3 = zeros(4,tf);
q_C_MEKF = zeros(4,tf);

% Zero quaternion associated to X=I
e = [1;0;0;0];

% Initial State
% ***small angle initial state
% q(:,1) = [10;-3;-3;-1];
% ***medium angle initial state
% q(:,1) = [5.5;-3;-3;-1];
% ***large angle initial state
% q(:,1) = [-1.5;-9;0;-10];
% 120 degrees
q_B_SO3(:,1) = [-7.5;-9;0;-10];
q_B_MEKF(:,1) = [-7.5;-9;0;-10];

q_C_SO3(:,1) = [-7.5;-9;0;-10];
q_C_MEKF(:,1) = [-7.5;-9;0;-10];

% 104 degrees
% q(:,1) = [-10.5;-9;0;-10];
% q_0=[1;2;1;1];


% Normilizing the initial quaternion
q_B_SO3(:,1) = q_B_SO3(:,1)/norm(q_B_SO3(:,1));
q_B_MEKF(:,1) = q_B_MEKF(:,1)/norm(q_B_MEKF(:,1));

q_C_SO3(:,1) = q_C_SO3(:,1)/norm(q_C_SO3(:,1));
q_C_MEKF(:,1) = q_C_MEKF(:,1)/norm(q_C_MEKF(:,1));



% Initial states angle in degrees
inistateang_B_SO3 = quat2angle(q_B_SO3(:,1),e);
```

```
inistateang_B_MEKF = quat2angle(q_B_MEKF(:,1),e);

inistateang_C_SO3 = quat2angle(q_C_SO3(:,1),e);
inistateang_C_MEKF = quat2angle(q_C_MEKF(:,1),e);




% Angle of rotation of the state
qangle_B_SO3 = zeros(tf,1);
qangle_B_SO3(1) = inistateang_B_SO3;
qangle_B_MEKF = zeros(tf,1);
qangle_B_MEKF(1) = inistateang_B_MEKF;

qangle_C_SO3 = zeros(tf,1);
qangle_C_SO3(1) = inistateang_C_SO3;
qangle_C_MEKF = zeros(tf,1);
qangle_C_MEKF(1) = inistateang_C_MEKF;


%--------- Reference directions

% y_0 =[-0.4925  0.3419; 0.2111  -0.9117; 0.8443   -0.2279];
%
% Alternative reference directions
%
% y_0 = eye(3);
%
%-----2 directions very close to each other
%y_0 =[-0.4925  -0.4925; 0.2111  0.2111; 0.8443   0.8443];
%-----------
% y_0 = [0.2182 -0.9701; -0.4364 -0.2425; 0.8729 0];
% y_0 = [0.2182 ; -0.4364 ; 0.8729];
% y_0 = [1 ; 2 ; 3];
% y_0 = [1*sin(3*t);cos(3*t);0*t];
% y_0 = [0.2182*ones(1,tf);-0.4364*ones(1,tf);0.8729*ones(1,tf)];

% ***Large: 90 degree angle***
y_B_SO3_0 = [1 0;0 1; 0 0];
y_B_MEKF_0 = [1 0;0 1; 0 0];

y_C_SO3_0 = [1 0;0 1; 0 0];
y_C_MEKF_0 = [1 0;0 1; 0 0];
%
% *** 31.3 degree angle***
% y_0=[-0.4925   -0.7947;0.2111    0.4122;0.8443    0.4455];

% *** Mediaum: 50.6 degree angle
% y_0=[-0.4925   -0.347;0.2111    0.4122;0.8443    0.4455];

% *** 3.7 degrees angle***
%  y_0 =[-0.4925   -0.4533;0.2111    0.2626;0.8443    0.8518];

%*** Small: 10.1 degrees angle
% y_0=[-0.4925   -0.6006 ; 0.2111    0.3049 ; 0.8443    0.7392];


% Angle between the two reference vector
```

```
refvecang_B_SO3 = 180/pi * acos(abs(y_B_SO3_0(:,1)' * y_B_SO3_0(:,2)));
refvecang_B_MEKF = 180/pi * acos(abs(y_B_MEKF_0(:,1)' * y_B_MEKF_0(:,2)));

refvecang_C_SO3 = 180/pi * acos(abs(y_C_SO3_0(:,1)' * y_C_SO3_0(:,2)));
refvecang_C_MEKF = 180/pi * acos(abs(y_C_MEKF_0(:,1)' * y_C_MEKF_0(:,2)));


% Number of reference directions---currently only works for two directions

%n = size(y_B_SO3_0,2);
n = 2;



% ------------------------------Noise Signals

% Process noise standard deviation (std)
%**************************************************************************
if (strcmp(noise, 'HIGH'))
    % The noise levels associated with UAV applications
    procnoisstd_B_SO3 = pi/4;
    procnoisstd_B_MEKF = pi/4;

    procnoisstd_C_SO3 = pi/4;
    procnoisstd_C_MEKF = pi/4;

elseif (strcmp(noise, 'LOW'))
    % The noise levels associated with spacecraft applications
    procnoisstd_B_SO3 = 2.6875 * 10^-7;
    procnoisstd_B_MEKF = 2.6875 * 10^-7;

    procnoisstd_C_SO3 = 2.6875 * 10^-7;
    procnoisstd_C_MEKF = 2.6875 * 10^-7;

end

% Omega v
%**************************************************************************

% Process noise coefficient
B_B_SO3 = procnoisstd_B_SO3 * eye(3);
B_B_MEKF = procnoisstd_B_MEKF * eye(3);

B_C_SO3 = procnoisstd_C_SO3 * eye(3);
B_C_MEKF = procnoisstd_C_MEKF * eye(3);


Q_B_SO3 = B_B_SO3*B_B_SO3';
Q_B_MEKF = B_B_MEKF*B_B_MEKF';

Q_C_SO3 = B_C_SO3*B_C_SO3';
Q_C_MEKF = B_C_MEKF*B_C_MEKF';
```

```
% % % %************* OPTION 2 TAC 2011 WHERE R,Q ~= I****************
% measurement noise std
%*************************************************************************
if (strcmp(noise, 'HIGH'))
    % The noise levels associated with UAV applications
    %60 degrees
    measnoisstd_B_SO3 = pi/3;
    measnoisstd_B_MEKF = pi/3;

    measnoisstd_C_SO3 = pi/3;
    measnoisstd_C_MEKF = pi/3;
elseif (strcmp(noise, 'LOW'))
    % The noise levels associated with spacecraft applications
    %1 degree
    measnoisstd_B_SO3 = 1 * pi/180;
    measnoisstd_B_MEKF = 1 * pi/180;

    measnoisstd_C_SO3 = 1 * pi/180;
    measnoisstd_C_MEKF = 1 * pi/180;

end

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

% 1 degree

% ---Measurement noise coefficient
D_B_SO3 = measnoisstd_B_SO3 * eye(3);
D_B_MEKF = measnoisstd_B_MEKF * eye(3);

D_C_SO3 = measnoisstd_C_SO3 * eye(3);
D_C_MEKF = measnoisstd_C_MEKF * eye(3);
% D = 1*diag([pi/12 pi/9 pi/6]);
% v = pi/12 * (randn (3,n,tf)+randn (3,n,tf));




Rinv_B_SO3 = (D_B_SO3*D_B_SO3')^-1;
Rinv_B_MEKF = (D_B_MEKF*D_B_MEKF')^-1;

Rinv_C_SO3 = (D_C_SO3*D_C_SO3')^-1;
Rinv_C_MEKF = (D_C_MEKF*D_C_MEKF')^-1;




%---------------------- measurement outputs
% Big matrix storing all the measurements through time of y
Y_B_SO3 = zeros(3,n,tf);
Y_B_MEKF = zeros(3,n,tf);

Y_C_SO3 = zeros(3,n,tf);
Y_C_MEKF = zeros(3,n,tf);

% A matrix storing the current time measurement y
```

```
y_B_SO3 = zeros(3,n);
y_B_MEKF = zeros(3,n);

y_C_SO3 = zeros(3,n);
y_C_MEKF = zeros(3,n);

% Function quattrans(q,y_0) does X'*y_0
for j=1:n
    y_B_SO3(:,j)=
quattrans(q_B_SO3(:,1),y_B_SO3_0(:,j))+D_B_SO3*randn(3,1);
    y_B_MEKF(:,j)=
quattrans(q_B_MEKF(:,1),y_B_MEKF_0(:,j))+D_B_MEKF*randn(3,1);

    y_C_SO3(:,j)=
quattrans(q_C_SO3(:,1),y_C_SO3_0(:,j))+D_C_SO3*randn(3,1);
    y_C_MEKF(:,j)=
quattrans(q_C_MEKF(:,1),y_C_MEKF_0(:,j))+D_C_MEKF*randn(3,1);
end



%-------------- SO(3) filter parameters
% State Xhat in quaternions
qs_B_SO3 = zeros(4,tf);
qs_B_SO3_0 = e;
qs_B_SO3(:,1) = qs_B_SO3_0;

qs_B_MEKF = zeros(4,tf);
qs_B_MEKF_0 = e;
qs_B_MEKF(:,1) = qs_B_MEKF_0;

qs_C_SO3 = zeros(4,tf);
qs_C_SO3_0 = e;
qs_C_SO3(:,1) = qs_C_SO3_0;

qs_C_MEKF = zeros(4,tf);
qs_C_MEKF_0 = e;
qs_C_MEKF(:,1) = qs_C_MEKF_0;



% Gain of the Riccati
Ps_B_SO3 = eye(3);
Ps_B_SO3_0 = 1 * eye(3);

Ps_B_MEKF = eye(3);
Ps_B_MEKF_0 = 1 * eye(3);

Ps_C_SO3 = eye(3);
Ps_C_SO3_0 = 1 * eye(3);

Ps_C_MEKF = eye(3);
Ps_C_MEKF_0 = 1 * eye(3);


% Rotation angle of the state
qsangle_B_SO3 = zeros(tf,1);
qsangle_B_MEKF = zeros(tf,1);
```

```
qsangle_C_SO3 = zeros(tf,1);
qsangle_C_MEKF = zeros(tf,1);


% Rotation angle error between the state and the systems angle
qseang_B_SO3 = zeros(tf,1);
qseang_B_SO3(1) = quat2angle(qs_B_SO3(:,1),q_B_SO3(:,1));

qseang_B_MEKF = zeros(tf,1);
qseang_B_MEKF(1) = quat2angle(qs_B_MEKF(:,1),q_B_MEKF(:,1));

qseang_C_SO3 = zeros(tf,1);
qseang_C_SO3(1) = quat2angle(qs_C_SO3(:,1),q_C_SO3(:,1));

qseang_C_MEKF = zeros(tf,1);
qseang_C_MEKF(1) = quat2angle(qs_C_MEKF(:,1),q_C_MEKF(:,1));



%% Display code variables

disp('################### S Y S T E M    P A R A M E T E R
S#######################')
disp('')
disp(['Simulation time step=',num2str(dt) ])
disp('')
disp(['Simulation final time=',num2str(Tf)])
disp('')
disp('Systems initial angle quat2angle(q(1)) in degrees=')
disp('')
display(180/pi * inistateang_B_SO3)
disp('')
disp('')
disp(['Input Frequency : inpfreq =',num2str(inpfreq)])
disp('Omega = [2*sin(inpfreq*t);-5*cos(inpfreq*t);-2*cos(inpfreq*t)]')
disp('')
disp(['Process noise STD in degrees =',num2str(180/pi *
procnoisstd_B_SO3)])
disp('')
disp(['Measurement noise STD in degrees =',num2str(180/pi *
measnoisstd_B_SO3)])
disp('')
disp('Reference directions in a matrix ')
disp('')
display(y_B_SO3_0)
disp('')
disp('Angle between the two reference vectors in degrees=')
disp('')
display(refvecang_B_SO3)
disp('#########################################################################
########')



%% Time evolution

for i = 1:tf-1
```

```
    %% --------------------------------System evolution
    time = i

    % Angular velocity input
    t = dt*i;
    omega_B_SO3 = [2*sin(inpfreq*t);-1*cos(inpfreq*t);2*cos(1*inpfreq*t)];
    omega_B_MEKF = [2*sin(inpfreq*t);-1*cos(inpfreq*t);2*cos(1*inpfreq*t)];

    omega_C_SO3 = [2*sin(inpfreq*t);-1*cos(inpfreq*t);2*cos(1*inpfreq*t)];
    omega_C_MEKF = [2*sin(inpfreq*t);-1*cos(inpfreq*t);2*cos(1*inpfreq*t)];

    % Measurement of Omega
    u_B_SO3 = omega_B_SO3 + (B_B_SO3 * randn(3,1));
    u_B_MEKF = omega_B_MEKF + (B_B_MEKF * randn(3,1));

    u_C_SO3 = omega_C_SO3 + (B_C_SO3 * randn(3,1));
    u_C_MEKF = omega_C_MEKF + (B_C_MEKF * randn(3,1));



    % Numerical integrator for quaternions
    q_B_SO3(:,i+1) = expm(dt * tangquat(omega_B_SO3)) * q_B_SO3(:,i);
    q_B_MEKF(:,i+1) = expm(dt * tangquat(omega_B_MEKF)) * q_B_MEKF(:,i);

    q_C_SO3(:,i+1) = expm(dt * tangquat(omega_C_SO3)) * q_C_SO3(:,i);
    q_C_MEKF(:,i+1) = expm(dt * tangquat(omega_C_MEKF)) * q_C_MEKF(:,i);



    y_B_SO3 = zeros(3,n);
    y_B_MEKF = zeros(3,n);

    y_C_SO3 = zeros(3,n);
    y_C_MEKF = zeros(3,n);


    for j = 1:n

        y_B_SO3(:,j) =  quattrans(q_B_SO3(:,i+1),y_B_SO3_0(:,j)) + D_B_SO3
* randn(3,1);
        y_B_MEKF(:,j) =  quattrans(q_B_MEKF(:,i+1),y_B_MEKF_0(:,j)) +
D_B_MEKF * randn(3,1);

        y_C_SO3(:,j) =  quattrans(q_C_SO3(:,i+1),y_C_SO3_0(:,j)) + D_C_SO3
* randn(3,1);
        y_C_MEKF(:,j) =  quattrans(q_C_MEKF(:,i+1),y_C_MEKF_0(:,j)) +
D_C_MEKF * randn(3,1);

    end
```

```
    %% ------------------------------------------------SO(3) Filter

    % The innovation term
    inos_B_SO3 = zeros(3,1);
    inos_B_MEKF = zeros(3,1);

    inos_C_SO3 = zeros(3,1);
    inos_C_MEKF = zeros(3,1);


    % The quadratic matrix in the Riccati
    quads_B_SO3 = zeros(3);
    quads_B_MEKF = zeros(3);

    quads_C_SO3 = zeros(3);
    quads_C_MEKF = zeros(3);

    for j = 1:n
        % Calculating \hat(y)_j s
        yhats_B_SO3 =  quattrans(qs_B_SO3(:,i),y_B_SO3_0(:,j));
        yhats_B_MEKF =  quattrans(qs_B_MEKF(:,i),y_B_MEKF_0(:,j));

        yhats_C_SO3 =  quattrans(qs_C_SO3(:,i),y_C_SO3_0(:,j));
        yhats_C_MEKF =  quattrans(qs_C_MEKF(:,i),y_C_MEKF_0(:,j));

        % Summation in the innovation terms
        inos_B_SO3 = inos_B_SO3 + cross((Rinv_B_SO3) * (yhats_B_SO3-
y_B_SO3(:,j)),yhats_B_SO3);
        inos_B_MEKF = inos_B_MEKF + cross((Rinv_B_MEKF) * (yhats_B_MEKF-
y_B_MEKF(:,j)),yhats_B_MEKF);

        inos_C_SO3 = inos_C_SO3 + cross((Rinv_C_SO3) * (yhats_C_SO3-
y_C_SO3(:,j)),yhats_C_SO3);
        inos_C_MEKF = inos_C_MEKF + cross((Rinv_C_MEKF) * (yhats_C_MEKF-
y_C_MEKF(:,j)),yhats_C_MEKF);



        % The symmetric projection in the Quadratic term
        C_B_SO3 = projsym((Rinv_B_SO3 * (yhats_B_SO3-y_B_SO3(:,j))) *
yhats_B_SO3');
        C_B_MEKF = projsym((Rinv_B_MEKF * (yhats_B_MEKF-y_B_MEKF(:,j))) *
yhats_B_MEKF');

        C_C_SO3 = projsym((Rinv_C_SO3 * (yhats_C_SO3-y_C_SO3(:,j))) *
yhats_C_SO3');
        C_C_MEKF = projsym((Rinv_C_MEKF * (yhats_C_MEKF-y_C_MEKF(:,j))) *
yhats_C_MEKF');

        % The geometric quadratic term in the Riccati
        A_B_SO3 = trace(C_B_SO3) * eye(3) - C_B_SO3;
        A_B_MEKF = trace(C_B_MEKF) * eye(3) - C_B_MEKF;

        A_C_SO3 = trace(C_C_SO3) * eye(3) - C_C_SO3;
        A_C_MEKF = trace(C_C_MEKF) * eye(3) - C_C_MEKF;

        % The ususal quadratic term
```

```
        S_B_SO3 = skew(yhats_B_SO3) * Rinv_B_SO3 * skew(yhats_B_SO3);
        S_B_MEKF = skew(yhats_B_MEKF) * Rinv_B_MEKF * skew(yhats_B_MEKF);

        S_C_SO3 = skew(yhats_C_SO3) * Rinv_C_SO3 * skew(yhats_C_SO3);
        S_C_MEKF = skew(yhats_C_MEKF) * Rinv_C_MEKF * skew(yhats_C_MEKF);

        % Summation of quadratic terms
        quads_B_SO3 = quads_B_SO3 + A_B_SO3 + S_B_SO3;
        quads_B_MEKF = quads_B_MEKF + S_B_MEKF;

        quads_C_SO3 = quads_C_SO3 - A_C_SO3 + S_C_SO3;
        quads_C_MEKF = quads_C_MEKF + S_C_MEKF;
    end

    % The observer equation for the SO(3) filter
    % tangquat returns the 4by4 skew matrix of the quaternion tangent space
    qs_B_SO3(:,i+1) = expm(dt * tangquat(u_B_SO3 - Ps_B_SO3(:,:,i) *
inos_B_SO3)) * qs_B_SO3(:,i);
    qs_B_MEKF(:,i+1) = expm(dt * tangquat(u_B_MEKF - Ps_B_MEKF(:,:,i) *
inos_B_MEKF)) * qs_B_MEKF(:,i);

    qs_C_SO3(:,i+1) = expm(dt * tangquat(u_C_SO3 - Ps_C_SO3(:,:,i) *
inos_C_SO3)) * qs_C_SO3(:,i);
    qs_C_MEKF(:,i+1) = expm(dt * tangquat(u_C_MEKF - Ps_C_MEKF(:,:,i) *
inos_C_MEKF)) * qs_C_MEKF(:,i);



    % Solution for P, the gain of the filters
    Ps_B_SO3(:,:,i+1) = Ps_B_SO3(:,:,i) + dt * ((1 *
projsym(Ps_B_SO3(:,:,i)...
        * LowerIndexOperator(2 * u_B_SO3-(Ps_B_SO3(:,:,i) * inos_B_SO3))))
+ ...
        Q_B_SO3 + Ps_B_SO3(:,:,i) * (quads_B_SO3) * Ps_B_SO3(:,:,i));

    Ps_B_MEKF(:,:,i+1) = Ps_B_MEKF(:,:,i) + dt *(Q_B_MEKF +
(Ps_B_MEKF(:,:,i)...
        *0.5*LowerIndexOperator(u_B_MEKF)) -
(0.5*LowerIndexOperator(u_B_MEKF)...
        *Ps_B_MEKF(:,:,i)) + (Ps_B_MEKF(:,:,i)* (quads_B_MEKF) *
Ps_B_MEKF(:,:,i)));


    Ps_C_SO3(:,:,i+1) = Choi_SylvesterSolver(Q_C_SO3, 0.5 *
LowerIndexOperator(u_C_SO3),...
        -0.5 * LowerIndexOperator(u_C_SO3), quads_C_SO3, dt,
Ps_C_SO3(:,:,i), r, Choi_NewtonIterations);

    Ps_C_MEKF(:,:,i+1) = Choi_SylvesterSolver(Q_C_MEKF, 0.5 *
LowerIndexOperator(u_C_MEKF),...
        -0.5 * LowerIndexOperator(u_C_MEKF), quads_C_MEKF, dt,
Ps_C_MEKF(:,:,i), r, Choi_NewtonIterations);




%*************************************************************************
```

```
    if (strcmp(eigenvalues, 'ON'))
        eigen_B_SO3(:,:,i) = eig(Ps_B_SO3(:,:,i));
        eigen_B_MEKF(:,:,i) = eig(Ps_B_MEKF(:,:,i));

        eigen_C_SO3(:,:,i) = eig(Ps_C_SO3(:,:,i));
        eigen_C_MEKF(:,:,i) = eig(Ps_C_MEKF(:,:,i));

    end

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



    %% variable storage

    % Storing the measurements
    for j = 1:n
        Y_B_SO3(:,j,i+1) = y_B_SO3(:,j);
        Y_B_MEKF(:,j,i+1) = y_B_MEKF(:,j);

        Y_C_SO3(:,j,i+1) = y_C_SO3(:,j);
        Y_C_MEKF(:,j,i+1) = y_C_MEKF(:,j);
    end


    % Rotation angle of the current state (as compared to zero)
    qangle_B_SO3(i+1) = quat2angle(q_B_SO3(:,i+1),e);
    qangle_B_MEKF(i+1) = quat2angle(q_B_MEKF(:,i+1),e);

    qangle_C_SO3(i+1) = quat2angle(q_C_SO3(:,i+1),e);
    qangle_C_MEKF(i+1) = quat2angle(q_C_MEKF(:,i+1),e);



    % Rotation angle of the filter's estimate (as compared to zero)
    qsangle_B_SO3(i+1) = quat2angle(qs_B_SO3(:,i+1),e);
    qsangle_B_MEKF(i+1) = quat2angle(qs_B_MEKF(:,i+1),e);

    qsangle_C_SO3(i+1) = quat2angle(qs_C_SO3(:,i+1),e);
    qsangle_C_MEKF(i+1) = quat2angle(qs_C_MEKF(:,i+1),e);

    % Rotation angle of the filter's estimate (as compared to current
state's angle)
    qseang_B_SO3(i+1) = quat2angle(qs_B_SO3(:,i+1),q_B_SO3(:,i+1));
    qseang_B_MEKF(i+1) = quat2angle(qs_B_MEKF(:,i+1),q_B_MEKF(:,i+1));

    qseang_C_SO3(i+1) = quat2angle(qs_C_SO3(:,i+1),q_C_SO3(:,i+1));
    qseang_C_MEKF(i+1) = quat2angle(qs_C_MEKF(:,i+1),q_C_MEKF(:,i+1));



end


%% Data Plotting

% % Angle Trajectories
```

```
% figure(1)
% plot(t,y_disp,'+g',t,qangle,'b',t,qsangle,'r','LineWidth',2)
% title('tracking systems trajectory rotation angle')
% Maximize(1)
figure(1)
subplot(2,3,1)
plot(qangle_B_SO3,'r','LineWidth',2)
title('SO3 (Behzad) Rotation Angle of the Current State (as compared to
zero)');

subplot(2,3,2)
plot(qsangle_B_SO3,'r','LineWidth',2)
title('SO3 (Behzad) Rotation Angle of the Filter Estimate (as compared to
zero)');

subplot(2,3,3)
plot(qseang_B_SO3,'r','LineWidth',2)
title('SO3 (Behzad) Estimation Error Comparison Between the Proposed Filter
and the MEKF')
xlabel('Time (Units)')
ylabel('Estimation Error Angle of Rotation (Radians)')


subplot(2,3,4)
plot(qangle_C_SO3,'r','LineWidth',2)
title('SO3 (Choi) Rotation Angle of the Current State (as compared to
zero)');

subplot(2,3,5)
plot(qsangle_C_SO3,'r','LineWidth',2)
title('SO3 (Choi) Rotation Angle of the Filter Estimate (as compared to
zero)');


subplot(2,3,6)
plot(qseang_C_SO3,'r','LineWidth',2)
title('SO3 (Choi) Estimation Error Comparison Between the Proposed Filter
and the MEKF')
xlabel('Time (Units)')
ylabel('Estimation Error Angle of Rotation (Radians)')

%**************************************************************************
figure(2)
subplot(2,3,1)
plot(qangle_B_MEKF,'r','LineWidth',2)
title('MEKF (Behzad) Rotation Angle of the Current State (as compared to
zero)');

subplot(2,3,2)
plot(qsangle_B_MEKF,'r','LineWidth',2)
title('MEKF (Behzad) Rotation Angle of the Filter Estimate (as compared to
zero)');


subplot(2,3,3)
plot(qseang_B_MEKF,'r','LineWidth',2)
title('MEKF (Behzad) Estimation Error Comparison Between the Proposed
Filter and the MEKF')
xlabel('Time (Units)')
```

```
ylabel('Estimation Error Angle of Rotation (Radians)')

subplot(2,3,4)
plot(qangle_C_MEKF,'r','LineWidth',2)
title('MEKF (Choi) Rotation Angle of the Current State (as compared to
zero)');

subplot(2,3,5)
plot(qsangle_C_MEKF,'r','LineWidth',2)
title('MEKF (Choi) Rotation Angle of the Filter Estimate (as compared to
zero)');


subplot(2,3,6)
plot(qseang_C_MEKF,'r','LineWidth',2)
title('MEKF (Choi) Estimation Error Comparison Between the Proposed Filter
and the MEKF')
xlabel('Time (Units)')
ylabel('Estimation Error Angle of Rotation (Radians)')
```