Sensors **2015**, *15*, 28402-28420; doi:10.3390/s151128402

OPEN ACCESS

sensors

ISSN 1424-8220

www.mdpi.com/journal/sensors

Article

Optimization Algorithm for Kalman Filter Exploiting the Numerical Characteristics of SINS/GPS Integrated Navigation Systems

Shaoxing Hu [1,*], Shike Xu [1], Duhu Wang [1] and Aiwu Zhang [2]

[1] School of Mechanical Engineering and Automation, Beihang University, Beijing 100191, China; E-Mails: xushike24k@163.com (S.X.); buaaben@163.com (D.W.)

[2] Ministry of Education Key Laboratory of 3D Information Acquisition and Application, Capital Normal University, Beijing100089, China; E-Mail: zhang_aiwu@126.com

***** Author to whom correspondence should be addressed; E-Mail: husx@buaa.edu.cn; Tel.: +86-10-8233-9130.

Academic Editor: Stefano Mariani

Received: 26 August 2015 / Accepted: 3 November 2015 / Published: 11 November 2015

Abstract: Aiming at addressing the problem of high computational cost of the traditional Kalman filter in SINS/GPS, a practical optimization algorithm with offline-derivation and parallel processing methods based on the numerical characteristics of the system is presented in this paper. The algorithm exploits the sparseness and/or symmetry of matrices to simplify the computational procedure. Thus plenty of invalid operations can be avoided by offline derivation using a block matrix technique. For enhanced efficiency, a new parallel computational mechanism is established by subdividing and restructuring calculation processes after analyzing the extracted "useful" data. As a result, the algorithm saves about 90% of the CPU processing time and 66% of the memory usage needed in a classical Kalman filter. Meanwhile, the method as a numerical approach needs no precise-loss transformation/approximation of system modules and the accuracy suffers little in comparison with the filter before computational optimization. Furthermore, since no complicated matrix theories are needed, the algorithm can be easily transplanted into other modified filters as a secondary optimization method to achieve further efficiency.

Keywords: computational optimization; SINS/GPS; closed-loop Kalman filter; block matrix; offline-derivation; parallel processing; accuracy-lossless decoupling; symbol operation

1. Introduction

In many SINS/GPS integrated navigation systems, common system models cannot conform to the use conditions of classical Kalman filter due to the "colored noise". To still be able to use the optimal estimation method, it is necessary to expand the system state model [1,2]. However, as the state dimensions increase, filtering computation will rapidly become expensive and unstable, and the so-called "dimension disaster" could even break out. In order to address this problem, scholars have reported various optimization algorithms which can lower the computational costs and/or enhance the numerical robustness. The typical optimization algorithms contain square-root information filtering SRIF [3], U-D decomposition filtering [4], singular value decomposition filtering [5] and their improved versions [6–8]. These modified filters, besides greatly reducing the time consumption, theoretically guarantee the positive definiteness of covariance and effectively avoid the numerical divergence, thus they are widely applied in the theoretical design of higher-order systems. Nevertheless, they require the support of relatively complex matrix theory in derivation and may pose certain difficulty in engineering. On the other hand, although these algorithms have decreased computational costs, their complexity is still $O(n^3)$. With further expansion of the state dimensions, the above decomposition algorithms, as a kind of general method, are also being challenged on their real-time capability. Recently, researchers have paid much attention to non-commonality methods, which are designed for some specific applications or based on specific models. For instance, aiming at integrated navigation applications, a kind of reduced-order Kalman filter (RDKF) [9–12] was promoted to ease the computational load. The idea of these filters is to reduce the model dimension by theoretical/engineering methods. As the filter dimension n is vital to computation time, the reduction of the state order will produce a direct benefit in terms of real time. However, the decrease of the state order will also bring partial accuracy damage, therefore, these methods may not be suitable for some high accuracy-demanding applications. Another kind of effective algorithm optimizes the float-point operations mainly by taking advantage of the sparse matrices (e.g., the transition matrix Φ_n in SINS/GPS) during the filtering computation [13–15]. In these algorithms, the so-called matrix accumulative method (MAM) or other online methods are used and the algorithm complexity can be simplified to $O(s^2 - u^2)$ (s/u represents the numbers of nonzero/1 elements in Φ_n). As the matrices in the high order model normally contain substantial numbers of zero elements, the computational time can be curtailed to an ideal level by usage of this sparse-matrix-based method. Meanwhile, being different from RDKF, the sparse-matrix-based methods do not change the system model and need little engineering approximation, thus they perform better in the accuracy-control aspect. But these methods also have their own limitations:

(1) Though they avoid massive unnecessary multiplying-zero operations, the methods still need extra $O(n^3)$ times estimation of zero elements in each matrix multiplication;
(2) Methods based on MAM, a kind of online method, could not seek any deeper optimization online and their efficiency is totally dependent on the number of zeros in the real-time matrix. To enhance the optimization efficiency, the usual way is to set more zero elements by approximation methods, but this comes at the cost of a certain accuracy loss.

In response to the above problems, we present here a new highly-efficient, accuracy-lossless, and engineering-tractable optimization algorithm based on offline derivation and a parallel method

exploiting the numerical characteristics of SINS/GPS. In comparison with other algorithms, the proposed method offers numerous advantages: (i) in comparison with the general algorithms, our proposed method requires little complex deduction and is easily understandable; (ii) in comparison with the RDKF optimization method, the proposed method needs little engineering approximation and would be more accurate; and (iii) in comparison with the filter based on MAM, it is free of zero-estimation operations and can deduce some stronger conclusions, thus it is more efficient. Emphasizing on the engineering simplicity, we will introduce our optimization method revolving around the classical Kalman filter and a loosely-coupled model of SINS/GPS. It is apprised here that the chosen model does not imply any constraints on the application scope of our method. Actually, the offline and parallel method, as a pure numerical optimization method, is equally applicable to the extended filter and other complex SINS/GPS models if some similar research on the special model is done. On the other hand, since there is no transformation on any models or equations in our method (the two distinct differences with the normal filtering computations are: (i) unnecessary floating-point operations are directly avoided; and (ii) de-coupled operations are processed in parallel), the precision and robustness of our method would be equal to the classical Kalman filter in theory. Owing to this fact, the efficiency rather than robustness or accuracy is focused on in our discussion later. Section 2 gives a brief account of closed-loop Kalman filtering and a widely-used loosely-coupled model in SINS/GPS. In Section 3, the details of offline derivation and the parallel method based on a block-matrix technique are described. Section 4 and Section 5 deal with the statistics on computational costs and the filter performance on error estimation.

2. Loosely-Coupled SINS/GPS Model Using the Classical Kalman Filter

Depending on the different ways of integration, SINS/GPS are classified into two modes: (i) loose mode; and (ii) tight mode. Loose mode using GPS output to adjust SINS errors is opted for in this research for its distinctive features of high redundancy and easy realization. The emphasis is placed on a common kind of loose model *i.e.*, position-velocity integration with 18 states and six observations.

2.1. State Equation

By using the indirect method, which treats the estimation variables as parametric errors instead of parameters themselves, the state equation is established in Equation (1):

$$\dot{\boldsymbol{x}}(t)=\boldsymbol{A}(t)\boldsymbol{x}(t)+\boldsymbol{G}(t)\boldsymbol{\omega}(t) \tag{1}$$

where, $\boldsymbol{x}(t)$ represents the state vector, including attitude errors $\boldsymbol{\gamma}$, velocity errors $\delta\boldsymbol{v}$, position errors $(\delta L \quad \delta\lambda \quad \delta h)^{\mathrm{T}}$, and the "colored noise" errors: gyro constant drift $\boldsymbol{\varepsilon}_c$, Markov process drift $\boldsymbol{\varepsilon}_r$, and accelerometer drift ∇_a; $\boldsymbol{\omega}(t)$ denotes the collected system noise vector, including gyro drift white noise $\boldsymbol{\omega}_g$, Markov driving white noise $\boldsymbol{\omega}_r$, and accelerometer drift white noise $\boldsymbol{\omega}_a$; $\boldsymbol{A}(t)$ is state transition matrix and $\boldsymbol{G}(t)$ is noise coefficient matrix. Combining references [16–22] with some modification, the variables of the state equation can be described by Equations (2)–(5):

$$\boldsymbol{x}(t)=\left(\gamma_x\ \gamma_y\ \gamma_z \vdots \delta v_x\ \delta v_y\ \delta v_z \vdots \delta L\ \delta\lambda\ \delta h \vdots \varepsilon_{cx}\ \varepsilon_{cy}\ \varepsilon_{cz} \vdots \varepsilon_{rx}\ \varepsilon_{ry}\ \varepsilon_{rz} \vdots \nabla_{ax}\ \nabla_{ay}\ \nabla_{az}\right)^{\mathrm{T}} \tag{2}$$

$$\boldsymbol{\omega}(t)=\left(\omega_{gx}\ \omega_{gy}\ \omega_{gz} \vdots \omega_{rx}\ \omega_{ry}\ \omega_{rz} \vdots \omega_{ax}\ \omega_{ay}\ \omega_{az}\right)^{\mathrm{T}} \tag{3}$$

$$\boldsymbol{A}(t)=\begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} & \boldsymbol{A}_{13} & \boldsymbol{C}_B^N & \boldsymbol{C}_B^N & \boldsymbol{O}_{3\times3} \\ \boldsymbol{A}_{21} & \boldsymbol{A}_{22} & \boldsymbol{A}_{23} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{C}_B^N \\ \boldsymbol{O}_{3\times3} & \boldsymbol{A}_{32} & \boldsymbol{A}_{33} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} \\ \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} \\ \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{A}_{IMU22} & \boldsymbol{O}_{3\times3} \\ \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{A}_{IMU33} \end{bmatrix}_{18\times18} \tag{4}$$

$$\boldsymbol{G}(t)=\begin{bmatrix} \boldsymbol{C}_B^N & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} \\ \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} \\ \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} \\ \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} \\ \boldsymbol{O}_{3\times3} & \boldsymbol{I}_{3\times3} & \boldsymbol{O}_{3\times3} \\ \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{I}_{3\times3} \end{bmatrix}_{18\times9} \tag{5}$$

In matrices (4) and (5), all blocks are 3-order square matrices. Blocks $\boldsymbol{O}_{3\times3}$, $\boldsymbol{I}_{3\times3}$, and $\boldsymbol{C}_B^N$ stand for the zero matrix, the identity matrix, and the attitude matrix, respectively. The other blocks are described in detail in [16–22]. In particular, we highlight here two kinds of nonzero blocks $\boldsymbol{A}_{i3(i=1,2,3)}$ and $\boldsymbol{A}_{IMUii(i=2,3)}$ of which the special single-nonzero-vector and diagonal structure may contribute to deeper optimization:

$$\boldsymbol{A}_{i3}=\begin{bmatrix} * & 0 & 0 \\ * & 0 & 0 \\ * & 0 & 0 \end{bmatrix},\ i=1,2,3\quad \boldsymbol{A}_{IMUii}=\begin{bmatrix} * & 0 & 0 \\ 0 & * & 0 \\ 0 & 0 & * \end{bmatrix},\ i=2,3 \tag{6}$$

In addition, the variance $\boldsymbol{Q}(t)$ of noise vector $\boldsymbol{\omega}(t)$ is described as:

$$\boldsymbol{Q}(t)=\begin{bmatrix} \boldsymbol{Q}_{11} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} \\ \boldsymbol{O}_{3\times3} & \boldsymbol{Q}_{22} & \boldsymbol{O}_{3\times3} \\ \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{Q}_{33} \end{bmatrix}_{9\times9} \tag{7}$$

where, block $\boldsymbol{Q}_{ii(i=1,2,3)}$ is also diagonal.

2.2. Measurement Equation

We adopt a position-velocity integration model and define the filter observation vector $\boldsymbol{z}(t)$ [16,22]:

$$\boldsymbol{z}_{obs}(t)=\begin{bmatrix} (L_{INS}-L_{GPS})R_n \\ (\lambda_{INS}-\lambda_{GPS})R_e\cos L \\ h_{INS}-h_{GPS} \\ v_{xINS}-v_{xGPS} \\ v_{yINS}-v_{yGPS} \\ v_{zINS}-v_{zGPS} \end{bmatrix}=\begin{bmatrix} R_n\delta L+N_x \\ R_e\cos L\delta\lambda+N_y \\ \delta h+N_z \\ \delta v_x+M_x \\ \delta v_y+M_y \\ \delta v_z+M_z \end{bmatrix} \tag{8}$$

Then, we define the measurement equation as follows:

$$\boldsymbol{z}(t)\triangleq\boldsymbol{H}(t)\boldsymbol{x}(t)+\boldsymbol{n}(t) \tag{9}$$

where $\boldsymbol{n}(t)$ represents the measurement noise vector; $\boldsymbol{R}(t)$ denotes the corresponding variance matrix, and $\boldsymbol{H}(t)$ is the coefficient matrix. $\boldsymbol{n}(t)$, $\boldsymbol{R}(t)$ and $\boldsymbol{H}(t)$ are described as follows according to [16–22]:

$$\boldsymbol{n}(t)=\left(N_x,N_y,N_z,M_x,M_y,M_z\right)^{\mathrm{T}} \tag{10}$$

$$\boldsymbol{R}(t)=\operatorname{diag}\left\{\sigma_{px}^2,\sigma_{py}^2,\sigma_{pz}^2,\sigma_{vx}^2,\sigma_{vy}^2,\sigma_{vz}^2\right\}=\begin{bmatrix}\boldsymbol{R}_{11} & \boldsymbol{O}_{3\times3}\\ \boldsymbol{O}_{3\times3} & \boldsymbol{R}_{22}\end{bmatrix} \tag{11}$$

$$\boldsymbol{H}(t)=\begin{bmatrix}\boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{H}_{13} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3}\\ \boldsymbol{O}_{3\times3} & \boldsymbol{I}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3}\end{bmatrix}_{6\times18} \tag{12}$$

where:

$$\boldsymbol{H}_{13}=diag\left\{R_n,R_e\cos L,1\right\} \tag{13}$$

2.3. Closed-Loop Kalman Filtering

In order to fulfill the requirement of closed-loop Kalman filtering, the state equation (Equation (1)) and measurement equation (Equation (9)) are changed to the discrete forms:

$$\boldsymbol{x}_n=\boldsymbol{\phi}_n\boldsymbol{x}_{n-1}+\boldsymbol{u}_{cn}+\boldsymbol{\Gamma}_n\boldsymbol{\omega}_n \tag{14}$$

$$\boldsymbol{z}_n=\boldsymbol{H}_n\boldsymbol{x}_n+\boldsymbol{n}_n \tag{15}$$

where, the transition matrix $\boldsymbol{\phi}_n$ is converted from $\boldsymbol{A}(t)$; the noise driving matrix $\boldsymbol{\Gamma}_n$ is converted from $\boldsymbol{G}(t)$; the coefficient $\boldsymbol{H}_n$ is converted from $\boldsymbol{H}(t)$; and the control vector $\boldsymbol{u}_{cn}=\left(\boldsymbol{\gamma}_c^{\mathrm{T}}\vdots\boldsymbol{\delta v}_c^{\mathrm{T}}\vdots\delta L_c\ \delta\lambda_c\ \delta h_c\vdots\boldsymbol{\varepsilon}_{cc}^{\mathrm{T}}\vdots\boldsymbol{\varepsilon}_{rc}^{\mathrm{T}}\vdots\nabla_{ac}^{\mathrm{T}}\right)_c^{\mathrm{T}}$ is added for real-time calibration on the parameters of SINS.

After discretization, the update of $\boldsymbol{x}_n$ is done recursively as follows:

$$\begin{aligned}&\tilde{\boldsymbol{x}}_n(-)=\boldsymbol{\phi}_n\tilde{\boldsymbol{x}}_{n-1}(+_c)\\&\tilde{\boldsymbol{z}}_n=\boldsymbol{H}_n\tilde{\boldsymbol{x}}_n(-)\\&\tilde{\boldsymbol{x}}_n(+_e)=\tilde{\boldsymbol{x}}_n(-)+\boldsymbol{K}_n(\boldsymbol{z}_{obsn}-\tilde{\boldsymbol{z}}_n)\\&\tilde{\boldsymbol{x}}_n(+_c)=\tilde{\boldsymbol{x}}_n(+_e)+\boldsymbol{u}_{cn}\end{aligned} \tag{16}$$

where, "~" represents the parameter needed estimation; $(-)$ represents the value before estimation; $(+_e)$ represents the value after estimation; $(+_c)$ represents the value after calibration with the vector $\boldsymbol{u}_{cn}$; $\boldsymbol{z}_{obsn}$ represents real observation vector; and $\boldsymbol{K}_n$ represents Kalman filtering gain.

To minimize the filtering error, state vector $\tilde{\boldsymbol{x}}_n(+_c)$ is permanently set to zero:

$$\tilde{\boldsymbol{x}}_n(+_c)\equiv\boldsymbol{0} \tag{17}$$

Substitution of Equation (17) into Equation (16), yields:

$$\boldsymbol{u}_{cn}=-\tilde{\boldsymbol{x}}_n(+_e)=-\boldsymbol{K}_n\boldsymbol{z}_{obsn} \tag{18}$$

Apparently, $\boldsymbol{u}_{cn}$ becomes the final output of the Kalman filter with control vector. As $\boldsymbol{z}_{obsn}$ is almost directly obtained by measurement, $\boldsymbol{K}_n$ appears to be the key of $\boldsymbol{u}_{cn}$ and even the filter. According to the Kalman filtering theory, $\boldsymbol{K}_n$ can be recursively computed as follows:

$$\boldsymbol{P}_n(-)=\boldsymbol{\phi}_n\boldsymbol{P}_{n-1}(+_e)\boldsymbol{\phi}_n^{\mathrm{T}}+\boldsymbol{\Gamma}_n\boldsymbol{Q}_n\boldsymbol{\Gamma}_n^{\mathrm{T}} \tag{19}$$

$$\boldsymbol{K}_n=\boldsymbol{P}_n(-)\boldsymbol{H}_n^{\mathrm{T}}\left(\boldsymbol{H}_n\boldsymbol{P}_n(-)\boldsymbol{H}_n^{\mathrm{T}}+\boldsymbol{R}_n\right)^{-1} \tag{20}$$

$$\boldsymbol{P}_n(+_e)=(\boldsymbol{I}-\boldsymbol{K}_n\boldsymbol{H}_n)\boldsymbol{P}_n(-) \tag{21}$$

where $\boldsymbol{P}_n$ is the covariance; $\boldsymbol{Q}_n$ and $\boldsymbol{R}_n$ are the discretized forms of $\boldsymbol{Q}(t)$ and $\boldsymbol{R}(t)$, respectively. In fact, Equations (18)–(21) constitute the main process of the closed-loop Kalman filter. This process is

generally subdivided into two processes: time propagation Equation (19) and measurement updating Equations (18), (20) and (21). In summary, the process flow of Kalman filter in SINS/GPS is depicted in Figure 1. Since most matrices involved in Figure 1 or Equations (19)–(21) are either 18 × 18 or 18 × 6 matrices, there will be a large-scale and time-consuming computation in each recursive cycle. Therefore, a certain optimization on the computation would be essential to the real-time properties of SINS/GPS.

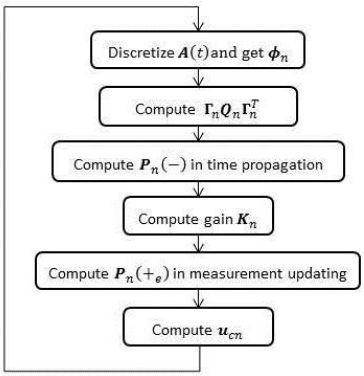

Figure 1. Iterative closed-loop Kalman filtering process.

3. Computational Optimization

Consistent with Figure 1, optimization are sequentially implemented in the updating processes of $\boldsymbol{\phi}_n$, $\boldsymbol{\Gamma}_n \boldsymbol{Q}_n \boldsymbol{\Gamma}_n^{\mathrm{T}}$, $\boldsymbol{P}_n(-)$, $\boldsymbol{K}_n$, $\boldsymbol{P}_n(+_e)$, where the most computational time are consumed.

3.1. Compute $\boldsymbol{\phi}_n$

According to [16–22], we can calculate $\boldsymbol{\phi}_n$ as follows:

$$\boldsymbol{\phi}_n = e^{\int_{t_{n-1}}^{t_n} \boldsymbol{A}(t)dt} \approx \boldsymbol{I} + T_n \boldsymbol{A}_n + \frac{T_n^2}{2!}\boldsymbol{A}_n^2 + \frac{T_n^3}{3!}\boldsymbol{A}_n^3 \tag{22}$$

Here, the 3rd order Taloy equation is selected for mainly two reasons:

(1) Equation (22) meets the requirements of most mid-high-accuracy systems;
(2) Equation (22) can deduce the universal optimization conclusion.

It's not difficult to prove that $\boldsymbol{\phi}_n$ would have a uniform numerical structure independent from which of the 3rd and higher order equation is chosen. On the other hand, lower order equations are obviously a special case of 3rd order equation. Therefore, the conclusion based on 3rd equation is suitable for the model based on the different order of equations too.

As the computation of Equation (22) is centered on the 18 × 18 matrix $\boldsymbol{A}_n = \boldsymbol{A}(t_n)$, it is necessary to inspect the numerical characteristics of $\boldsymbol{A}_n$ before computational optimization: matrix $\boldsymbol{A}_n$, in which only 64 of the total 324 elements are non-zero, is clearly a sparse matrix. A substantial reduction of useless multiplying-zero operations can be achieved by directly expanding $\boldsymbol{A}_n$ and operating every element in deduction (namely "direct derivation"), but because of the high order, direct derivation is quite complicated to handle and is not advisable in practice. Fortunately, the 3-order-block form of $\boldsymbol{A}_n$, having 23 zero blocks among the total of 36 blocks (see Equation (4)), is still a sparse matrix structurally.

Therefore, we can carry out our offline optimization using an indirect method, in which the blocks instead of the elements are treated as the atomic unit in deduction. In this way, we can largely decrease the derivation complexity and retain most of the efficiency of a direct method at the same time. For these reasons, the indirect method is applied to all the derivation processes in this paper. In addition, the nonzero blocks $\boldsymbol{A}_{i3}$ and $\boldsymbol{A}_{IMUii}$ (see Equation (6)), which need few operations in multiplication by any 3-order block (no more than 3^2 float-pointing multiplications and 2×3 additions), are also introduced to the derivation for higher efficiency.

It is a fact that manual derivation with indirect method is still a cumbersome job in view of multiple matrix multiplications (e.g., $\boldsymbol{A}_n^2$, $\boldsymbol{A}_n^3$), so we handle our derivation by a computer program instead. With the powerful symbolic-computation function of some well-known math software, e.g., MATLAB and Maple, the complicated derivation can be easily accomplished. After necessary programming, the final derivation results are shown in Equations (23) and (24):

$$\boldsymbol{A}_n^2 = \begin{bmatrix} \boldsymbol{S}_{11} & \boldsymbol{S}_{12} & \boldsymbol{S}_{13} & \boldsymbol{S}_{14} & \boldsymbol{S}_{15} & \boldsymbol{S}_{16} \\ \boldsymbol{S}_{21} & \boldsymbol{S}_{22} & \boldsymbol{S}_{23} & \boldsymbol{S}_{24} & \boldsymbol{S}_{25} & \boldsymbol{S}_{26} \\ \boldsymbol{S}_{31} & \boldsymbol{S}_{32} & \boldsymbol{S}_{33} & \boldsymbol{O}_{3\times1} & \boldsymbol{O}_{3\times3} & \boldsymbol{S}_{36} \\ \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} \\ \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{S}_{55} & \boldsymbol{O}_{3\times3} \\ \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{S}_{66} \end{bmatrix}_{18\times18} \tag{23}$$

$$\boldsymbol{A}_n^3 = \begin{bmatrix} \boldsymbol{T}_{11} & \boldsymbol{T}_{12} & \boldsymbol{T}_{13} & \boldsymbol{T}_{14} & \boldsymbol{T}_{15} & \boldsymbol{T}_{16} \\ \boldsymbol{T}_{21} & \boldsymbol{T}_{22} & \boldsymbol{T}_{23} & \boldsymbol{T}_{24} & \boldsymbol{T}_{25} & \boldsymbol{T}_{26} \\ \boldsymbol{T}_{31} & \boldsymbol{T}_{32} & \boldsymbol{T}_{33} & \boldsymbol{T}_{34} & \boldsymbol{T}_{35} & \boldsymbol{T}_{36} \\ \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} \\ \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{T}_{55} & \boldsymbol{O}_{3\times3} \\ \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{T}_{66} \end{bmatrix}_{18\times18} \tag{24}$$

Also, the computation of blocks $\boldsymbol{S}_{ij}$ and $\boldsymbol{T}_{ij}$ is described in a computerized sequence as follows:

$$\begin{aligned}
&\boldsymbol{S}_{ij} = \sum_{k=1}^{3} \boldsymbol{A}_{ik} \boldsymbol{A}_{kj}, \ i, j = 1,2,3 \\
&\boldsymbol{S}_{ij} = \boldsymbol{A}_{i(j/2-1)} \boldsymbol{C}_B^N, \ i = 1,2,3; \ \mathrm{j} = 4,6 \\
&\boldsymbol{S}_{i5} = \boldsymbol{S}_{i4}, \ i, j = 1,2 \\
&\boldsymbol{S}_{15} = \boldsymbol{S}_{15} + \left(\boldsymbol{C}_B^N \boldsymbol{A}_{IMU22} \right)_{Sim} \\
&\boldsymbol{S}_{26} = \boldsymbol{S}_{26} + \left(\boldsymbol{C}_B^N \boldsymbol{A}_{IMU33} \right)_{Sim} \\
&\boldsymbol{S}_{55} = \left(\boldsymbol{A}_{IMU22}^2 \right)_{Sim} \\
&\boldsymbol{S}_{66} = \left(\boldsymbol{A}_{IMU33}^2 \right)_{Sim} \\
&\boldsymbol{T}_{ij} = \sum_{k=1}^{3} \boldsymbol{A}_{ik} \boldsymbol{S}_{kj}, \ i = 1,2,3; \ j = 1,2,\ldots,6 \\
&\boldsymbol{T}_{26} = \boldsymbol{T}_{26} + \left(\boldsymbol{C}_B^N \boldsymbol{A}_{IMU33}^2 \right)_{Sim} \\
&\boldsymbol{T}_{55} = \left(\boldsymbol{A}_{IMU22} \boldsymbol{S}_{55} \right)_{Sim} \\
&\boldsymbol{T}_{66} = \left(\boldsymbol{A}_{IMU33} \boldsymbol{S}_{66} \right)_{Sim}
\end{aligned} \tag{25}$$

whereas all products involving block $\boldsymbol{A}_{31}$, $\boldsymbol{S}_{34}$ or $\boldsymbol{S}_{35}$ are zero blocks $\boldsymbol{O}_{3\times3}$ and need no real-time computations; products involving the special blocks, e.g., $\boldsymbol{A}_{i3}$, $\boldsymbol{S}_{i3}$ (single-nonzero –vector block), $\boldsymbol{A}_{IMUii}$, $\boldsymbol{S}_{55}$, or $\boldsymbol{S}_{66}$ (diagonal blocks), are simplified and are partly indicated in symbol $(\ )_{Sim}$.

Substituting of $\boldsymbol{A}_n$ Equation (4), $\boldsymbol{A}_n^2$ Equation (23), and $\boldsymbol{A}_n^3$ Equation (24) in $\boldsymbol{\phi}_n$ Equation (22), yields:

$$\boldsymbol{\phi}_n = \begin{bmatrix} \boldsymbol{\phi}_{11} & \boldsymbol{\phi}_{12} & \boldsymbol{\phi}_{13} & \boldsymbol{\phi}_{14} & \boldsymbol{\phi}_{15} & \boldsymbol{\phi}_{16} \\ \boldsymbol{\phi}_{21} & \boldsymbol{\phi}_{22} & \boldsymbol{\phi}_{23} & \boldsymbol{\phi}_{24} & \boldsymbol{\phi}_{25} & \boldsymbol{\phi}_{26} \\ \boldsymbol{\phi}_{31} & \boldsymbol{\phi}_{32} & \boldsymbol{\phi}_{33} & \boldsymbol{\phi}_{34} & \boldsymbol{\phi}_{35} & \boldsymbol{\phi}_{36} \\ \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{I}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} \\ \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{\phi}_{55} & \boldsymbol{O}_{3\times3} \\ \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{\phi}_{66} \end{bmatrix}_{18\times18} \tag{26}$$

whereas:

$$\begin{aligned} \boldsymbol{\phi}_{ij} &= T_n\boldsymbol{A}_{ij} + \frac{T_n^2}{2!}\boldsymbol{S}_{ij} + \frac{T_n^3}{3!}\boldsymbol{T}_{ij}, \quad i \in [1,3], j \in [1,6], i \neq j \\ \boldsymbol{\phi}_{ii} &= \boldsymbol{I}_{3\times3} + T_n\boldsymbol{A}_{ii} + \frac{T_n^2}{2!}\boldsymbol{S}_{ii} + \frac{T_n^3}{3!}\boldsymbol{T}_{ii}, \quad i = 1,2,3,5,6 \end{aligned} \tag{27}$$

Here, $\boldsymbol{\phi}_{13}$ and $\boldsymbol{\phi}_{23}$ are single-nonzero-column blocks; $\boldsymbol{\phi}_{33}$ is the sum of $\boldsymbol{I}_{3\times3}$ and single-nonzero-column block; $\boldsymbol{\phi}_{55}$ and $\boldsymbol{\phi}_{66}$ are diagonal blocks. These blocks would be specially used in other processes.

In conclusion, with offline derivation based on 3rd order block method, nearly half amount of blocks $\boldsymbol{\phi}_{ij}$ is proved to be constant blocks: $\boldsymbol{O}_{3\times3}$ or $\boldsymbol{I}_{3\times3}$ (see Equation (26)), thus, the computations on them are simply eliminated. Additionally, by using the properties of special blocks $\boldsymbol{A}_{i3}$, $\boldsymbol{S}_{i3}$, $\boldsymbol{T}_{i3}$, $\boldsymbol{A}_{jj}$, $\boldsymbol{S}_{jj}$, $\boldsymbol{T}_{jj}$ and introducing the intermediate variables $\boldsymbol{A}_n^2$, $\boldsymbol{A}_n^3$, real-time computations on the other half can be further reduced.

3.2. Compute $\boldsymbol{\Gamma}_n\boldsymbol{Q}_n\boldsymbol{\Gamma}_n^{\mathrm{T}}$

According to [16–22], the 2nd order approximate of $\boldsymbol{\Gamma}_n\boldsymbol{Q}_n\boldsymbol{\Gamma}_n^{\mathrm{T}}$ are computed as follows:

$$\boldsymbol{\Gamma}_n\boldsymbol{Q}_n\boldsymbol{\Gamma}_n^{\mathrm{T}} = \frac{T_n}{2}\left(\boldsymbol{Q}_{1n} + \boldsymbol{\phi}_n\boldsymbol{Q}_{1n}\boldsymbol{\phi}_n^{\mathrm{T}}\right) \tag{28}$$

where:

$$\boldsymbol{Q}_{1n} = \boldsymbol{G}(t)\boldsymbol{Q}(t)\boldsymbol{G}(t)^{\mathrm{T}} = \begin{bmatrix} \boldsymbol{C}_B^N\boldsymbol{Q}_{11}\left(\boldsymbol{C}_B^N\right)^{\mathrm{T}} & \boldsymbol{O}_{3\times9} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} \\ \boldsymbol{O}_{9\times3} & \boldsymbol{O}_{9\times9} & \boldsymbol{O}_{9\times3} & \boldsymbol{O}_{9\times3} \\ \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times9} & \boldsymbol{Q}_{22} & \boldsymbol{O}_{3\times3} \\ \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times9} & \boldsymbol{O}_{3\times3} & \boldsymbol{Q}_{33} \end{bmatrix}_{18\times18} \tag{29}$$

Obviously, $\boldsymbol{Q}_{1n}$ and $\boldsymbol{\Gamma}_n\boldsymbol{Q}_n\boldsymbol{\Gamma}_n^{\mathrm{T}}$ are both symmetric. Thus, for matrix $\boldsymbol{\Gamma}_n\boldsymbol{Q}_n\boldsymbol{\Gamma}_n^{\mathrm{T}}$, only its upper (or lower) triangle elements need to be computed in real time. Substitution of $\boldsymbol{Q}_{1n}$ (Equation (29)) in $\boldsymbol{\Gamma}_n\boldsymbol{Q}_n\boldsymbol{\Gamma}_n^{\mathrm{T}}$ (Equation (28)), yields:

$$\boldsymbol{\Gamma}_n\boldsymbol{Q}_n\boldsymbol{\Gamma}_n^{\mathrm{T}}=\frac{T_n}{2}\begin{bmatrix}\boldsymbol{Q}_{n11} & \boldsymbol{Q}_{n12} & \boldsymbol{Q}_{n13} & \boldsymbol{O}_{3\times3} & \boldsymbol{Q}_{n15} & \boldsymbol{Q}_{n16}\\ \boldsymbol{Q}_{n12}^T & \boldsymbol{Q}_{n22} & \boldsymbol{Q}_{n23} & \boldsymbol{O}_{3\times3} & \boldsymbol{Q}_{n25} & \boldsymbol{Q}_{n26}\\ \boldsymbol{Q}_{n13}^T & \boldsymbol{Q}_{n23}^T & \boldsymbol{Q}_{n33} & \boldsymbol{O}_{3\times3} & \boldsymbol{Q}_{n35} & \boldsymbol{Q}_{n36}\\ \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3}\\ \boldsymbol{Q}_{n15}^T & \boldsymbol{Q}_{n25}^T & \boldsymbol{Q}_{n35}^T & \boldsymbol{O}_{3\times3} & \boldsymbol{Q}_{n55} & \boldsymbol{O}_{3\times3}\\ \boldsymbol{Q}_{n16}^T & \boldsymbol{Q}_{n26}^T & \boldsymbol{Q}_{n36}^T & \boldsymbol{O}_{3\times3} & \boldsymbol{O}_{3\times3} & \boldsymbol{Q}_{n66}\end{bmatrix}_{18\times18} \tag{30}$$

where:

$$\begin{aligned}
&\boldsymbol{Q}_{nij}=\frac{T_n}{2}\left[\boldsymbol{\phi}_{i1}\boldsymbol{Q}_{11}^{'}\boldsymbol{\phi}_{j1}^{\mathrm{T}}+\boldsymbol{\phi}_{i5}\left(\boldsymbol{Q}_{22}\boldsymbol{\phi}_{j5}^{\mathrm{T}}\right)_{Sim}+\boldsymbol{\phi}_{i6}\left(\boldsymbol{Q}_{33}\boldsymbol{\phi}_{j6}^{\mathrm{T}}\right)_{Sim}\right],\ i,j=1,2,3;i\le j\\
&\boldsymbol{Q}_{n11}=\boldsymbol{Q}_{n11}+\frac{T_n}{2}\boldsymbol{Q}_{11}^{'}\\
&\boldsymbol{Q}_{nij}=\frac{T_n}{2}\left(\boldsymbol{\phi}_{ij}\boldsymbol{Q}_{j-3,j-3}\boldsymbol{\phi}_{jj}\right)_{Sim},\ i=1,2,3;\ j=5,6\\
&\boldsymbol{Q}_{n55}=\frac{T_n}{2}\left(\boldsymbol{Q}_{22}+\boldsymbol{\phi}_{55}\boldsymbol{Q}_{22}\boldsymbol{\phi}_{55}\right)_{Sim}\\
&\boldsymbol{Q}_{n66}=\frac{T_n}{2}\left(\boldsymbol{Q}_{33}+\boldsymbol{\phi}_{66}\boldsymbol{Q}_{33}\boldsymbol{\phi}_{66}\right)_{Sim}
\end{aligned} \tag{31}$$

In Equation (31), $\boldsymbol{Q}_{n55}$ and $\boldsymbol{Q}_{n66}$ are apparently diagonal blocks that may be used to simplify other processes. As stated above, we optimize the computation of $\boldsymbol{\Gamma}_n\boldsymbol{Q}_n\boldsymbol{\Gamma}_n^{\mathrm{T}}$ mainly by applying some special numerical properties to the offline derivation (e.g., the sparsity of $\boldsymbol{G}(t)$, the simplicity of diagonal blocks $\boldsymbol{Q}_{22}$, $\boldsymbol{Q}_{33}$, $\boldsymbol{\phi}_{55}$, $\boldsymbol{\phi}_{66}$, and the symmetry of $\boldsymbol{\Gamma}_n\boldsymbol{Q}_n\boldsymbol{\Gamma}_n^{\mathrm{T}}$) and much of the computation time is saved.

3.3. Compute $\boldsymbol{P}_n(-)$

For simplicity, $\boldsymbol{P}_n(-)$ and $\boldsymbol{P}_n(+_e)$ will be written in a unified form $\boldsymbol{P}_n$ (this is allowed because of that $\boldsymbol{P}_n(-)$ and $\boldsymbol{P}_n(+_e)$ are actually the same covariance matrices in different state and share the same memories). Similar to $\boldsymbol{\phi}_n$ or $\boldsymbol{\Gamma}_n\boldsymbol{Q}_n\boldsymbol{\Gamma}_n^{\mathrm{T}}$, $\boldsymbol{P}_n$ is written in 3rd-order-block-matrix form:

$$\boldsymbol{P}_n=\begin{bmatrix}\boldsymbol{P}_{n,11} & \boldsymbol{P}_{n,12} & \cdots & \boldsymbol{P}_{n,16}\\ \boldsymbol{P}_{n,21} & \boldsymbol{P}_{n,22} & \cdots & \boldsymbol{P}_{n,26}\\ \vdots & \vdots & \ddots & \vdots\\ \boldsymbol{P}_{n,61} & \boldsymbol{P}_{n,62} & \cdots & \boldsymbol{P}_{n,66}\end{bmatrix}_{18\times18} \tag{32}$$

According to its definition, $\boldsymbol{P}_n$ is a symmetric matrix:

$$\boldsymbol{P}_{n,ij}=\boldsymbol{P}_{n,ji}^{\mathrm{T}}\quad,\quad i,j=1,2\ldots6 \tag{33}$$

Thus, only the lower (or upper) triangle part needs substantive computation. Regardless of $\boldsymbol{\Gamma}_n\boldsymbol{Q}_n\boldsymbol{\Gamma}_n^{\mathrm{T}}$ at first, Equation (19) is written as:

$$\boldsymbol{P}_n(-)=\boldsymbol{\phi}_n\boldsymbol{P}_{n-1}(+_e)\boldsymbol{\phi}_n^{\mathrm{T}} \tag{34}$$

Substitution of Equation (26) and (32), yields:

$$
\begin{aligned}
\boldsymbol{P}_{n,11} &= \sum_{i=1}^{6}\left\{\boldsymbol{\phi}_{1i}\left[\sum_{j=1}^{6}\left(\boldsymbol{P}_{n-1,ij}\boldsymbol{\phi}_{1j}^{\mathrm{T}}\right)\right]_{Temp}\right\}\\
\boldsymbol{P}_{n,21} &= \sum_{i=1}^{6}\left\{\boldsymbol{\phi}_{2i}\left[\sum_{j=1}^{6}\left(\boldsymbol{P}_{n-1,ij}\boldsymbol{\phi}_{1j}^{\mathrm{T}}\right)\right]_{Temp}\right\}\\
&\vdots\\
\boldsymbol{P}_{n,61} &= \left[\boldsymbol{\phi}_{66}\sum_{j=1}^{6}\left(\boldsymbol{P}_{n-1,6j}\boldsymbol{\phi}_{1j}^{\mathrm{T}}\right)_{Rep}\right]_{Sim}\\
\boldsymbol{P}_{n,22} &= \sum_{i=1}^{6}\left\{\boldsymbol{\phi}_{2i}\left[\sum_{j=1}^{6}\left(\boldsymbol{P}_{n-1,ij}\boldsymbol{\phi}_{2j}^{\mathrm{T}}\right)\right]_{Temp}\right\}\\
\boldsymbol{P}_{n,32} &= \sum_{i=1}^{6}\left\{\boldsymbol{\phi}_{3i}\left[\sum_{j=1}^{6}\left(\boldsymbol{P}_{n-1,ij}\boldsymbol{\phi}_{2j}^{\mathrm{T}}\right)\right]_{Temp}\right\}\\
&\vdots\\
\boldsymbol{P}_{n,62} &= \left[\boldsymbol{\phi}_{66}\sum_{j=1}^{6}\left(\boldsymbol{P}_{n-1,6j}\boldsymbol{\phi}_{2j}^{\mathrm{T}}\right)_{Rep}\right]_{Sim}\\
&\vdots\\
\boldsymbol{P}_{n,66} &= \left(\boldsymbol{\phi}_{66}\boldsymbol{P}_{n-1,66}\boldsymbol{\phi}_{66}\right)_{Sim}
\end{aligned}
\tag{35}
$$

where the symbol $(\ )_{Temp}$ represents intermediate items that would be temporarily stored in memory to avoid repeated computation; the symbol $(\ )_{Rep}$ represents the repeated item that have been computed before and are available by directly accessing $(\ )_{Temp}$ and the symbol $(\ )_{Sim}$ indicates that the special properties of $\boldsymbol{\phi}_{i3}^{\mathrm{T}}$ are exploited to perform further optimization.

After Equation (35) is done, $\boldsymbol{\Gamma}_n\boldsymbol{Q}_n\boldsymbol{\Gamma}_n^{\mathrm{T}}$ should be added. Substitution of Equations (30) and (32) in Equation (19), yields:

$$\boldsymbol{P}_{n,ij} = \boldsymbol{P}_{n,ij} + \boldsymbol{Q}_{n,ij},\quad i,j = 1,2,3,5,6;\ i \geq j \tag{36}$$

Above all, the computation of $\boldsymbol{P}_n(-)$ is divided into two steps: Equations (34) and (36). As the latter has few computations, optimization mainly focuses on the former. Employing the sparsity of $\boldsymbol{\phi}_n$, the symmetry of $\boldsymbol{P}_n(-)$ and the repeatability of intermediate items, the real time of calculating Equation (34) is greatly improved.

3.4. Compute $\boldsymbol{K}_n$

Substitution of Equations (11), (12) and (32) in the inversion part of Equation (20), yields:

$$\boldsymbol{H}_n\boldsymbol{P}_n(-)\boldsymbol{H}_n^{\mathrm{T}} + \boldsymbol{R}_n = \begin{bmatrix}\boldsymbol{H}_{13}\boldsymbol{P}_{n,33}\boldsymbol{H}_{13} + \boldsymbol{R}_{11} & \boldsymbol{H}_{13}\boldsymbol{P}_{n,23}^{\mathrm{T}}\\ \boldsymbol{P}_{n,23}\boldsymbol{H}_{13} & \boldsymbol{P}_{n,22} + \boldsymbol{R}_{22}\end{bmatrix}_{6\times 6} \tag{37}$$

Since $\boldsymbol{H}_n\boldsymbol{P}_n(-)\boldsymbol{H}_n^{\mathrm{T}} + \boldsymbol{R}_n$ has a low order, few computations are needed during inversion (the complexity is $\boldsymbol{O(6}^3)$ that can be neglected in comparison with other procedures) and the details of inversion are not discussed accordingly. Here, we use the symbol $\boldsymbol{D}$ to represent the result of inversion directly:

$$\boldsymbol{D} = \left[\boldsymbol{H}_n\boldsymbol{P}_n(-)\boldsymbol{H}_n^{\mathrm{T}} + \boldsymbol{R}_n\right]^{-1} \tag{38}$$

Matrix $\boldsymbol{D}$ is easily proved to be symmetric and can be written in the 3-order-block-matrix form as:

$$\boldsymbol{D}=\begin{bmatrix}\boldsymbol{D}_{11} & \boldsymbol{D}_{12}\\ \boldsymbol{D}_{12}^{T} & \boldsymbol{D}_{22}\end{bmatrix}_{6\times 6} \tag{39}$$

Then $\boldsymbol{K}_n$ can be computed as:

$$\boldsymbol{K}_n=\begin{bmatrix}\boldsymbol{P}_{n,12}\boldsymbol{D}_{12}^{\mathrm{T}}+\boldsymbol{P}_{n,13}\boldsymbol{H}_{13}\boldsymbol{D}_{11} & \boldsymbol{P}_{n,12}\boldsymbol{D}_{22}+\boldsymbol{P}_{n,13}\boldsymbol{H}_{13}\boldsymbol{D}_{12}\\ \boldsymbol{P}_{n,22}\boldsymbol{D}_{12}^{\mathrm{T}}+\boldsymbol{P}_{n,23}\boldsymbol{H}_{13}\boldsymbol{D}_{11} & \boldsymbol{P}_{n,22}\boldsymbol{D}_{22}+\boldsymbol{P}_{n,23}\boldsymbol{H}_{13}\boldsymbol{D}_{12}\\ \vdots & \vdots\\ \boldsymbol{P}_{n,62}\boldsymbol{D}_{12}^{\mathrm{T}}+\boldsymbol{P}_{n,63}\boldsymbol{H}_{13}\boldsymbol{D}_{11} & \boldsymbol{P}_{n,62}\boldsymbol{D}_{22}+\boldsymbol{P}_{n,63}\boldsymbol{H}_{13}\boldsymbol{D}_{12}\end{bmatrix}_{18\times 6} \tag{40}$$

Define $\boldsymbol{K}_n$ as:

$$\boldsymbol{K}n=\begin{bmatrix}\boldsymbol{K}_{11} & \boldsymbol{K}_{12}\\ \boldsymbol{K}_{21} & \boldsymbol{K}_{22}\\ \vdots & \vdots\\ \boldsymbol{K}_{61} & \boldsymbol{K}_{62}\end{bmatrix}_{18\times 6} \tag{41}$$

then:

$$\boldsymbol{K}_{ij}=\boldsymbol{P}_{n,i2}\boldsymbol{D}_{j2}^{\mathrm{T}}+\boldsymbol{P}_{n,i3}\boldsymbol{H}_{13}\boldsymbol{D}_{1j},\quad i=1,2,\ldots,6;\ j=1,2 \tag{42}$$

The derivation result shows that, by using the block-matrix optimization technique, only a simple process (Equation (37), (38) and (42)) is needed in $\boldsymbol{K}_n$ updating. Compared with the complicated Equation (20) which needs multiple 18-order matrix production, our method seems much more simple and efficient in computing.

3.5. Compute $\boldsymbol{P}_n(+_e)$

Transform Equation (21) into:

$$\boldsymbol{P}_n(+_e)=\boldsymbol{P}_n(-)-\boldsymbol{K}_n\boldsymbol{H}_n\boldsymbol{P}_n(-) \tag{43}$$

Obviously, the pressure in computation mainly comes from $\boldsymbol{K}_n\boldsymbol{H}_n\boldsymbol{P}_n(-)$. Fortunately, $\boldsymbol{K}_n\boldsymbol{H}_n\boldsymbol{P}_n(-)$ is a symmetrical matrix (this is an evident inference according to Equation (43) in the condition that $\boldsymbol{P}_n(+_e)$, $\boldsymbol{P}_n(-)$ are both symmetrical) so that only its upper (or lower) triangular part needs real-time updating. Define $\boldsymbol{K}_n\boldsymbol{H}_n\boldsymbol{P}_n(-)$ as the variance's increment $\Delta\boldsymbol{P}_n$:

$$\Delta\boldsymbol{P}_n=\boldsymbol{K}_n\boldsymbol{H}_n\boldsymbol{P}_n(-)=\begin{bmatrix}\Delta\boldsymbol{P}_{11} & \Delta\boldsymbol{P}_{12} & \cdots & \Delta\boldsymbol{P}_{16}\\ \Delta\boldsymbol{P}_{21} & \Delta\boldsymbol{P}_{22} & \cdots & \Delta\boldsymbol{P}_{26}\\ \vdots & \vdots & \ddots & \vdots\\ \Delta\boldsymbol{P}_{61} & \Delta\boldsymbol{P}_{62} & \cdots & \Delta\boldsymbol{P}_{66}\end{bmatrix}_{18\times 18} \tag{44}$$

Substitution of Equations (12), (32) and (41), yields:

$$\Delta\boldsymbol{P}_{ij}=\boldsymbol{K}_{i1}\boldsymbol{H}_{13}\boldsymbol{P}_{n,3j}+\boldsymbol{K}_{i2}\boldsymbol{P}_{n,2j},\ i,j=1,2,\ldots,6;i\le j \tag{45}$$

then:

$$\boldsymbol{P}_{n,ij}=\boldsymbol{P}_{n,ij}-\Delta\boldsymbol{P}_{ij},\ i,j=1,2,\ldots,6;i\le j \tag{46}$$

In conclusion, taking advantage of the symmetry of $\boldsymbol{P}_n(+_e)$ and the sparsity of $\boldsymbol{H}_n$, we deduct a simple form of computing $\boldsymbol{P}_n(+_e)$: Equations (45) and (46). Equation (46) is apparently costless as only the

matrix subtraction is involved. Equation (45) involves matrix products, but requires no computations on most blocks of $\boldsymbol{P}_n$ (except the blocks in 2nd and 3rd rows) and proves to be highly efficient.

3.6. Parallel Computation

The optimization results (see Tables 2–6) reveal that, computation on $\boldsymbol{P}_n(-)$ costs nearly half of the processing time and becomes a bottleneck for further efficiency. The low efficiency mainly arises from that $\boldsymbol{P}_n(-)$ becomes increasingly dense and can provide few zero blocks after initial filtering cycles. In low-accuracy applications, some researchers may force the some matrix elements to be zero for low computational costs with the engineering approximate method. However, to some accuracy-demanded applications, methods with more accuracy and efficiency are still looked for. Parallel computation is one of the feasible approaches.

Before discussing the parallel method, we perform an inspection of the whole computation processes in order to clarify the dependency of every parameter on $\boldsymbol{P}_n(-)$. The numerical relationship between $\boldsymbol{P}_n(-)$ and other parameters can be consulted in Table 1.

Except for $\boldsymbol{P}_n(+_e)$, all parameters are either irrelevant to $\boldsymbol{P}_{n,ij}$ or only relevant to 2nd and 3rd block rows (columns) of $\boldsymbol{P}_n(-)$. Taking this fact into account, we can call 2nd and 3rd block rows (columns) "useful" data and treat the rest as "useless" (it is emphasized here that "useless" only refers to the requirement of most computation processes, but not that the data is really useless and needs no updates).

Furthermore, because $\boldsymbol{P}_n(-)$ is a symmetrical matrix, the "useful" blocks are only 2nd, 3rd block columns in fact. If " × " represents "useless" blocks, $\boldsymbol{P}_n(-)$ can be written as:

$$\boldsymbol{P}_n = \begin{bmatrix} \times & \boldsymbol{P}_{n,12} & \boldsymbol{P}_{n,31} & \times & \times & \times \\ \times & \boldsymbol{P}_{n,22} & \boldsymbol{P}_{n,32} & \times & \times & \times \\ \times & \boldsymbol{P}_{n,32} & \boldsymbol{P}_{n,33} & \times & \times & \times \\ \times & \boldsymbol{P}_{n,42} & \boldsymbol{P}_{n,34} & \times & \times & \times \\ \times & \boldsymbol{P}_{n,52} & \boldsymbol{P}_{n,35} & \times & \times & \times \\ \times & \boldsymbol{P}_{n,62} & \boldsymbol{P}_{n,36} & \times & \times & \times \end{bmatrix}_{18\times 18} \tag{47}$$

Table 1. Dependency of every real-time parameter on $\boldsymbol{P}_n(-)$ blocks.

ϕ_n, Q_n, $\boldsymbol{P}_n(-)$	K_n (Equations (40) and (41))	$\boldsymbol{P}_n(+_e)$ (Equation (43))	$\Delta\boldsymbol{P}_{ij}$ (Equation (45))
Irrelevant to $\boldsymbol{P}_{n,ij}$	Dependent on $\boldsymbol{P}_{n,i2}$, $\boldsymbol{P}_{n,i3}$ only	Dependent on $\boldsymbol{P}_{n,ij}$	Dependent on $\boldsymbol{P}_{n,2i}$, $\boldsymbol{P}_{n,3i}$ only

In measurement updating (Equations (18), (20) and (21)), all parameters except $\boldsymbol{P}_n(+_e)$ are decoupled with the "useless" data (Equation (19)) which belong to the time propagation. Moreover, to the coupled $\boldsymbol{P}_n(+_e)$ its time-consuming part $\Delta\boldsymbol{P}_{ij}$ is also independent of the "useless" blocks. In other words, with some appropriate modification of measurement updating and time propagation, we can produce a new decoupling mechanism of the two procedures. The modification contains three steps:

Step1. Subdivide time propagation.

Classify the blocks of $\boldsymbol{P}_n(-)$ into two kinds: "useful" data and "useless" data;

Step 2. Subdivide measurement updating.

Subdivide $\boldsymbol{P}_n(+_e)$ into two processes: computing $\Delta\boldsymbol{P}_{ij}$ (Equation (45)) and adding $\boldsymbol{P}_n(-)$ (Equation (46));

Step 3. Restructure measurement updating.

Restructure measurement updating process as $\Delta\boldsymbol{P}_{ij}$ (instead of $\boldsymbol{P}_n(+_e)$) (Equations (18) and (21)).

With the above steps, the restructured process and the "useless" data updating process are completely numerically decoupled and can be computed in parallel. In this way, the efficiency bottleneck in $\boldsymbol{P}_n(-)$ updating is broken.

Furthermore, in order to achieve the parallel computation on the "useful" data as well, we similarly subdivide the process of updating "useful" data into two steps: computing $\boldsymbol{\phi}_n\boldsymbol{P}_{n-1}(+_e)\boldsymbol{\phi}_n^{\mathrm{T}}$ (Equation (34)) and adding $\boldsymbol{\Gamma}_n\boldsymbol{Q}_n\boldsymbol{\Gamma}_n^{\mathrm{T}}$ (Equation (36)). Apparently, computing $\boldsymbol{\phi}_n\boldsymbol{P}_{n-1}(+_e)\boldsymbol{\phi}_n^{\mathrm{T}}$ and updating $\boldsymbol{\Gamma}_n\boldsymbol{Q}_n\boldsymbol{\Gamma}_n^{\mathrm{T}}$ (Equation (31), which belongs to time propagation) can be processed in parallel. In summary, the process of Kalman filtering with parallel computation is described as Figure 2.

It is pointed out here that the parallel method in this paper has a fundamental difference with some other parallel methods [23,24]. The others owe their decoupling to the so-called "mandatory delay", which may cause certain accuracy damage. On the contrary, the decoupling in this paper needs neither "mandatory delay" nor engineering approximation and is achieved mainly by introducing the "useful" data and subdividing the computation process. Essentially, it is the numerical characteristic of SINS/GPS that brings up this special decoupling. Therefore, the parallel processing in this paper is an accuracy-lossless method.

In respect of efficiency, the parallel optimization method can reduce much execution time needed in updating both $\boldsymbol{\Gamma}_n\boldsymbol{Q}_n\boldsymbol{\Gamma}_n^{\mathrm{T}}$ and the "useless" blocks (including 918 multiplications and 603 additions in updating $\boldsymbol{\Gamma}_n\boldsymbol{Q}_n\boldsymbol{\Gamma}_n^{\mathrm{T}}$, and 1089 multiplications and 1107 additions in updating "useless" blocks). In comparison with non-parallel methods, efficiency of parallel method increases by about 25%.

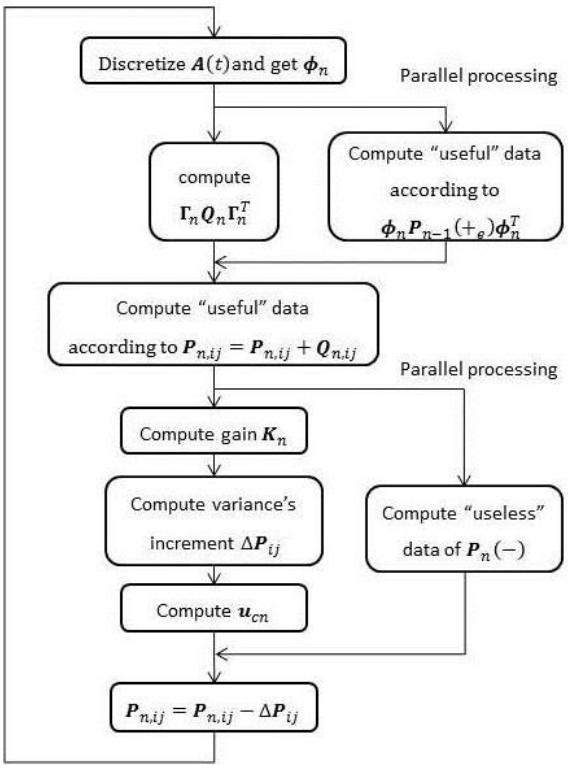

Figure 2. Filtering process with parallel computation.

4. Optimization Efficiency

According to the derivation result, we can figure out the optimization efficiency by counting and comparing the computational counts needed in both the proposed method and the existing algorithm. In detail, the discussion about counts proceeds on each parameter updating procedure, respectively. As matrix multiplication is mainly involved, the calculation will focus on the floating-point operations of multiplication and addition. Tables 2–6 will show the statistical details of each parameter. Then, Tables 7 and 8 give the global efficiency and the comparison with other general algorithms, respectively. As shown in the tables, the offline-derivation and parallel method saves about 90% of computational time and 66% of memory space, while keeping the required accuracy level. Compared with general decomposition optimization algorithms, the proposed method needs no complicated matrix theory but producing higher efficiency. Under the condition that high hardware precision has guaranteed the numerical robustness, our algorithm appears quite simple and practical.

Table 2. Optimization efficiency of ϕ_n updating process.

	Offline-Derivation Method	**Traditional Method**
Multiplication	$42\times3^3+36\times3^2+13\times3\times3^2=1809$ (I), 10%	$3\times18^3+3\times18^2=18468$ (II)
Addition	$42\times2\times3^2+9\times2\times3+13\times2\times3^2=1044$ (I), 6%	$3\times17\times18^2+3\times18^2=17496$ (II)
Memory	$32\times3^2\times8\,\text{Byte}=2.3\,\text{KB}$ (III)	$3\times18^2\times8\,\text{Byte}=7.6\,\text{KB}$ (III)

In (I), 42×3^3 ($42\times2\times3^2$), 36×3^2 ($9\times2\times3$) and $13\times3\times3^2$ ($13\times2\times3^2$) represent the counts on multiplication (addition) of items in Equation (25) without special blocks, of items in Equation (25) with special block and of items in Equation (27) respectively, and only 3^2 & 3^3 items are counted; In (II), the first item represents the counts on operations of 18-order-matrix multiplication, the second item represents that of multiplication by scalars or matrix addition; In (III), data are assumed to be double-precision (8-byte word length); the percentages represent the efficiency in comparison with the traditional method.

Table 3. Optimization efficiency of $\boldsymbol{\Gamma}_n\boldsymbol{Q}_n\boldsymbol{\Gamma}_n^{\mathrm{T}}$ updating process.

	Offline-Derivation Method	**Traditional Method**
Multiplication	$24\times3^3+18\times3^2+12\times3^2=918$, 6%	$2\times18^3+18^2+18\times6^2+6\times18^2=14580$
Addition	$24\times2\times3^2+9\times2\times3+13\times3^2=603$, 4%	$2\times17\times18^2+18^2+5\times6\times18+17\times18\times6=13716$
Memory	$13\times3^2\times8\,\text{Byte}=0.9\,\text{KB}$	$3\times18^2\times8\,\text{Byte}=7.6\,\text{KB}$

Table 4. Optimization efficiency of $\boldsymbol{P}_n(-)$ updating process.

	Offline-Derivation Method	**Traditional Method**
Multiplication	$120\times3^3+37\times3^2=3573$, 31%	$2\times18^3=11664$
Addition	$120\times2\times3^2+18\times2\times3+134\times3^2=3474$, 31%	$2\times17\times18^2+18^2=11340$
Memory	$60\times3^2\times8\,\text{Byte}=4.2\,\text{KB}$	$3\times18^2\times8\,\text{Byte}=7.6\,\text{KB}$

Table 5. Optimization efficiency of $\boldsymbol{K}_n$ updating process.

	Offline-Derivation Method	Traditional Method
Multiplication	$24\times3^3+6\times3^2=702$, 11%	$18\times18\times6+18\times6\times6+6\times18\times18+6\times18\times6=5184$
Addition	$24\times2\times3^2+12\times3^2=540$, 11%	$17\times18\times6+17\times6\times6+5\times18\times6+17\times18\times6=4824$
Memory	$16\times3^2\times8\,\text{Byte}=1.1\,\text{KB}$	$(3\times6\times18+6\times6)\times8\,\text{Byte}=2.8\,\text{KB}$

Table 6. Optimization efficiency of $\boldsymbol{P}_n(+_e)$ updating process.

	Offline-Derivation Method	Traditional Method
Multiplication	$42\times3^3+6\times3^2=1188$, 15%	$6\times18\times18+18^3=7776$
Addition	$42\times2\times3^2+42\times3^2=1134$, 15%	$5\times18^2+17\times18^2+18^2=7452$
Memory	$16\times3^2\times8\,\text{Byte}=1.1\,\text{KB}$	$(3\times6\times18+6\times6)\times8\,\text{Byte}=2.8\,\text{KB}$

Table 7. Optimization efficiency of offline-derivation and parallel method.

	Offline-Derivation & Parallel Method	Traditional Method
Multiplication	$1809+2484+702+1188=6183$ (IV), 10.7%	$18468+14580+11664+5184+7776=57672$
Addition	$1044+2475+540+1134=5193$ (IV), 9.5%	$17496+13716+11340+4824+7452=54828$
Memory	$2.3+0.9+4.2+1.1+1.1=9.6\,\text{KB}$	$7.6+7.6+7.6+2.8+2.8=28.4\,\text{KB}$

In (IV), 2484 (2475) is the operation times of "useful" blocks updating.

Table 8. Comparison of offline-derivation & parallel method and general optimization filters.

	Multiplication	Addition
SRIF filtering	$\frac{7}{6}\times18^3+36\times18^2=18468$ (V)	$\frac{7}{6}\times18^3+36\times18^2=18468$ (V)
U-D decomposing filtering	$\frac{1}{2}\times18^3+\frac{1}{2}\times18^2+10\times18^2+8\times18=6462$ (VI)	$\frac{1}{2}\times18^3+\frac{1}{2}\times18^2+9\times18^2+9\times18=6156$ (VI)
SVD filtering	$26\times18^3+78\times18^2=176904$ (VII)	$26\times18^3+78\times18^2=176904$ (VII)
offline-derivation and parallel method	$2484+702+1188=4374$ (VIII)	$2475+540+1134=4149$ (VIII)

(V) is explained in Reference [3], while (VI) in Reference [25], Reference [26] and (VII) in Reference [5]; Point out here that, (VI) relies on the precondition that $\boldsymbol{\phi}_n$ is already an upper-triangle matrix in 9-order-block form (what is the conclusion of offline derivation), otherwise, 9702 multiplications and 9396 additions are actually needed; concerning (VIII), the procedure of computing $\boldsymbol{\phi}_n$ is no longer counted as no optimizations are against $\boldsymbol{\phi}_n$ in the general algorithms.

5. Simulation

To evaluate the method validity, we design a practical loosely-coupled SINS/GPS program in CCS v5.5 (a well-known programming tool on DSP platform). The program comprises three modules:

(1) Sensor data sampling module.
(2) SINS algorithm module. This module is designed to implement the calculation of attitude, velocity and position independently (only the inertial sensors data is needed).
(3) Kalman filter module. This part is for data fusion. The system module is designed according to Section 2, and the computation process is programmed using the offline and parallel method

described in Section 3. Besides, for accuracy evaluation, the classical KF is also designed as a control group.

With the developed program, we can simulate filtering process and evaluate the performance of our method in CCS. The simulation parameters and conditions are set as follows:

(1) DSP TMS320C6722, a famous float-point CPU, is chosen in CCS as the computing device.

(2) The carrier of the navigation system is assumed to be static. With this assumption, the gyro/accelerometer would detect no valid rotate/velocity rate aside from devices noise. Therefore, noise is considered the sampling data driving the SINS/GPS program. As the carrier is static, velocity/position should stay at the zero/initial value in theory, thus, this assumption can largely facilitate the evaluation of program results.

(3) Velocity/position measured by GPS is constantly set:

$$v_{GPS} \equiv (0,0,0)^T,\ (L \quad \lambda \quad h)^T_{GPS} \equiv (0.6977686 \quad 2.0306152 \quad 90.0)^T$$

Velocity/position calculated by SINS is initialized:

$$v_{SINS} = (0,0,0)^T,\ (L \quad \lambda \quad h)^T_{SINS} = (0.6977688370723981 \quad 2.030615370778482 \quad 91.0)^T$$

(4) IMU data rate (noise frequency): 100 Hz; SINS velocity/position updating rate: 50/20 Hz; filtering rate: 10 Hz.

With the above setting, we run the program in CCS for 100 s and record the navigation outputs (e.g., the X-axis velocity and latitude). The simulation results are illustrated in Figures 3 and 4.

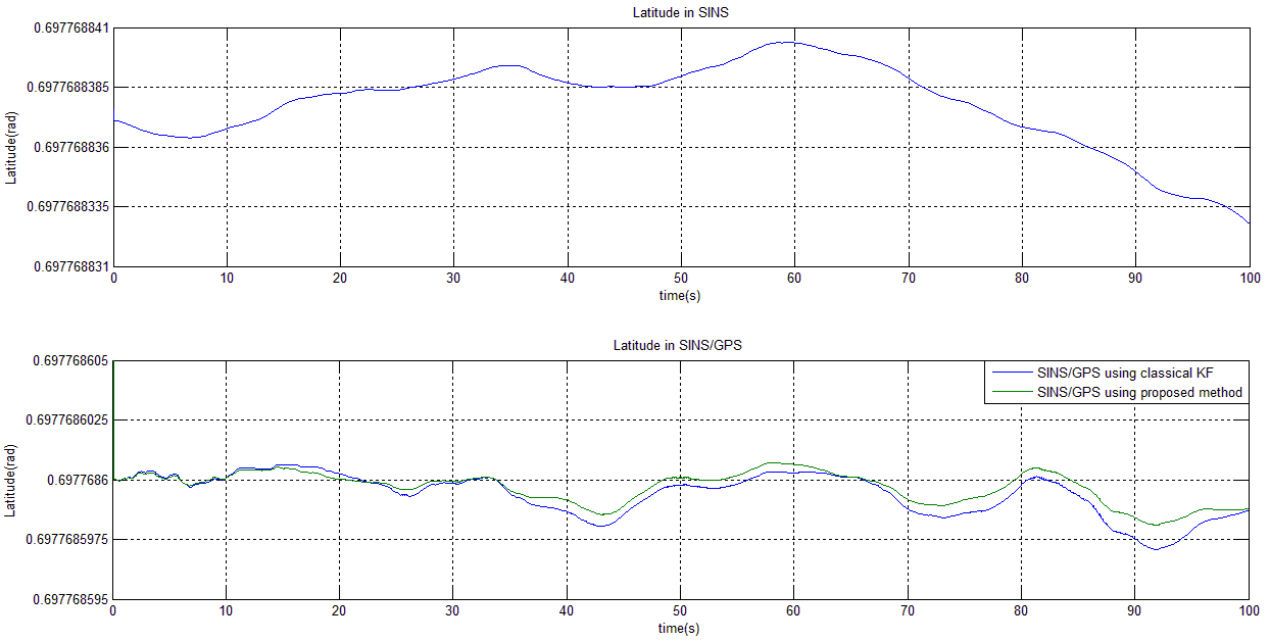

Figure 3. Latitude calculation by different methods.

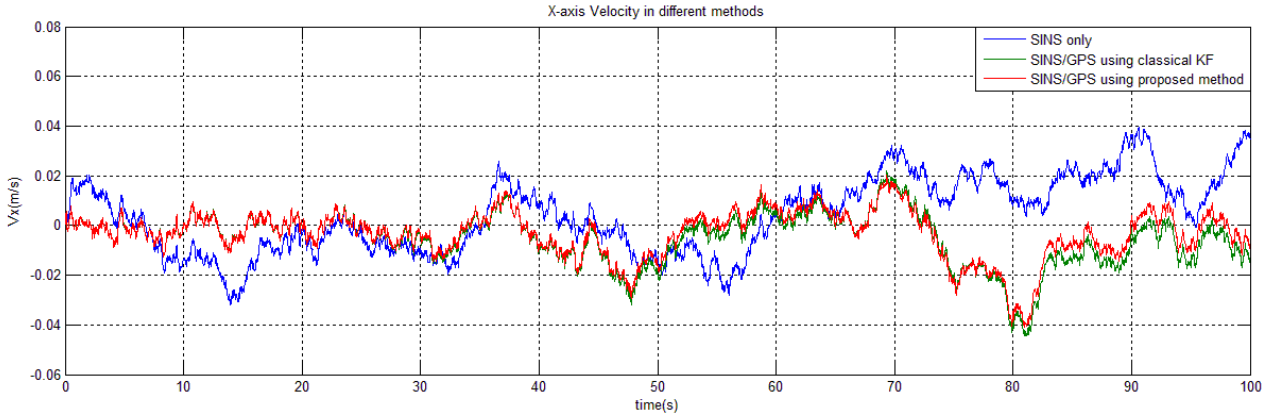

Figure 4. Velocity calculation by different methods.

The results show that the proposed method effectively restrains the unbounded error accumulation in SINS and keep the estimated variables around their mean values. On the aspect of estimation performance, our method is nearly at par to the classical KF (the proposed-method curve does not completely overlap the classical-KF one because of less calculation and less truncation errors). This conclusion agrees with the theoretical analysis: our numerical method, needing no modification of the system module and no engineering approximation, causes little damage to the estimation accuracy, but from the point of view of real-time computation, our method is much more efficient than the classical KF. In the DSP program, the proposed method needs 64,820 system clocks in each filtering cycle while classical KF needs 390,480 clocks. If a clock frequency of 200 MHz, the typical frequency of a DSP TMS320C6722 is chosen, then our method will cost only 324 μs in each filtering operation. Such a level of real-time efficiency is highly valuable to the many accurate navigation applications.

6. Conclusions

Starting from a common classical-Kalman-fitler-based SINS/GPS model (position-speed integration with 18 states and six measurements), we present an optimization algorithm based on the system's numerical characteristics. The algorithm employs a block-matrix technique in offline derivation, where special blocks (e.g., zero blocks, diagonal blocks, *etc.*) are used to simplify the calculation. In this way, plenty of invalid multiplying by zero and repeated operations are avoided offline. Furthermore, aiming at the time-consuming update of $\boldsymbol{P}_n(-)$, a novel parallel method is implemented by defining "useful" data and subdividing computational process, and the entire filtering procedure is greatly accelerated. The statistical analysis and simulation results reveal that the offline algorithm, coupled with the parallel method. can reduce the CPU processing time by 90% and the memory usage by 66%. Compared with several general decomposition-optimization algorithms, the proposed method requires no complex matrix theory rather more efficiency. The distinguished feature of the algorithm is that no modification of the system module and no engineering approximation are needed, thus it causes little harm to the base filter. It is pointed out that although the derivation is based on a specific integration model, the algorithm as a complete numerical optimization approach can be transplanted to other advanced models of SINS/GPS or other extended filters. With the powerful symbolic-operation function of the MATLAB program, researchers can manipulate formulas involving high order matrices in an easy way.

Consequently, the proposed method is an engineering-tractable approach with high efficiency and high precision for SINS/GPS integrated navigation systems.

Author Contributions

Shaoxing Hu and Shike Xu developed the overall algorithm and wrote the papers. Shike Xu coded the program and performed the simulation. Wangdu Hu analyzed the experiment data. Aiwu Zhang reviewed and revised the manuscript.

Conflicts of Interest

The authors declare no conflict of interest.

References

1. Mohinder, S.G.; Angus, P.A. *Global Navigation Satellite Systems, Inertial Navigation, and Integration*; John Wiley & Sons Inc.: New York, NY, USA, 2013.
2. Chui, C.K.; Chen, G. *Kalman Filtering with Real-Time Applications*; Tsinghua University Press: Beijing, China, 2013.
3. Bierman, G.J. A comparison of discrete linear filtering algorithms. *IEEE Trans. Aero. Electr. Syst.* **1973**, *9*, 28–37.
4. Thornton, C.L.; Bierman, G.J. UDUT covariance factorization for Kalman filtering. *Control Dyn.* **1980**, *16*, 177–248.
5. Wang, L.; Libert, G.; Manneback, P. Kalman filter algorithm based on singular value decomposition. In Proceedings of the 31st IEEE Conference on Decision and Control, Tucson, AZ, USA, 16–18 December 1992.
6. Boncelet, C.; Dickinson, B. An extension of the SRIF Kalman filter. *IEEE Trans. Autom. Control* **1987**, *32*, 176–179.
7. Tsyganova, J.V.; Kulikova, M.V. State sensitivity evaluation within UD based array covariance filters. *IEEE Trans. Autom. Control* **2013**, *58*, 2944–2950.
8. Tang, Y.; Deng, Z.; Manoj, K.K.; Chen, D. A practical scheme of the sigmapoint Kalman filter for high-dimensional systems. *J. Adv. Model. Earth Syst.* **2014**, *6*, 21–37.
9. He, X.; Chen, Y.; Iz, H.B. A reduced-order model for integrated GPS/INS. *IEEE Aerosp. Electron. Syst. Mag.* **1998**, *13*, 40–45.
10. Huang, J.; Tan, H.S. A low-order DGPS-based vehicle positioning system under urban environment. *IEEE ASME Trans. Mechatron.* **2006**, *11*, 567–575.
11. Papazoglou, M.; Tsioras, C. Integrated SAR/GPS/INS for Target Geolocation Improvement. *J. Comput. Model.* **2014**, *4*, 267–298.
12. Mutambara, A.G. *Decentralized Estimation and Control for Multisensor Systems*; CRC Press: Boca Raton, FL, USA, 1998.
13. Zhu, Q.J.; Yan, G.M. A Rapid Computation Method for Kalman Filtering in Vehicular SINS/GPS Integrated System. *Appl. Mech. Mater.* **2012**, *182*, 541–545.

14. Wei, C.; Fang, J.; Sheng, J. Fast data fusion method for integrated navigation system and hardware in loop simulation. *J. Beijing Univ. Aeronaut. Astronaut.* **2006**, *11*, 1281–1285. (In Chinese)
15. Holmes, S.; Klein, G.; Murray, D.W. An O(N²) Square Root Unscented Kalman Filter for Visual Simultaneous Localization and Mapping. *IEEE Trans. Pattern Anal. Mach. Intell.* **2009**, *31*, 1251–1263.
16. González, R.; Giribet, J.I.; Patiño, H.D. An approach to benchmarking of loosely coupled low-cost navigation systems. *Math. Comput. Model. Dyn. Syst.* **2015**, *21*, 272–287.
17. Quinchia, A.G.; Falco, G.; Falletti, E.; Dovis, F.; Ferrer, C. A comparison between different error modeling of MEMS applied to GPS/INS integrated systems. *Sensors* **2013**, *13*, 9549–9588.
18. Falco, G.; Einicke, G.A.; Malos, J.T.; Dovis, F. Performance analysis of constrained loosely coupled GPS/INS integration solutions. *Sensors* **2012**, *12*, 15983–16007.
19. Solimeno, A. Low-cost INS/GPS Data Fusion with Extended Kalman Filter for Airborne Applications. Master's Thesis, Universidade Technica de Lisboa, Lisboa, Portugal, 2007.
20. Farrell, J.A.; Barth, M. *The Global Positioning Systems and Inertial Navigation*; McGraw-Hill: New York, NY, USA, 1999.
21. Wang, H. *GPS Navigation Principle and Application*; Science Press: Beijing, China, 2010. (In Chinese)
22. Wu, T.; Ma, L. *Strapdown Inertial Navigation System Application Analysis*; National Defence Industry Press: Beijing, China, 2011. (In Chinese)
23. Brown, D.W.; Gaston, F.M.F. The design of parallel square-root covariance Kalman filters using algorithm engineering. *Integr. VLSI J.* **1995**, *20*, 101–119.
24. Rosén, O.; Medvedev, A. Parallelization of the Kalman filter on multicore computational platforms. *Control Eng. Pract.* **2013**, *21*, 1188–1194.
25. Bierman, G.J. Measurement Updating Using the U-D Factorization. *Automatica* **1976**, *12*, 375–382.
26. Bierman, G.J. Efficient Time Propagation of U-D Covariance Factors. *IEEE Trans. Autom. Control* **1981**, *26*, 890–894.