Validation and Performance Evaluation of a Simulink Inertial Navigation System Simulator

Richard Giroux * René Jr. Landry * Barrie Leach ** Richard Gourdeau ***

ABSTRACT

Great interest has been generated in low-cost inertial navigation systems (INS) in the last few years. The development of Micro-Electro-Mechanical Systems (MEMS) in the last decade has permitted mass production of devices, thereby reducing the cost of previously expensive sensors. Simulation is part of the design process of an INS. However, if we exclude proprietary simulators, simulation package tools available until now to achieve this goal are Matlab script files. To realize fast prototyping systems, to take advantage of modular design and to allow rapid real-time testing, a Simulink-based INS simulator has been created. This simulator is designed in several modules allowing point-wise improvements or modifications that do not affect the overall modules. This paper addresses the validation and the performance evaluation of the simulator. Given the results of two validation methods, it can be concluded that the Simulink simulator can now be used to design inertial navigation algorithms. Also, performance analysis of built-in Simulink integration schemes, compared with the well-known Savage two-speed integration algorithm, indicates that the automated-code integration algorithm is suitable for inertial navigation applications. In fact, for similar processing time, the position error of the 4th-order Runge–Kutta solution, compared to the Savage one-speed simplified integration

continued on page 150

* Département de génie électrique
École de technologie supérieure
1100, rue Notre Dame ouest
Montréal, QC H3C 1K3, Canada.

** Institute for Aerospace Research
National Research Council Canada
Ottawa, ON K1A 0R6, Canada.

*** École Polytechnique de Montréal
C.P. 6079, Succursale A
Montréal, QC H3C 3A7, Canada.

E-mail: rgiroux@ele.etsmtl.ca

Received 26 May 2003.

NOMENCLATURE

a	Earth major semi-axis
$[{}^{I}\boldsymbol{a}_{\mathrm{B}}]_{\mathrm{B}}$	accelerometer output
$\overline{C}_{2,0}$	dominant spherical harmonic coefficient of ellipsoid mass-attraction gravity equation
${}^{\mathrm{N}}C_{\mathrm{B}}$	rotation matrix from the body frame to the navigation frame
${}^{\mathrm{N}}C_{\mathrm{C}}$	rotation matrix from the geocentric frame to the navigation frame
${}^{\mathrm{E}}C_{\mathrm{N}}$	rotation matrix from the navigation frame to the Earth frame
${}^{\mathrm{N}}F_{\mathrm{C}}$	Earth curvature matrix
GM	gravitational constant × Earth mass
$\boldsymbol{g}_{\mathrm{p}}$	plum-bob gravity
$\boldsymbol{g}$	ellipsoid mass-attraction gravity
h	altitude
L	longitude
l	geodesic latitude
l_{c}	geocentric latitude
$\boldsymbol{r}$	position vector (from the center of the Earth to the body-frame origin)
$S\{\boldsymbol{v}\}$	anti-symmetric matrix of vector $\boldsymbol{v}$
$\overline{T}$	normalized simulation time
T_k	sampling time of high-frequency Savage algorithm
T_m	sampling time of intermediate-frequency Savage algorithm
T_n	sampling time of low-frequency Savage algorithm
$[{}^{Z}\boldsymbol{u}_{\mathrm{N}}]_{\mathrm{N}}$	unit vertical vector
U, V, W	x, y, z components of the translational velocity in the body frame
$[{}^{\mathrm{E}}\boldsymbol{v}_{\mathrm{N}}]_{\mathrm{N}}$	translational velocity in the navigation frame
$\boldsymbol{v}_{\mathrm{m}}$	velocity increment (from accelerometer)
V_{a}	airspeed
W_{A}	wander angle
α	angle of attack
$\boldsymbol{\alpha}(t)$	rotation increment (from rate gyro)

suite de la page 149

algorithm, is improved by 8% for low-sample-rate inertial measurements. The Simulink simulator will be an important tool in the design of innovative algorithms suitable for use in conjunction with MEMS inertial sensors in low-cost navigation systems. Upon completion, this software will permit rapid prototyping and easy-to-use design for low-cost GPS-aided electronic inertial navigation systems.

RÉSUMÉ

Les sytèmes de navigation inertielle (SNI) à faible coût ont généré beaucoup d'intérêt depuis quelques années. En effet, le développement des capteurs micro-électro-mécaniques (CMEM) a permis la réduction du coût des capteurs inertiels, principalement à cause de la production de masse associée à ce type de technologie. Une étape importante dans la conception des SNI est la simulation de son comportement de façon à valider l'atteinte des objectifs établis au départ. Par contre, en excluant les simulateurs privés, les outils disponibles pour concrétiser cette étape sont basés sur du code MatLab. Insatisfait des possibiltés offertes par une telle architecture, et de façon à optimiser le processus de conception des SNI, un simulateur basée sur l'application Simulink a été conçu. Cet article présente la validation et l'évaluation de la performance de ce simulateur. Par deux méthodes de validation, il est démontré que le simulateur est efficace pour la simulation d'algorithmes de navigation. De plus, la comparaison des algorithmes d'intégration internes à Simulink avec l'algorithme standard utilisé jusqu'à maintenant permet d'affirmer que les algorithmes générés automatiquement par Simulink sont appropriés pour l'utilisation dans des SNI. Dans certaines conditions, l'erreur de position avec l'utilisation d'une méthode Runge–Kutta donne une amélioration de 8 % par rapport aux algorithmes de Savage. Le simulateur basé sur Simulink sera un outil très important pour la conception et la validation d'algorithmes innovateurs utilisant des capteurs inertiels CMEM. La version finale du simulateur permettra de concevoir rapidement des algorithmes pour des SNI à faible coût hybridés avec un récepteur GPS.

β	angle of side-slip
$\boldsymbol{\beta}_m$	coning increment
β_{sc}	cone angle of the spin-cone trajectory
$\Delta\boldsymbol{v}_{scul_m}$	sculling increment
δ_l	deflection of the vertical
θ	pitch angle
$\boldsymbol{\rho}_Z^N$	vertical component of $\boldsymbol{\rho}^N$
ϕ	roll angle
$\boldsymbol{\phi}_m$	rotation vector of the body frame during one integration step
ψ	yaw angle
ψ_{tr}	true heading
ω_c	precessional rate of the spin-cone trajectory
ω_e	Earth rotational rate
ω_s	spin rate of the spin-cone trajectory
$[{}^I\boldsymbol{\omega}_B]_B$	rate gyro output
$[{}^N\boldsymbol{\omega}_B]_B$	body rate
$[{}^E\boldsymbol{\omega}_N]_N$	transport rate ($\equiv\boldsymbol{\rho}^N$)

INTRODUCTION

For many years, low-cost inertial navigation systems (INS) have been a subject of great interest. In the last decade, the development of Micro-Electro-Mechanical Systems (MEMS) has permitted mass production of devices, thereby reducing the cost of previously expensive sensors (Lawrence, 1992). Applications for such systems are numerous and many researchers around the world are now making efforts to integrate low-cost and low-precision sensors into INS with improved performance.

Simulation of INS integration algorithms is a mandatory step prior to real-time implementation to validate the design and assess the performance (Biezad, 1999). Furthermore, numerical analysis of algorithm behavior is necessary since the highly non-linear equations governing the system prohibit extensive analytical analysis of closed-form solutions.

If we exclude proprietary simulators, the only simulation package tools available until now to achieve this goal are Matlab script files (GpSoftNav, 2003). To our knowledge, only one simulator that uses Simulink for INS budget error analysis has been publicly presented (Eck et al. 2001). However, this simulator uses a flat-Earth hypothesis and neglects the Earth rate and the transport rate in the computation of inertial measurements. Even if those small contributions cannot be measured by the low-cost sensors used in the study (due to their low accuracy), their effects should be taken into account in the simulation of the true vehicle motion.

It is believed that Simulink-based simulators will form the next generation of INS algorithm validation schemes. The integration methods already implemented in Simulink permit the design of algorithms based on their continuous-time version.

Simulink's modularity and easy graphical design make it convenient for point-wise improvements and facilitate the ramp-up knowledge of future contributors (Otter and Cellier, 2000). Also, one of its powerful assets is the possibility to do rapid real-time testing through the Real-Time Workshop (RTW) suite and the xPC Target of Mathworks (or any other real-time environment compatible with RTW). This design scenario (simulation of an algorithm and real-time testing via

rapidm prototyping) reduces the delay between the validation of the algorithm and its real-time implementation.

The core function of an inertial navigation system is to integrate the acceleration and the angular-velocity measurements to produce position, velocity, and attitude solutions, based on non-linear ordinary differential equations of motion. Savage's two-speed integration algorithm (Savage, 1998a, 1998b) is considered nowadays as the state-of-the-art integration scheme (Litmanovich et al., 2000). However, how does the resulting integration algorithm from the Simulink environment compare to the "conventional" approach? Although many people state that one-speed higher order integration algorithms are now applicable, due to modern-day computer capabilities (e.g., Savage, 1998a; Litmanovich et al., 2000; Chatfield, 1997), only a few seem to have actually implemented them. In this respect, Eck et al. (2001) implemented a one-cycle integration algorithm and neglected the coning and sculling terms in the overall solution. Their paper stated that several algorithms and integration rates were compared, but no results on that topic were shown.

This paper will address two significant aspects of a new generic Simulink INS simulator. First, the validation of the inertial-data -generation module (based on the ideal trajectory) will be performed. This validation will be twofold: an input–output validation, and a comparison with another, already validated, simulator used at the National Research Council, Ottawa (Leach and Hui, 1999). Second, and more importantly, the performance of the Simulink integration schemes (for direct implementation) will be compared to the conventional approach.

The first section will introduce the simulator, with basic mathematics about the kinematics that constitute the core of the inertial measurement and trajectory generation. Then, the software architecture will be presented and the functionality of each module will be described. The next section will validate the inertial-data generation module. Finally, the basics of the INS integration algorithms, and performance analysis of those algorithms, will be given.

Mathematical Background

The following mathematical conventions are introduced to evaluate sensor measurements, i.e., angular rates and linear accelerations; and to determine the kinematic variables associated with the desired trajectory. The coordinate frames used are the same as the ones defined by Savage (1998a), and shown in **Figures 1 and 2**.

- Earth-Centered Inertial (I) frame: non-rotating inertial coordinate frame used as a reference for angular rotation, axis *Y* is directed towards the celestial north pole, axis *X* towards the vernal equinox, and axis *Z* is set to define rectangular coordinates;
- Earth-Centered Earth-Fixed (E) frame: Earth-fixed frame used for position location definition, the *Y* axis is parallel to the Earth's polar axis, the *X* axis passes through the Greenwich meridian, and the *Z* axis is perpendicular to the *X* and *Y* axes;
- Geographic (G) frame: locally level geographic frame defined with its *Z* axis upward along the local geodetic vertical, *Y* axis north (horizontal) and *X* axis east (horizontal);
- Navigation (N) frame: locally level frame used to integrate acceleration into velocity, defining the orientation of the local

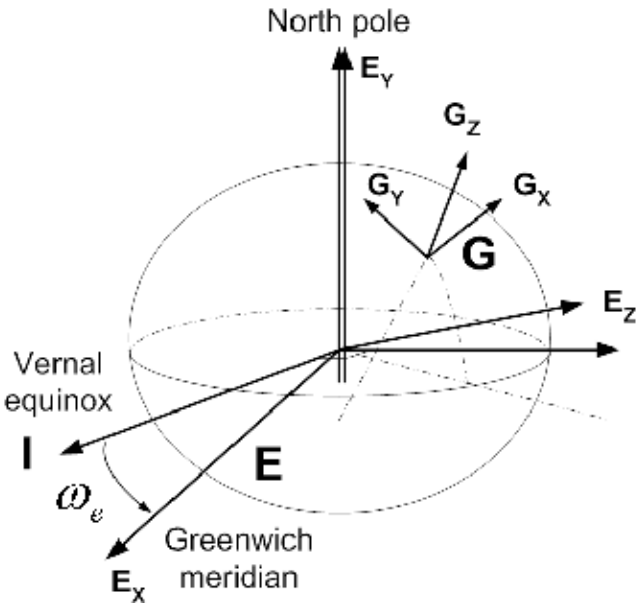

Figure 1. Inertial Frame (I), Earth-Centered Earth-Fixed Frame (E), and Local Geographic Frame.

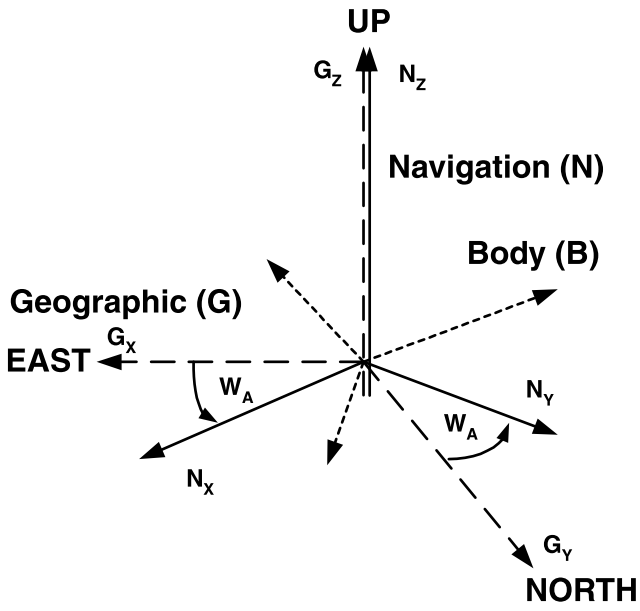

Figure 2. Local Geographic Frame (G), Navigation Frame (N), and Body Frame (B).

vertical in the E frame, and as a reference for describing the strapdown sensor coordinate frame orientation. It is defined with its *Z* axis upward along the local geodetic vertical, and rotated with respect to the *Z* axis of the Geographic frame by the wander angle (W_A);

- Body (B) frame: vehicle-fixed frame with axes parallel to nominal right-handed orthogonal sensor input axes. Without loss of generality, the sensor frame is assumed to be coincident with the body frame for the rest of the paper.

To express the equations of motion, the following conventions are used:

- ${}^{A}C_{B}$: matrix of rotation (direction cosine matrix) from the frame B to the frame A
- $[{}^{A}\boldsymbol{v}_{B}]_{C}$: the velocity vector of the frame B with respect to (w.r.t.) the frame A, measured w.r.t. the frame C
- $S\{[{}^{A}\boldsymbol{v}_{B}]_{C}\}$: the anti-symmetric matrix of the vector $[{}^{A}\boldsymbol{v}_{B}]_{C}$, where

$$S\{[{}^{A}\boldsymbol{v}_{B}]_{C}\} = \begin{pmatrix} 0 & -v_z & v_y \\ v_z & 0 & -v_x \\ -v_y & v_x & 0 \end{pmatrix}$$

Simulator Architecture

The software simulator is designed in a modular fashion to allow easy reconfiguration of all sub-systems. **Figure 3** shows a blockset of the simulator. Only the trajectory generation, the inertial-data generation, and the inertial navigation algorithm modules are addressed in this paper. In fact, for the validation and performance evaluation of the simulator, the model used for the sensors is simply a sampling of the ideal inertial variables. Also, the GPS model will be of use only while evaluating algorithm performance with respect to Kalman-filtering implementation.

Trajectory Generation

The trajectory generation phase of the simulator is separated into two sequences: one off-line and a second on-line.

Off-line Data Products

The trajectory generation is first based on a set of data points representing a flight profile, e.g., airspeed (velocity of the vehicle w.r.t the air mass), heading, attitude, angle of attack, and angle of side-slip. The last two parameters can be neglected for a first approximation, by letting the vehicle longitudinal axis be parallel to the airspeed vector. **Table 1** shows a typical set of data points for the flight profile and represents the step No. 1 and the variable set No. 1 in **Figure 4**.

Then, the first set of variables is used to obtain trajectory information at a given sampling rate using cubic splines (step No. 2 in **Figure 4**). Cubic splines are also used to compute the rate of change of those variables. Given the order of the interpolation, the rate of change of the variables is a continuous function. Hence, the variable set No. 2 is composed of:

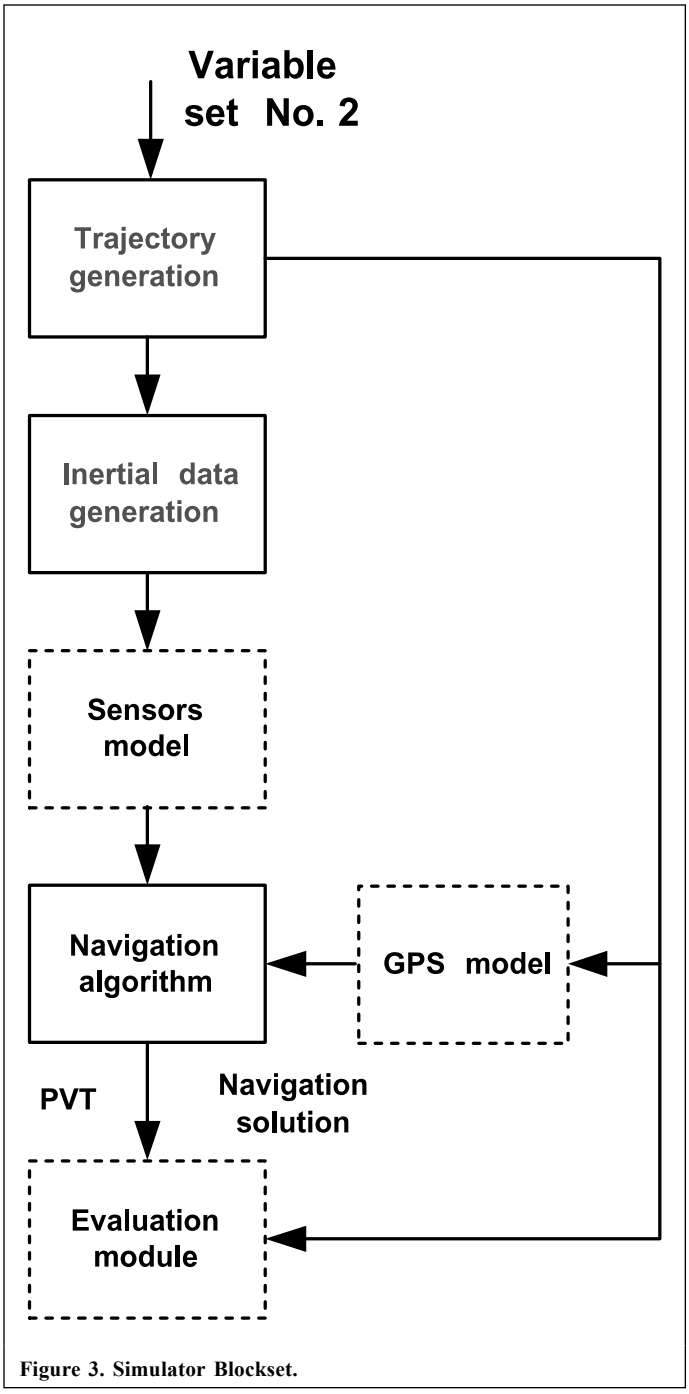

Figure 3. Simulator Blockset.

- the Euler angles (ϕ, θ, ψ)
- the Euler angle rates ($\dot{\phi}$, $\dot{\theta}$, $\dot{\psi}$)
- the airspeed (V_a)
- the rate of change of the airspeed ($\dot{V}_a$)
- the angle of attack and angle of side-slip, where applicable (α, β)

Table 1. Example of Flight Profile Data.

Time (s)	Heading (°)	Pitch angle (°)	Roll angle (°)	Airspeed (m/s)
0.0	45.0	0.0	0.0	0.0
1.0	45.0	0.0	0.0	5.0
2.0	45.0	0.0	0.0	10.0
14.0	45.0	0.0	0.0	70.0
15.0	45.0	0.0	0.0	75.0
⋮	⋮	⋮	⋮	⋮
181.0	45.0	9.5	0.0	76.2
182.0	45.0	9.0	0.0	77.5
198.0	45.0	4.5	0.0	97.5
⋮	⋮	⋮	⋮	⋮

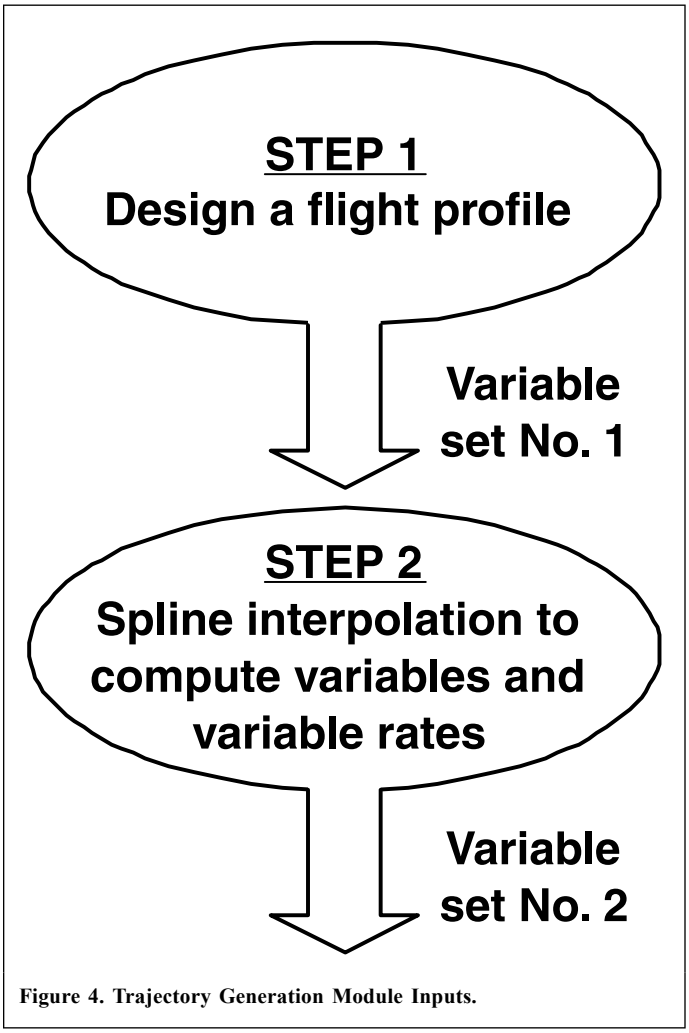

Figure 4. Trajectory Generation Module Inputs.

- the angle of attack rate and angle of side-slip rate, where applicable ($\dot{\alpha}$, $\dot{\beta}$)

On-line Simulink Module

After the off-line data processing, the Simulink module "Trajectory generation" (in **Figure 3**) uses this information to compute several kinematic variables to be used in the "Inertial-data -generation" module, starting with the velocity in the body (B) frame

$$[{}^{E}\boldsymbol{v}_{N}]_{B} = [{}^{E}\boldsymbol{v}_{B}]_{B} = \begin{pmatrix} U \\ V \\ W \end{pmatrix} = \begin{pmatrix} V_a c\beta c\alpha \\ V_a s\beta \\ V_a c\beta s\alpha \end{pmatrix} \tag{1}$$

where $c\beta$ and $s\beta$ are, respectively, the cosine and the sine of the β angle (and so on for other angles). Also, $U = V_a$ and $V = W = 0$ for zero angle of attack and angle of side-slip.

The velocity in the navigation (N) frame is then calculated from the velocity in the body frame

$$[{}^{E}\boldsymbol{v}_{N}]_{N} = {}^{B}C_{N}^{T}\,[{}^{E}\boldsymbol{v}_{N}]_{B} \tag{2}$$

where ${}^{B}C_{N}^{T} = f(\phi, \theta, \psi_{tr}, W_A)$ is the transpose of the well-known direction cosine matrix.

The rate of change of the velocity in the navigation frame is also needed. It is calculated as follows:

$$[{}^{E}\dot{\boldsymbol{v}}_{N}]_{N} = {}^{B}\dot{C}_{N}^{T}\,[{}^{E}\boldsymbol{v}_{N}]_{B} + {}^{B}C_{N}^{T}\,[{}^{E}\dot{\boldsymbol{v}}_{N}]_{B} \tag{3}$$

$$= {}^{B}C_{N}^{T}\; S\{[{}^{N}\boldsymbol{\omega}_{B}]_{B}\}\,[{}^{E}\boldsymbol{v}_{N}]_{B} + {}^{B}C_{N}^{T}\,[{}^{E}\dot{\boldsymbol{v}}_{N}]_{B} \tag{4}$$

with

$$[{}^{E}\dot{\boldsymbol{v}}_{N}]_{B} = \begin{pmatrix} \dot{V}_a c\beta c\alpha - V_a\dot{\beta}s\beta c\alpha - V_a c\beta\dot{\alpha}s\alpha \\ \dot{V}_a s\beta + V_a\dot{\beta}c\beta \\ \dot{V}_a c\beta s\alpha - V_a\dot{\beta}s\beta s\alpha + V_a c\beta\dot{\alpha}c\alpha \end{pmatrix} \tag{5}$$

from the derivative chain rule applied to Equation (1) and $[{}^{N}\boldsymbol{\omega}_{B}]_{B}$ given by Equation (6).

$$[{}^{N}\boldsymbol{\omega}_{B}]_{B} = \begin{pmatrix} 1 & 0 & -s\theta \\ 0 & c\phi & s\phi c\theta \\ 0 & -s\phi & c\phi c\theta \end{pmatrix} \begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} \tag{6}$$

Inertial-data-Generation Module

Angular Velocity

The angular velocity of the body frame w.r.t. the inertial frame measured in the body frame (the rate gyro measurements) is given by Equation (7)

$$[{}^{I}\boldsymbol{\omega}_{B}]_{B} = {}^{B}C_{N}\,{}^{N}C_{E}[{}^{I}\boldsymbol{\omega}_{E}]_{E} + {}^{B}C_{N}[{}^{E}\boldsymbol{\omega}_{N}]_{N} + [{}^{N}\boldsymbol{\omega}_{B}]_{B} \tag{7}$$

where ${}^{N}C_{E}$ is the position matrix (refer to Equation (12) in section "Inertial navigation integration algorithm"). The angular velocity can be decomposed into three components: the attitude rate given by the Euler angle rates (Equation (6)), the Earth rate $[{}^{I}\boldsymbol{\omega}_{E}]_{E} = [\,0\;\omega_e\;0]^{T}$, and the transport rate $\boldsymbol{\rho}^{N} \equiv [{}^{E}\boldsymbol{\omega}_{N}]_{N}$. The complete expression for $\boldsymbol{\rho}^{N}$ can be found in Savage (1998a).

Specific Force

The specific force is the total non-gravitational acceleration of the body frame w.r.t. the inertial frame measured in the body frame, and it is the output of the accelerometers. It is calculated by inverting the velocity-rate equation (see Equation (11)) and is given in Equation (8). The specific force is composed of the kinematic acceleration, the negative of the plumb-bob gravity, and the Coriolis acceleration, all of them measured in the navigation frame.

$$[{}^{I}\boldsymbol{a}_{B}]_{B} = {}^{B}C_{N}([{}^{E}\dot{\boldsymbol{v}}_{N}]_{N} - [\mathbf{g}_{p}]_{N} + (S\{\boldsymbol{\rho}^{N}\} + 2S\{[{}^{I}\boldsymbol{\omega}_{E}]_{N}\})[{}^{E}\boldsymbol{v}_{N}]_{N}) \quad (8)$$

The plumb-bob gravity is the sum of the mass attraction of a fixed-Earth and the centripetal acceleration.

$$[\boldsymbol{g}_{p}]_{N} = [\boldsymbol{g}]_{N} - {}^{N}C_{E}(S\{[{}^{I}\boldsymbol{\omega}_{E}]_{E}\}\, S\{[{}^{I}\boldsymbol{\omega}_{E}]_{E}\}\, \boldsymbol{r}) \quad (9)$$

where $[\boldsymbol{g}]_N$ is the acceleration based on the mass attraction. Refer to Chatfield (1997) for the details of the calculation of $[\boldsymbol{g}]_N$. The position vector $\boldsymbol{r}$ is the vector from the center of the Earth to the vehicle position.

Simulator Validation

The validation of the inertial-data -generation and trajectory-generation modules of the simulator will be performed in two steps. First, the ideal continuous inertial navigation algorithm will be implemented and the output will be compared to the initial designed flight path. This will highlight any coding errors. A second validation will be based on a comparison of the outputs of the inertial generation module with the outputs of a similar already validated inertial simulator from the National Research Council's Institute for Aerospace Research (Leach and Hui, 1999). The validation flight profile is depicted in Appendix A.

Input–output validation

The first method of comparison is based on the difference between the input to the trajectory-generation module and the output from the ideal continuous inertial navigation algorithm. The continuous version of the integration algorithm is represented by Equations (10) to (13). **Figure 5** shows a graphical representation of the validation scheme.

To evaluate discrepancies between the two block-functions, no sensor errors nor sampling rate are used in the sensor module. Moreover, the integration scheme in the Simulink environment is set to a variable-step, high-order, modified Runge–Kutta method (refer to section "the ODE integration suite of the Simulink environment" for more information).

No significant errors have been found in any variables. As a result, the maximum position error magnitude was 10^{-7} m for the flight profile. However, since this validation scheme only guarantees that the trajectory generation and inertial-data

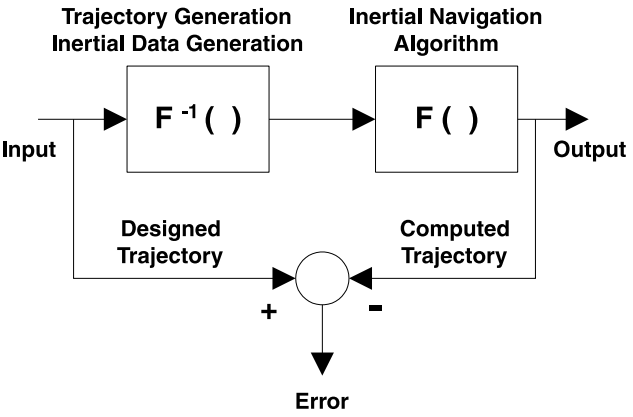

Figure 5. Input–Output Validation.

-generation modules are the exact inverse of the inertial navigation algorithm, another type of validation has to be done. This second validation procedure is explained in the next section.

Validation by Model Comparison

The second method used to validate the Simulink simulator is to compare its inertial-data -generation-module outputs to another, already validated, simulator. **Figure 6** illustrates the implementation of the validation structure.

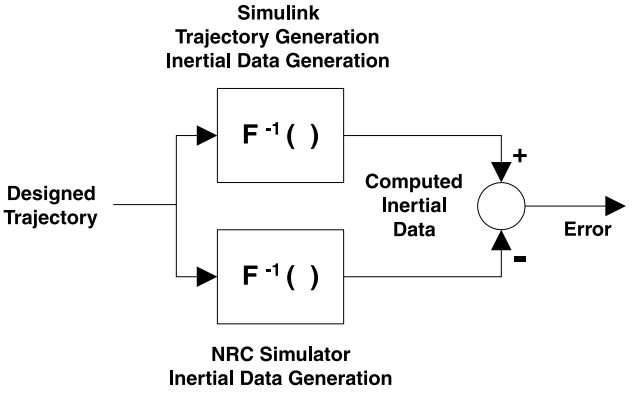

Figure 6. Validation by Model Comparison.

It should be noted that there is a difference in the gravity-vector models being used. The NRC's simulator uses a variant of the International Gravity Formula (Li and Goetze, 2001) while the Simulink simulator uses the ellipsoid mass-attraction gravity formulation (Chatfield, 1997). Because of this difference, the accelerometer outputs have been compared without the computation of the gravity vector. **Figure 7** illustrates that difference and **Figure 8** points out the difference in angular velocity.

It can be seen from these figures that small discrepancies exist in the inertial-data generation (the X and Z axes are acceleration and the Y axis angular rate). In fact, these errors are only present at the beginning and at the end of the trajectory

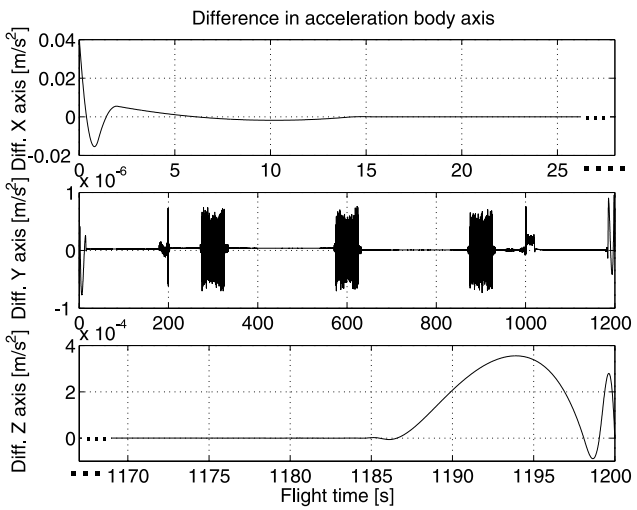

Figure 7. Difference in Acceleration from Both Simulators.

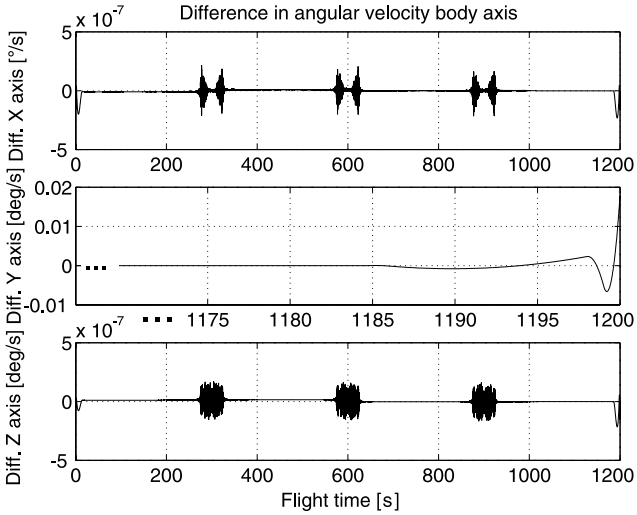

Figure 8. Difference in Angular Velocity from Both Simulators.

and are mainly due to different cubic spline algorithms, resulting in a slightly different rate of change of velocity and rate of change of attitude.

Figure 9 shows the real altitude computation. It can be seen that there are some differences between the true altitude computed from the NRC simulator and the Simulink simulator. In addition to the different integration algorithms used in both simulators, which may cause slightly different outcomes, the cubic spline algorithms have an influence on the trajectory-profile computation accuracy (true altitude, longitude, and latitude). As for the errors found in the acceleration and rate gyro measurements (**Figures 7** and **8**), the different cubic spline algorithms gave different extrapolation values during the dynamic parts of the flight profile (time tags: 200, 300, 600, 900, and 1000 s) that are transmitted to the trajectory generation variables through Equations (1) and (2).

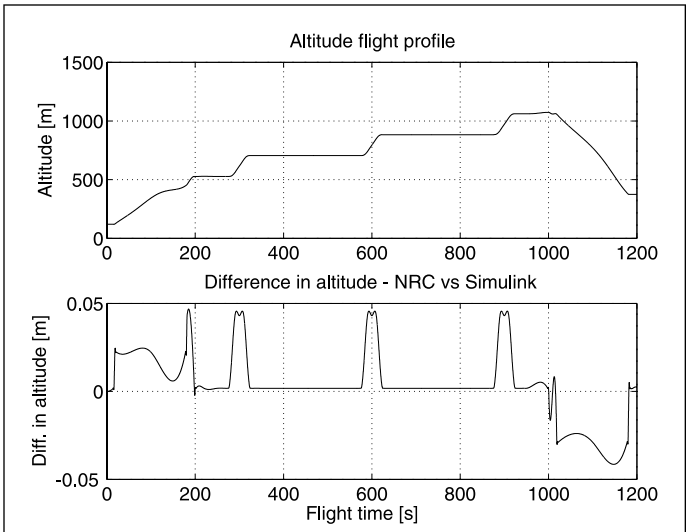

Figure 9. Difference in Altitude from Both Simulators.

It should be clarified that the real trajectory computation of the NRC simulator is exact for the generated inertial sensor measurements, as the real trajectory computation of the Simulink simulator is for its inertial-data sets. Because of their respective intrinsic algorithms, one should not have expected perfect match between their outcomes. However, given that the resulting trend is highly similar, it can be concluded that the Simulink simulator is now ready for practical use in the design of inertial navigation algorithms.

Inertial Navigation Integration Algorithm

Most of the algorithms used to integrate the acceleration and the angular velocity measurements to produce position, velocity, and attitude solutions are based on the following ordinary differential equations of motion:

$$ {}^{N}\dot{C}_{B} = {}^{N}C_{B}\, S\{[{}^{I}\boldsymbol{\omega}_{B}]_{B}\} - S\{[{}^{I}\boldsymbol{\omega}_{N}]_{N}\}\, {}^{N}C_{B} \tag{10} $$

$$ [{}^{E}\dot{\boldsymbol{v}}_{N}]_{N} = {}^{N}C_{B}[{}^{I}\boldsymbol{a}_{B}]_{B} + [\boldsymbol{g}_{p}]_{N} - \tag{11} $$

$$ (S\{[{}^{E}\boldsymbol{\omega}_{N}]_{N}\} + 2S\{[{}^{I}\boldsymbol{\omega}_{E}]_{N}\})[{}^{E}\boldsymbol{v}_{N}]_{N} $$

$$ {}^{E}\dot{C}_{N} = {}^{E}C_{N}\, S\{[{}^{E}\boldsymbol{\omega}_{N}]_{N}\} \tag{12} $$

$$ \dot{h} = [{}^{Z}\boldsymbol{u}_{N}]_{N} \cdot [{}^{E}\boldsymbol{v}_{N}]_{N} = v_{Z}^{N} \tag{13} $$

with the inputs being $[{}^{I}\boldsymbol{\omega}_{B}]_{B}$ (rate gyro) and $[{}^{I}\boldsymbol{a}_{B}]_{B}$ (accelerometer). In this formulation, the position update (Equation (12)) is based on the position matrix ${}^{E}C_{N}$, by which the latitude and the longitude can be computed from its components. However, position updates can also be performed by integrating the rate of change of the position vector $[{}^{E}\dot{\boldsymbol{r}}_{N}]_{N}) = [{}^{E}\boldsymbol{v}_{N}]_{N}$ and using geographic position equivalent

transformations, or directly by integrating the rate of change of the latitude $\dot{l} = f([^{E}\boldsymbol{v}_{N}]_{N})$ and the longitude $\dot{L} = f([^{E}\boldsymbol{v}_{N}]_{N})$.

During the early days of INS integration-algorithm design, major trade-offs were done between solution accuracy and computation load. The two approaches were high-speed first-order and low-speed high-order digital algorithms. In the mid-1960s, Savage proposed a two-speed approach: a simple high-speed first-order algorithm that feeds a moderate-speed high-order algorithm. This algorithm (Savage, 1998a, 1998b) is now considered the state-of-the-art for inertial navigation integration and is used in most of the modern day strapdown inertial systems for aircraft (Litmanovich et al., 2000).

However, the throughput limitations of early embedded computers are now becoming irrelevant with the increase of computing capabilities of modern technology (Savage, 1998a; Litmanovich et al., 2000). As an example, the PC104 standard makes use of the same development tools as the full-size PCs. Although only about $4'' \times 4''$, PC104 boards are very powerful for their size (1 in = 2.54 cm). These products are designed for minimal power consumption, small foot print, modularity, expendability, and ruggedness — basic needs of an embedded system. Hence, more computationally demanding algorithms (and, therefore, more accurate) are nowadays potential candidates for INS integration schemes. The next sections will compare one-speed integration schemes included in the Simulink environment and the two-speed approach of Savage.

The Two-Speed Integration Scheme of Savage

The complete set of equations for the two-speed integration algorithm of Savage can be found in two comprehensive papers (Savage, 1998a, 1998b). Only the basic ideas will be presented here. **Figure 10** shows the implementation blockset of the two-speed integration algorithm.

The algorithm solves the differential equations based on the exact form. As an example, the velocity-update equation will be presented. Equation (11) shows the continuous version of the velocity update. By taking the discrete counterpart, it gives

$$[^{E}\boldsymbol{v}_{N}]_{N}(m) = [^{E}\boldsymbol{v}_{N}]_{N}(m-1) + \int_{t_{m-1}}^{t_m} ({}^{N}C_{B}[^{I}\boldsymbol{a}_{B}]_{B} + [\boldsymbol{g}_{p}]_{N} - \mathcal{M}[^{E}\boldsymbol{v}_{N}]_{N})\,dt \tag{14}$$

where the matrix $\mathcal{M}$ is given by

$$\mathcal{M} = S\{[^{E}\boldsymbol{\omega}_{N}]_{N}\} + 2S\{[^{I}\boldsymbol{\omega}_{E}]_{N}\} \tag{15}$$

The integral term of Equation (14) can be split into two integrals

$$[^{E}\boldsymbol{v}_{N}]_{N}(m) = [^{E}\boldsymbol{v}_{N}]_{N}(m-1) + \int_{t_{m-1}}^{t_m} ([\boldsymbol{g}_{p}]_{N} - \mathcal{M}[^{E}\boldsymbol{v}_{N}]_{N})\,dt + \int_{t_{m-1}}^{t_m} ({}^{N}C_{B}[^{I}\boldsymbol{a}_{B}]_{B})\,dt \tag{16}$$

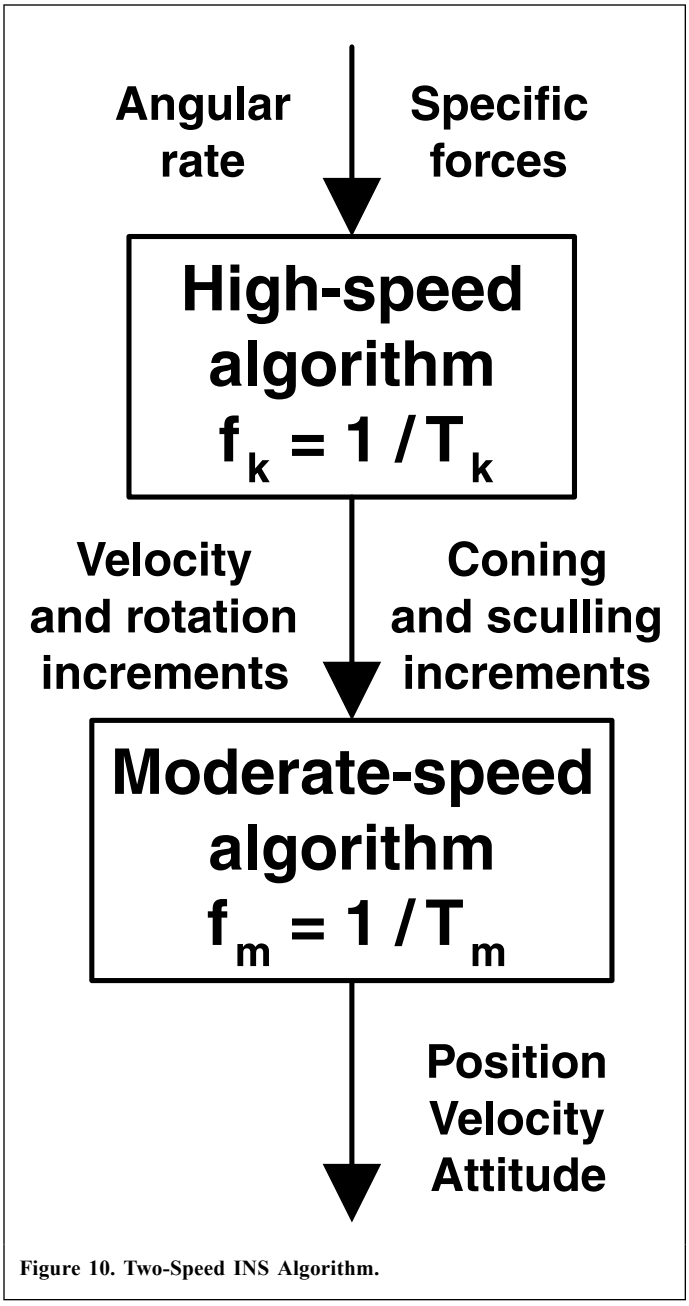

Figure 10. Two-Speed INS Algorithm.

The first integral is called the gravity/Coriolis correction-velocity increment and can be integrated using mid-interval extrapolation and a simple trapezoidal integration algorithm. The second integral of Equation (16) is the velocity increment from the accelerometer and can be computed as follows:

$$[^{I}\Delta\boldsymbol{v}_{B}]_{N}(m) = \int_{t_{m-1}}^{t_m} ({}^{N}C_{B}[^{I}\boldsymbol{a}_{B}]_{B})\,dt) \tag{17}$$

$$\approx {}^{N}C_{B}(\boldsymbol{v}_{m} + \frac{1}{2} S\{\alpha_{m}\}\boldsymbol{v}_{m} + \Delta\boldsymbol{v}_{\mathrm{scul}_{m}}) \tag{18}$$

where

$\boldsymbol{v}_{\mathrm{m}}$ is the velocity increment from accelerometer; $\frac{1}{2} S\{\boldsymbol{\alpha}_{\mathrm{m}}\}\boldsymbol{v}_{\mathrm{m}}$ is the velocity rotation compensation increment; and $\Delta\boldsymbol{v}_{\mathrm{scul}_{\mathrm{m}}}$ is the sculling increment.

The sculling increment takes into account combined dynamic angular-rate/specific-force motion and is calculated in the high-speed portion of the algorithm. The attitude update equation also has a high-frequency component due to the coning effect. The coning increment takes into account the rotation of the angular rate vector and is a compensation term for the rotation-matrix equivalent-rotation vector. Those components take into account high-frequency input and characteristic motion effects in the inertial measurements and are computed at a high rate $f_k = 1/T_k$.

On the other hand, the moderate-speed algorithm ($T_m > T_k$) computes the attitude and velocity solution. The value of the high-frequency components at the m time step are then used in the moderate-speed portion of the algorithm (Equation (18) into Equation (16)) for the velocity update; the attitude update equation is not elaborated herein.

The position solution can be determined at an even slower rate ($n > m > l$), depending on the application. However, to facilitate the implementation of Savage's algorithm in Simulink, only one high-speed cycle will be implemented. In other words, the position matrix, the local-level orientation, the velocity, and attitude-direction cosine matrix are updated at the same time as the coning and sculling increments (Savage, 1998a, 1998b): ($T_k = T_m = T_n$). Also, the trapezoidal position algorithm will be preferred to the high-resolution position algorithm.

The ODE Integration Suite of the Simulink Environment

There are many ways to discretize continuous-time differential equations (Santina et al., 2000). The use of the state transition matrix is the most accurate but not practical in our application. However, the integration of continuous equations within Simulink is based on the approximation method. The approximation method refers to Equation (14). Instead of splitting the integrand into several integration rates (like the Savage algorithm does), the integral of Equation (14) is performed using one integration algorithm, preferably of high order. Many integration schemes are available with Simulink (Shampine and Reichelt, 1997). Only the fixed-step methods will be evaluated, since they are the ones compatible with the Simulink Real-Time Workshop application that creates automated compiled code. The five methods of integration are

- ode5: the Dormand–Prince formula
- ode4: the fourth-order Runge–Kutta formula
- ode3: the Bogacki–Shampine formula
- ode2: Heun's method
- ode1: Euler's method

The Bogacki–Shampine formula, the 4th-order Runge–Kutta, and the Dormand–Prince formula are all variants of the classic Runge–Kutta method and are characterized as high-order integration methods. On the other hand, the Euler method is the basic integration scheme of first order. In between, Heun's method, also known as the Improved Euler or the 2nd-order Runge–Kutta, gives good accuracy at a minimum computation load. Since the input functions (inertial data) of the ODE's algorithm are sampled and no interpolation is done (sample and hold), the high-order methods are only useful to take into account non-linearities present in the equation of motion.

It should also be noted that this way of solving the equations of motion makes use of the velocity rate and the attitude rate information. Most inertial sensors provide only the increments of velocity and attitude. Given such a case, the rate can be retrieved by scaling the output of the sensors by the sample rate. Such an operation will inevitably increase the noise of the data, but that will be compensated for by the double integration nature of the system. In addition, the signal-to-noise ratio is not affected by scaling, so the Kalman-filter accuracy (for error compensation) should not be affected.

Performance Analysis

As with the validation sequence, integration algorithms have been tested using the same trajectory, shown in Appendix A. Two main parameters will be compared, the computation load and the integration solution accuracy. For a first approximation, the computation load can be linked to the off-line simulation time. To roughly evaluate the real-time implementation feasibility, the simulation time will be normalized with respect to the real-trajectory flight time. On the other hand, the solution accuracy is represented by the magnitude of the position error vector (the equivalent spherical error).

Figure 11 shows the normalized simulation time $\overline{T}$ of Simulink's ODE integration suite and Savage's one-speed algorithm. The simulation has been performed on a personal computer with a 933 MHz Pentium III processor and 256 Mb of random access memory (RAM). Of course, computation load increases with increasing data-sample rates, leading to longer simulation time. The portion of the curves below the unity normalized time, $\overline{T} < 1$, represents possible pairs of parameters (integration algorithm and sample rate) for real-time implementation.

A few remarks have to be made at that point. First, Simulink uses an interpreted language compiler. Hence, the auto-generated C code from Simulink RTW will be much faster, between 5 to 10 times. Second, the high sample rates used in the graph are useful for seeing the trend, but are not very realistic for real-life scenarios. Also, it should be noted that, with a slight decrease in numerical accuracy, the computation burden of the two-speed integration algorithm would be less than its one-cycle counterpart. And finally, since the actual implemented algorithm does not include any error compensation by Kalman filtering, further computation load analysis should be performed with an error compensation

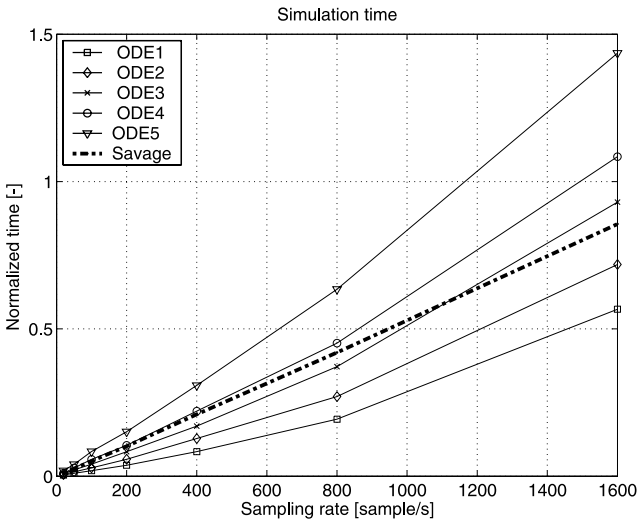

Figure 11. Simulation Time for Simulink ODE Integration Suite.

algorithm implemented to ensure real-time capabilities. Nevertheless, these preliminary results suggest adequate computational load for real-time implementation.

However, ODE's solution accuracy tells more about the integration algorithm effectiveness. From **Figure 12**, it is clear that the Euler method (ODE1) is not suitable for the application. In addition to the integration error of the translational variables, the attitude cosine matrix solution from Equation (10) has to be orthogonalized at nearly every integration cycle, leading to unacceptable position accuracy.

From **Figure 12**, the other integration methods seem to have similar numerical accuracy for the proposed flight profile. However, with a closer look at **Figure 13**, we see that for low-rate inertial data (20 Hz), the accuracy of the RK4 compared to Savage is improved by 8%. This can be partly explained by the

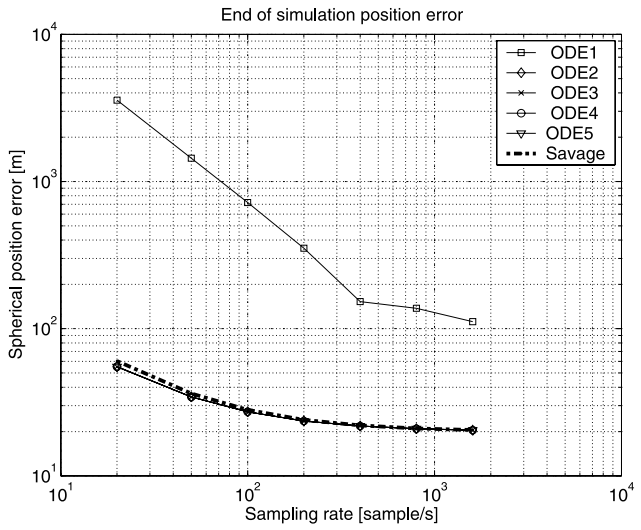

Figure 12. End of Simulation Spherical Position Error for Simulink ODE Integration Suite.

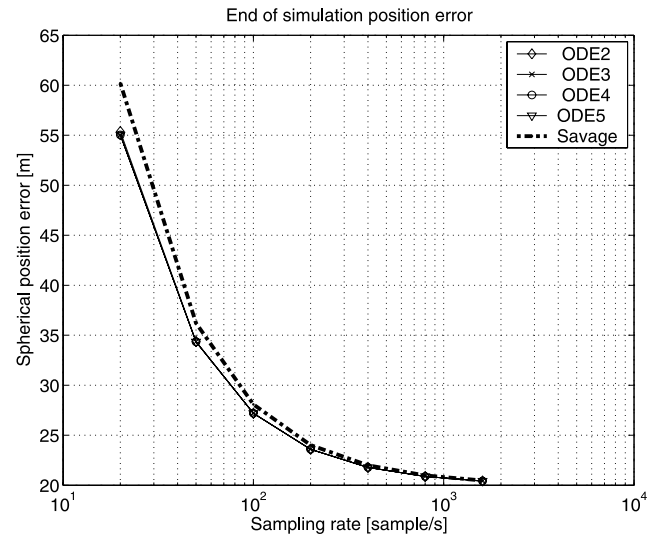

Figure 13. Zoom of the End of Simulation Spherical Position Error for Simulink ODE Integration Suite.

use of a trapezoidal integration algorithm for the moderate-speed part of Savage's algorithm compared with high-order integration. As the sample rate increases, this difference becomes less, as expected.

The previous flight profile does not include important dynamic motion, nor coning and sculling effects. So, to evaluate the effectiveness of the integration algorithm in the presence of high-dynamic motion and the coning effect, a Spin-Cone trajectory profile has been implemented. Details of this kind of trajectory are given in Savage (2000). **Figure 14** illustrates the Spin-Cone spherical position error for different integration schemes.

Again, the Euler first-order method is not practical. **Figure 15** zooms the results at the sample rate of 20 Hz and it shows that the Savage algorithm gives slightly better numerical

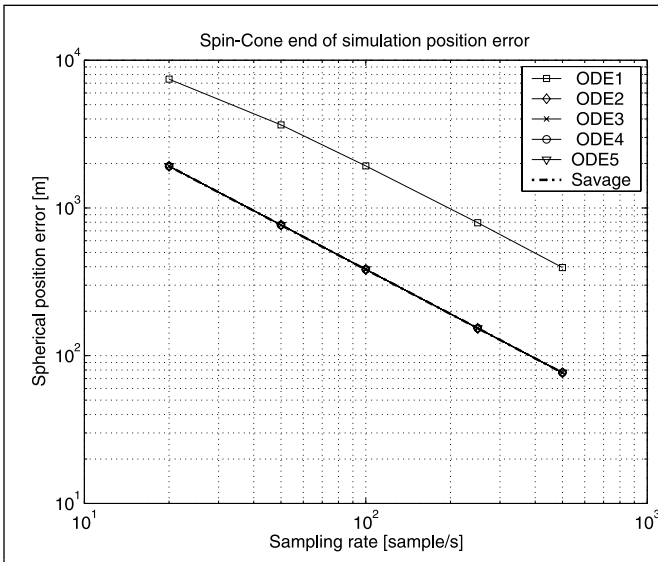

Figure 14. Spin-Cone End of Simulation Spherical Position Error.

accuracy for the Spin-Cone trajectory. In fact, the gain is very small (0.1%) in using Savage's algorithm for conic motion compared to RK4, whereas it does not provide better accuracy than the fourth-order Runge–Kutta (RK4) integration algorithm, in the long run, with a realistic flight profile, as was shown before (**Figure 13**). Hence, the RK4 integration algorithm is suitable for the application discussed herein. Moreover, Chatfield (1997) suggests in a book the use of the RK4 method to perform the integration of the equations of motion.

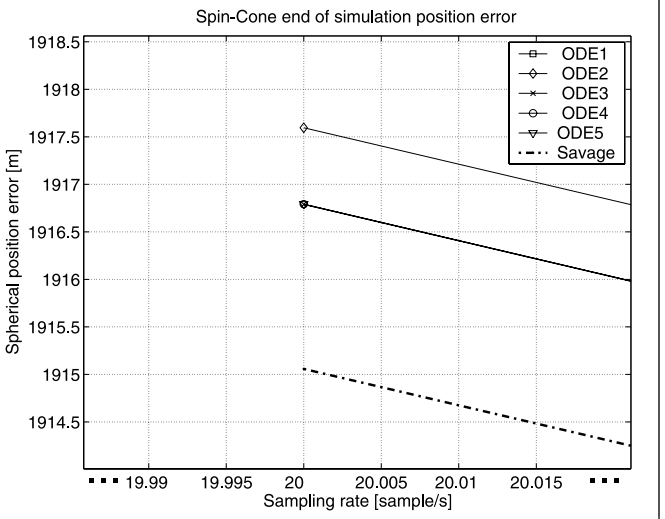

Figure 15. Zoom at 20 Hz Sampling Rate on Spin-Cone End of Simulation Spherical Position Error.

Another rationale for choosing the RK4 algorithm is shown in **Figure 16**. An accepted rule of thumb (Savage, 2000) suggests that the numerical error resulting from the integration algorithm should not be more than 5% of the equivalent error produced by the INS inertial sensors. **Figure 16** shows the 5% spherical position error of several combinations of rate gyro and accelerometer biases, based on the trajectory of Appendix A. This figure also includes the spherical position error for the RK4 integration method as a function of the inertial-data sampling rate. Based on this criterion, it might be assumed that all combinations of sensors above the RK4 curve are possible sets of sensors for an INS using the RK4 as the integration scheme. These include tactical grade inertial sensors and less accurate sensors. The sample rate would be chosen based on the dynamic environment expected.

Conclusion

This paper has addressed two fundamental aspects of inertial navigation integration-algorithm design: the validation of a Simulink simulator, and the performance evaluation of the integration algorithms provided within Simulink for inertial-data integration.

The tools available until now have not been found adequate for our needs and have forced the development of a modular

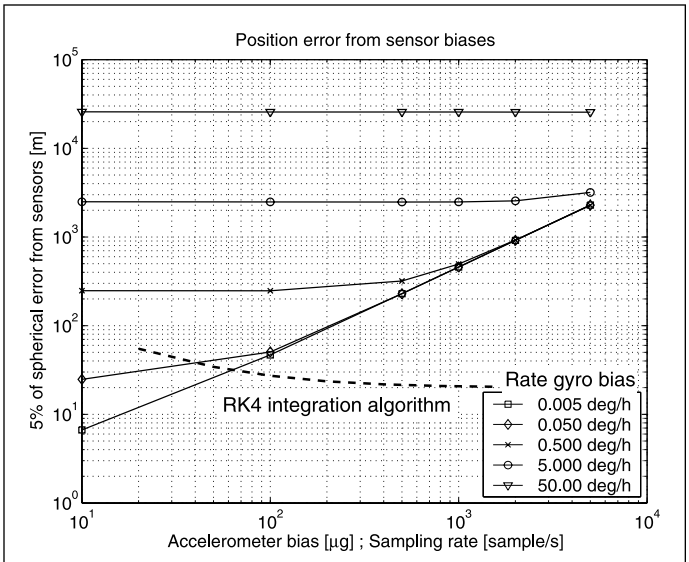

Figure 16. End of Simulation Spherical Position Error from Sensors Bias Compared to RK4 Spherical Position Error.

and easy to reconfigure inertial navigation simulator. The successful two-step validation sequence of the simulator (input–output and by comparison) led to the conclusion that it was robust enough to be used in the design of inertial navigation algorithms.

Simulink-based simulators will certainly be the next generation of INS algorithm validation schemes. It has been shown that the built-in integration methods of Simulink permit the design of inertial integration algorithms with accuracy performance and computation load similar to a state-of-the-art implementation. Even better accuracy than Savage's integration solution (up to 8% improvement) can be achieved at low inertial measurement rates. With this performance evaluation, design of inertial integration algorithms will be very straightforward with the use of a real-time environment compatible with Simulink and the Real-Time Workshop.

The next steps in the simulator development include the use of realistic sensor models (**Figure 3**: sensors model, GPS model) and the use of a state flow-based Extended Kalman filter to estimate the navigation errors. At the end of the development phase, the Simulink simulator and the associated hardware will constitute a powerful tool to make rapid designs and prototyping of low-cost GPS-aided INS.

Acknowledgements

The financial support for this work was provided partly by the Natural Sciences and Engineering Research Council of Canada (NSERC), the "Fonds Nature et Technologie du Québec" (NATEQ), and the École de technologie supérieure (ETS). Also, the authors are very grateful to the National Research Council's Institute for Aerospace Research for providing information regarding its inertial navigation simulator.

References

Biezad, D.J. (1999). "Integrated Navigation and Guidance Systems", AIAA Education Series, Reston, Virginia.

Chatfield, A.B. (1997). "Fundamentals of High Accuracy Inertial Navigation". *Prog. Astronaut. Aeronaut*. Vol. 174.

Eck, C., Chapuis, J., and Geering, H.P. (2001). "Software-Supported Design and Evaluation of Low-Cost Navigation Units". *Proceedings of the 8th St. Petersburg International Conference on Integrated Navigation Systems*, pp. 163–172.

GPSoftNav. (2003). Available from http://www.gpsoftnav.com (accessed April 2003).

Lawrence, A. (1992). "Modern Inertial Technology: Navigation, Guidance and Control", Springer-Verlag, New York, New York.

Leach, B., and Hui, K. (1999). "Low Cost Strapdown Inertial/DGPS Integration for Flight Test Requirements". *Can. Aeronaut. Space J*. Vol. 45, No. 3, pp. 253–263.

Li, X., and Goetze, H.-J. (2001). "Ellipsoid, Geoid, Gravity, Geodesy and Geophysics". *Geophys*. Vol. 66, No. 6, pp. 1600–1668.

Litmanovich, Y.A., Lesyuchevsky, V.M., and Gusinsky, V.Z. (2000). "Two new classes of strapdown navigation algorithms". *J. Guid. Control Dyn*. Vol. 23, No. 1, pp. 34–44.

Otter, M., and Cellier, F.E. (2000). "Software for Modeling and Simulating Control Systems". *Control System Fundamentals*, edited by W.S. Levine. CRC Press, Boca Raton, Florida. pp. 419–432.

Santina, M.S., Stubberud, A.R., and Hostetter, G.H. (2000). "Discrete-Time Equivalents to Continuous-Time Systems". *Control System Fundamentals*, edited by W.S. Levine. CRC Press, Boca Raton, Florida. pp. 267–281.

Savage, P.G. (1998a). "Strapdown Inertial Navigation Integration Algorithm Design Part 1: Attitude Algorithms". *J. Guid. Control Dyn*. Vol. 21, No. 1, pp. 19–28.

Savage, P.G. (1998b). "Strapdown Inertial Navigation Integration Algorithm Design Part 2: Velocity and Position Algorithms". *J. Guid. Control Dyn*. Vol. 21, No. 2, pp. 208–221.

Savage, P.G. (2000). "Strapdown Analytics, Part 1", Strapdown Associates Inc., Maple Plain, Minnesota.

Shampine, L.F., and Reichelt, M.W. (1997). "The Matlab Ode Suite". *SIAM J. Sci. Comput*. Vol. 18, pp. 1–22.

Appendix A. Flight Trajectory for Validation and Performance Evaluation

The flight trajectory is composed of a climbing segment, a cruise segment with turns, and a descent segment. The flight time is 20 min. **Figure A1** gives a general three-dimensional view of the trajectory, while **Figure A2** gives the projection of the trajectory on the North–East plane, and **Figure A3** shows the altitude profile w.r.t flight time.

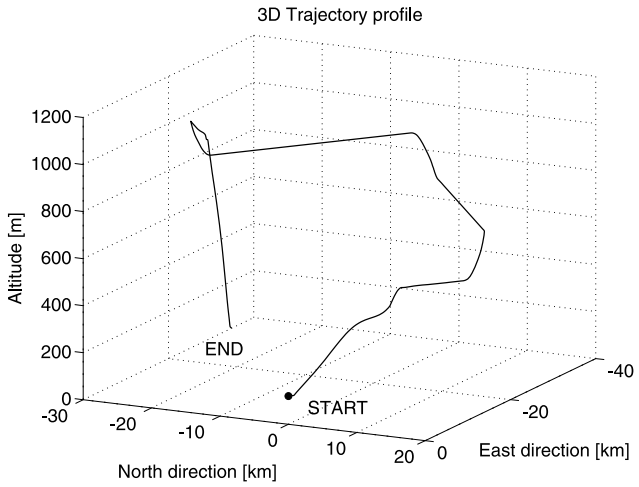

Figure A1. 3D Trajectory.

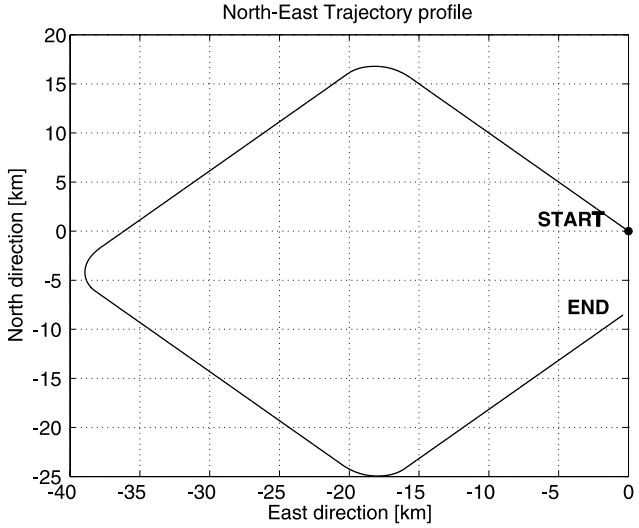

Figure A2. North-East Projection of the Trajectory.

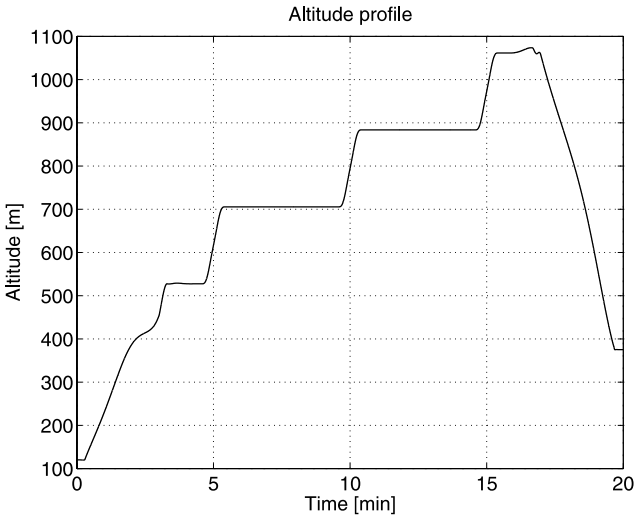

Figure A3. Altitude Profile.